Altova UModel 2023

Manual del usuario y referencia

Altova UModel 2023
Manual del usuario y referencia

Fecha de publicación: 2022

Contenido

1 Introducción 12

1.1 Notas sobre compatibilidad 13

1.2 Compatibilidad con bases de datos 16

2 Tutorial de UModel 17

2.1 Introducción 18

2.2 Casos de uso 21

2.3 Diagramas de clases 32

2.3.1 Crear clases derivadas 41

2.4 Diagramas de objetos 46

2.5 Diagramas de componentes 53

2.6 Diagramas de implementación 59

2.7 Ingeniería directa (del modelo al código) 64

2.8 Ingeniería inversa (del código al modelo) 73

3 Interfaz gráfica del usuario de UModel 80

3.1 Ventana Estructura del modelo 82

3.2 Ventana Árbol de diagramas 86

3.3 Ventana Favoritos 87

3.4 Ventana Propiedades 88

3.5 Ventana Estilos 89

3.6 Ventana Jerarquía 90

3.7 Ventana Vista general 92

3.8 Ventana Documentación 93

3.9 Ventana Capas 94

3.10 Ventana Mensajes 95

3.11 Ventana de diagramas 97

3.12 Panel Diagramas 99

4 Interfaz de la línea de comandos 101

4.1 Crear, cargar y guardar proyectos por lotes ... 106

5 Cómo modelar 108

5.1 Elementos ... 109
- 5.1.1 Crear elementos ... 109
- 5.1.2 Insertar elementos del modelo en un diagrama ... 110
- 5.1.3 Renombrar, mover y copiar elementos ... 112
- 5.1.4 Borrar elementos ... 113
- 5.1.5 Convertir elementos ... 114
- 5.1.6 Buscar y reemplazar texto ... 114
- 5.1.7 Comprobar si se están usando ciertos elementos y dónde ... 116
- 5.1.8 Restricción de elementos ... 117
- 5.1.9 Agregar hipervínculos a elementos ... 119
- 5.1.10 Documentar elementos ... 122
- 5.1.11 Cambiar el estilo de los elementos de un diagrama ... 123

5.2 Diagramas ... 125
- 5.2.1 Crear diagramas ... 125
- 5.2.2 Generar diagramas ... 126
- 5.2.3 Abrir diagramas ... 129
- 5.2.4 Borrar diagramas ... 130
- 5.2.5 Cambiar el estilo de diagramas ... 131
- 5.2.6 Alinear y ajustar el tamaño de elementos de modelado ... 132
- 5.2.7 Añadir capas a los diagramas ... 134
- 5.2.8 Finalización automática en clases ... 136
- 5.2.9 Acercar y alejar diagramas ... 138

5.3 Relaciones ... 139
- 5.3.1 Crear relaciones entre elementos ... 139
- 5.3.2 Cambiar el estilo de las líneas y relaciones ... 141
- 5.3.3 Ver las relaciones de los elementos ... 142
- 5.3.4 Associations ... 143
- 5.3.5 Asociación de colecciones ... 146

5.3.6 Contención ... 149
5.4 Estereotipos y valores etiquetados ... 150
5.4.1 Valores etiquetados ... 151
5.4.2 Aplicar estereotipos ... 152
5.4.3 Mostrar u ocultar valores etiquetados ... 154

6 Proyectos e ingeniería de código 157

6.1 Administrar proyectos de UModel ... 158
6.1.1 Crear, abrir y guardar proyectos ... 158
6.1.2 Abrir proyectos desde una URL ... 160
6.1.3 Mover proyectos a un directorio nuevo ... 163
6.1.4 Aplicar perfiles de UModel ... 164
6.1.5 Dividir proyectos de UModel ... 165
6.1.6 Incluir otros proyectos de UModel ... 168
6.1.7 Compartir paquetes y diagramas ... 169
6.1.8 Consejos para mejorar el rendimiento ... 172
6.2 Generar código de programa ... 173
6.2.1 Definir un paquete como raíz de espacio de nombres ... 173
6.2.2 Agregar un componente de ingeniería de código ... 174
6.2.3 Revisar la sintaxis del proyecto ... 176
6.2.4 Opciones de generación de código ... 179
6.2.5 Ejemplo: generar código C# ... 180
6.2.6 Ejemplo: generar código Java desde UModel ... 186
6.2.7 Ejemplo: generar código C++ ... 195
6.2.8 Plantillas SPL ... 200
6.3 Importar código fuente ... 202
6.3.1 Ingeniería inversa en código C++ ... 204
6.3.2 Opciones de importación de código ... 205
6.3.3 Ejemplo: importar un proyecto C# ... 210
6.4 Importar binarios Java, C# y VB.NET ... 217
6.4.1 Añadir tiempos de ejecución Java personalizados ... 218
6.4.2 Opciones de importación de tipos binarios ... 219
6.4.3 Ejemplo: importar ensamblados del .NET ... 222
6.4.4 Ejemplo: importar archivos Java .class ... 225

6.5 Sincronizar el modelo y el código fuente ... 231
6.5.1 Consejos prácticos ... 232
6.5.2 Refactorización de código y sincronización ... 234
6.5.3 Configurar la sincronización del código ... 235
6.6 Correspondencias con elementos de UModel ... 238
6.6.1 Correspondencias con C# ... 238
6.6.2 Correspondencias con VB.NET ... 258
6.6.3 Correspondencias con Java ... 272
6.6.4 Correspondencias con XML Schema ... 278
6.6.5 Correspondencias con elementos de BD ... 287
6.7 Combinar proyectos de UModel ... 290
6.7.1 Fusión de proyectos a tres bandas ... 291
6.7.2 Ejemplo de fusión manual a tres bandas ... 292
6.8 Plantillas UML ... 296
6.8.1 Firmas de plantilla ... 297
6.8.2 Enlace de plantilla ... 298
6.8.3 Usar plantillas en operaciones y propiedades ... 299

7 Transformar modelos UML 300

7.1 Configuración de la transformación ... 303
7.2 Ejemplo: transformar un modelo Java en un modelo C++ ... 305
7.3 Transformar un modelo C# en un modelo Java ... 312
7.4 Ejemplo: convertir la estructura de una BD Access en SQLite ... 318

8 Generar documentación UML 327

8.1 Con una hoja de estilos SPS predeterminada ... 331
8.2 Con hojas de estilos predefinidas por el usuario ... 336

9 Diagramas UML 338

9.1 Diagramas de comportamiento ... 339
9.1.1 Diagrama de actividades ... 339
9.1.2 Diagrama de máquina de estados ... 356
9.1.3 Diagrama de máquina de estados de protocolos ... 380

9.1.4 Diagrama de casos de uso ... 385
9.1.5 Diagrama de comunicación ... 385
9.1.6 Diagrama global de interacción ... 389
9.1.7 Diagrama de secuencia ... 394
9.1.8 Diagrama de ciclo de vida ... 424
9.2 Diagramas de estructura ... 433
9.2.1 Diagrama de clases ... 433
9.2.2 Diagrama de estructura compuesta ... 451
9.2.3 Diagrama de componentes ... 454
9.2.4 Diagrama de implementación ... 454
9.2.5 Diagrama de objetos ... 455
9.2.6 Diagrama de paquetes ... 456
9.2.7 Diagrama de perfil ... 461
9.3 Otros diagramas ... 474
9.3.1 Diagramas de esquema XML ... 474
9.3.2 Diagramas BPMN 1.0 / 2.0 ... 492
9.3.3 Diagramas SysML ... 520

10 Trabajar con bases de datos en UModel 538

10.1 Modelar bases de datos con UModel ... 540
10.1.1 Importar bases de datos SQL en UModel ... 541
10.1.2 Diseñar objetos de base de datos ... 548
10.1.3 Configurar la ingeniería de ida y vuelta para bases de datos ... 553
10.1.4 Ejemplo: actualizar una BD desde el modelo ... 554
10.2 Conectarse a un origen de datos ... 560
10.2.1 Iniciar el asistente para la conexión de base de datos ... 561
10.2.2 Resumen de controladores de base de datos ... 563
10.2.3 Conexiones ADO ... 566
10.2.4 Conexiones ADO.NET ... 571
10.2.5 Conexiones ODBC ... 578
10.2.6 Conexiones JDBC ... 581
10.2.7 Conexiones PostgreSQL ... 586
10.2.8 Conexiones SQLite ... 587
10.2.9 Ejemplos de conexión a bases de datos ... 588

11 XMI: intercambio de metadatos XML 643

12 Complemento de UModel para Visual Studio 645

12.1 Instalar el complemento de UModel para Visual Studio ... 647
12.2 Agregar funciones de UModel a proyectos de Visual Studio ... 648
12.3 Cargar y descargar proyectos de UModel ... 652
12.4 Sincronización de modelo y código ... 653

13 Complemento de UModel para Eclipse 656

13.1 Instalar el complemento de UModel para Eclipse ... 659
13.2 La perspectiva UModel ... 661
13.3 Agregar funciones de UModel a proyectos de Eclipse ... 664
13.4 Importar proyectos de UModel ... 666
13.5 Cargar y descargar proyectos de UModel ... 668
13.6 Funcionamiento de la sincronización automática ... 669
13.7 Ejemplo: configurar la sincronización automática ... 670

14 Control de código fuente 682

14.1 Sistemas de control de código fuente compatibles ... 684
14.2 Comandos de control de código fuente ... 686
14.2.1 Abrir desde el control de código fuente ... 686
14.2.2 Habilitar control de código fuente ... 689
14.2.3 Obtener la versión más reciente ... 690
14.2.4 Obtener ... 691
14.2.5 Obtener carpetas ... 692
14.2.6 Desproteger ... 693
14.2.7 Proteger ... 695
14.2.8 Anular desprotección ... 696
14.2.9 Agregar al control de código fuente ... 697
14.2.10 Quitar del control de código fuente ... 700
14.2.11 Compartir desde el control de código fuente ... 701

14.2.12 Mostrar historial ... 702
14.2.13 Mostrar diferencias ... 704
14.2.14 Mostrar propiedades ... 706
14.2.15 Actualizar estado ... 706
14.2.16 Administrador del control de código fuente ... 706
14.2.17 Cambiar control de código fuente ... 707
14.3 Control de código fuente con Git ... 708
14.3.1 Habilitar Git con el complemento de control de código fuente ... 709
14.3.2 Agregar un proyecto al control de código fuente de Git ... 709
14.3.3 Clonar un proyecto desde el control de código fuente de Git ... 711

15 Iconos en los diagramas de UModel 713

15.1 Diagramas de actividades ... 714
15.2 Diagramas de clases ... 716
15.3 Diagramas de comunicación ... 717
15.4 Diagramas de estructura de un compuesto ... 718
15.5 Diagramas de componentes ... 719
15.6 Diagramas de implementación ... 720
15.7 Diagramas global de interacción ... 721
15.8 Diagramas de objetos ... 722
15.9 Diagramas de paquetes ... 723
15.10 Diagramas de perfil ... 724
15.11 Diagramas de máquina de estados de protocolos ... 725
15.12 Diagramas de secuencia ... 726
15.13 Diagramas de máquina de estados ... 727
15.14 Diagramas de ciclo de vida ... 728
15.15 Diagramas de casos de uso ... 729
15.16 Diagramas de esquema XML ... 730
15.17 Diagramas BPMN ... 731
15.18 Diagramas BPMN 2.0 ... 732
15.19 Modelado de bases de datos ... 733

16 Referencia del usuario 734

16.1 Menú Archivo ... 735
16.2 Menú Edición ... 737
16.3 Menú Proyecto ... 739
16.4 Menú Diseño ... 742
16.5 Menú Vista ... 744
16.6 Menú Herramientas ... 745
16.6.1 Ortografía ... 745
16.6.2 Opciones de ortografía ... 749
16.6.3 Editor de scripts ... 751
16.6.4 Macros ... 751
16.6.5 Herramientas definidas por el usuario ... 751
16.6.6 Personalizar ... 751
16.6.7 Restaurar barras de herramientas y ventanas ... 762
16.6.8 Opciones ... 762
16.7 Menú Ventanas ... 773
16.8 Menú Ayuda ... 775

17 Referencia del programador 780

17.1 Notas sobre la versión ... 782
17.2 Editor de scripts ... 788
17.2.1 Crear un proyecto de scripting ... 789
17.2.2 Comandos integrados ... 802
17.2.3 Habilitar scripts y macros ... 812
17.3 Complementos para entornos IDE ... 815
17.3.1 Cómo crear un complemento IDE para UModel ... 815
17.3.2 Implementación de complementos para IDE en UModel ... 824
17.3.3 XML de configuración ... 826
17.3.4 Complementos como controles ActiveX ... 829
17.3.5 Interfaz IUModelPlugIn ... 830
17.4 The UModel API ... 834
17.4.1 Accessing the API ... 834
17.4.2 Object Model ... 835
17.4.3 How to ... 841
17.4.4 C# API Examples ... 853

17.4.5 Java API Example ... 879
17.4.6 JScript Examples ... 880
17.5 UModel API Reference ... 895
17.5.1 UModel Plug-Ins ... 895
17.5.2 UModel API Interfaces ... 897
17.5.3 UMLData Interfaces ... 985

18 SPL: el lenguaje de programación Spy 1348
18.1 Estructura básica de SPL ... 1349
18.2 Variables ... 1350
18.3 Operadores ... 1358
18.4 Condiciones ... 1359
18.5 Colecciones y foreach ... 1360
18.6 Subrutinas ... 1362
18.6.1 Declaración de subrutinas ... 1362
18.6.2 Invocación de subrutinas ... 1363

19 Información sobre licencias 1364
19.1 Distribución electrónica de software ... 1365
19.2 Activación del software y medición de licencias ... 1366
19.3 Contrato de licencia para el usuario final ... 1368

Índice 1369

1 Introducción

Sitio web de Altova: Herramienta UML

Altova UModel 2023 es una herramienta de modelado UML con una potente interfaz gráfica y avanzadas funciones que le facilitarán el trabajo con UML. Además incluye funciones para trabajar con los aspectos más prácticos de la especificación 2.5. UModel es una aplicación de 32/64 bits para Windows compatible con Windows 7 SP1 con actualización de la plataforma, Windows 8, Windows 10, Windows 11 y Windows Server 2008 R2 SP1 con actualización de la plataforma o superior. Las ediciones Enterprise y Professional son compatibles con plataformas de 64 bits. Para más información consulte las Notas sobre compatibilidad.

Última actualización: 10 October 2022

1.1 Notas sobre compatibilidad

UModel es una aplicación Windows de 32/64 bits compatible con estos sistemas operativos:

- Windows Server 2008 R2 SP1 con actualización de la plataforma o superior
- Windows 7 SP1 con actualización de la plataforma, Windows 8, Windows 10, Windows 11

Solamente las ediciones Enterprise y Professional Edition están disponibles como aplicación de 64 bits.

Diagramas UML

UModel es compatible con los catorce diagramas de la especificación UML 2.5.1, además de con otros tipos de diagramas especializados.

De estructura	De comportamiento	Otros
Diagramas de clase	Diagramas de actividades	Diagramas de esquema XML
Diagramas de componentes	Diagramas de comunicación	Diagramas BPMN (Business Process Modeling Notation) 1.0 / 2.0 (*Ediciones UModel Enterprise y Professional*)
Diagramas de estructura compuesta	Diagramas globales de interacción	Diagramas SysML 1.2, 1.3, 1.4, 1.5, 1.6 (*Ediciones UModel Enterprise y Professional*)
Diagramas de implementación	Diagramas de secuencia	Diagramas de bases de datos* (*Ediciones UModel Enterprise y Professional*)
Diagramas de objetos	Diagramas de máquina de estados (y diagramas de máquina de estados de protocolo)	
Diagramas de paquetes	Diagramas de ciclo de vida	
Diagramas de perfil	Diagramas de casos de uso	

UModel está diseñado para ofrecer flexibilidad total durante el proceso de modelado:

- Los diagramas de UModel se pueden crear en el orden que se quiera y en cualquier momento. No es necesario seguir un orden determinado durante el modelado.
- Color de sintaxis en diagramas para poder trabajar de forma más intuitiva. Los elementos de modelado y sus propiedades (fuente, color, borde, etc.) se pueden personalizar de forma jerárquica por proyectos, por nodos/líneas, por familias de elementos o por elementos (*véase Cambiar el estilo de los elementos de un diagrama*).
- Las operaciones de Deshacer/Rehacer son ilimitadas y no solo abarcan cambios en el contenido sino también cambios de estilo realizados en los elementos del modelo.
- Opción para crear hipervínculos entre diagramas y elementos de modelado.
- Puede crear varias capas en un mismo diagrama UML, (*consulte Agregar capas a los diagramas*).

Ingeniería de código e importación de archivos binarios

Las funciones de generación de código y de ingeniería inversa son compatibles con estos lenguajes:

Lenguaje	Ingeniería de código	Importar binarios
C#	1.2, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 7.1, 7.2, 7.3, 8.0, 9.0[1], 10	Mismas versiones del lenguaje que para la ingeniería de código[2]
C++ (*Edición UModel Enterprise*)	C++98, C++11 and C++14, C++17, C++20 En C++20 la compatibilidad solo es parcial, ya que no se admiten módulos.	No aplica
Java	1.4, 5.0 (1.5), 6 (1.6), 7 (1.7), 8 (1.8), 9 (1.9), 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	Mismas versiones del lenguaje que para la ingeniería de código[3]
Visual Basic .NET	7.1 o superior	Mismas versiones del lenguaje que para la ingeniería de código
XML Schemas[4]	1.0	No aplica
Databases[5] (*Ediciones UModel Enterprise y Professional*)	Consulte Compatibilidad con bases de datos.	No aplica

Notas de la tabla:

1. Si importa archivos binarios compilados con código C# 9.0, los *registros* se importan como *clases*. Esta limitación se debe a que los registros están marcados como clases en el ensamblado, por lo que son indistinguibles de las clases.
2. La ingeniería de código en C# y la importación de archivos binarios son compatibles con .NET Framework, .NET Core, .NET 5 y .NET 6. Debe instalar el que corresponda. Los archivos binarios de otras implementaciones .NET que no mencionamos aquí probablemente también se importen. Consulte también Importar archivos binarios Java, C# y VB.NET.
3. También se pueden importar archivos binarios que apunten a equipos virtuales Java distintos a Oracle JDK, como OpenJDK, SapMachine, Liberica JDK, ect. Consulte Añadir tiempos de ejecución Java personalizados.
4. En el caso de los esquemas XML, la ingeniería de código permite importar un esquema (o varios esquemas de un directorio) en UModel, visualizar o modificar el modelo y pasar esos cambios al archivo del esquema. Al sincronizar los datos de ambos, el modelo siempre sobrescribe al archivo de esquema. Consulte también Diagramas de esquema XML.
5. En el caso de las bases de datos, la ingeniería de código permite (i) modelar una BD en UModel con la opción de actualizarla mediante un script generado a partir del modelo o (ii) importar una estructura de BD que ya existe en un modelo, realizar cambios en él y después implementar un script generado a partir del modelo en la BD. Algunos tipos de objetos de BD no son compatibles con el modelado, consulte Trabajar con bases de datos en UModel.

Notas:

- La actualización/combinación del código o del modelo se puede hacer por proyectos, por paquetes o por clases. Para poder realizar ingeniería de ida y vuelta en UModel no hace falta tener pseudocódigo ni que el código generado incluya comentarios.
- Cada proyecto puede ser compatible con Java, C#, C++ y VB al mismo tiempo.
- En UModel se pueden usar plantillas UML y sus asignaciones a o desde genéricos Java, C# y Visual Basic.
- Umodel incluye compatibilidad para la arquitectura dirigida por modelos (MDA por sus siglas en inglés), que permite cambiar el lenguaje de programación de los modelos, por ejemplo, de Java a C# o viceversa (*véase Transformar modelos UML*).
- Al importar código fuente tiene la posibilidad de generar diagramas de clases y paquetes. Una vez que se ha importado el código fuente en el modelo también puede generar diagramas de secuencia.
- Puede generar código a partir de diagramas de secuencia y diagramas de máquina de estados.
- Los proyectos de UModel se pueden dividir en varios subproyectos, lo que permite a los programadores editar distintas partes de un mismo proyecto al mismo tiempo. Al terminar puede reintegrar todos los cambios en un modelo común. También puede combinar proyectos de UModel a dos y tres bandas (*véase Combinar proyectos de UModel*).
- En UModel, la generación de código se realiza sobre plantillas SPL y, por tanto, se puede personalizar.

Generación de documentación UML

Puede generar documentación de proyectos UModel en formato HTML, RTF, Microsoft Word 2000 o superior. Existen varias opciones que permiten configurar el nivel de detalle de la documentación que se va a generar, su apariencia y otros aspectos. Con Altova StyleVision (https://www.altova.com/stylevision) también puede generar documentación en formato PDF y personalizar en detalle las plantillas de generación de documentos. Para más información consulte Generar documentación UML.

Integración IDE

UModel también puede utilizarse como complemento para estos entornos de desarrollo integrado:

- Visual Studio 2012/2013/2015/2017/2019/2022 (*véase Complemento de UModel para Visual Studio*).
- Eclipse 2022-09, 2022-06, 2022-03, 2021-12 (*véase Complemento de UModel para Eclipse*).

UModel ofrece una API basada en COM y también permite integrar complementos para entornos IDE (bibliotecas DLL) en su interfaz gráfica del usuario. El Editor de scripts permite desarrollar scripts VBScript o JScript personalizados y macros para automatizar varias tareas.

Integración con Microsoft Office

Gracias a su compatibilidad para el modelado de bases de datos, UModel puede importar bases de datos de Access en un modelo y generar scripts SQL para estas. Para más información consulte UModel and Databases.

Interoperabilidad

UModel también ofrece compatibilidad para importar o exportar proyectos de UModel hacia o desde el formato de intercambio de metadatos XML (XMI), (*véase XMI: intercambio de metadatos XML*).

1.2 Compatibilidad con bases de datos

Las bases de datos compatibles y sus objetos raíz aparecen a continuación. Altova procura ofrecer compatibilidad con otras bases de datos, pero sólo se garantiza una conexión y un procesamiento correctos con las bases de datos de la lista. Si usa la versión de 64 bits de UModel, compruebe que tiene acceso a los controladores de BD de 64 bits de la BD a la que quiere conectarse.

Base de datos	Observaciones
Firebird 2.x, 3.x	
IBM DB2 8.x, 9.x, 10.x, 11.x	
IBM Db2 for i 6.x, 7.4	Los archivos lógicos son compatibles y se muestran en vistas.
IBM Informix 11.70 y superior	
MariaDB 10 y superior	
Microsoft Access 2003 y superior	En el momento de escribir esta documentación (principios de septiembre de 2019) no hay ningún Microsoft Access Runtime disponible para Access 2019. Solo puede conectarse a la BD de Access 2019 con productos de Altova si tiene instalado Microsoft Access 2016 Runtime y solamente si la BD no usa el tipo de datos "Large Number" (número grande).
Microsoft Azure SQL Database	SQL Server 2016 codebase
Microsoft SQL Server 2005 y superior Microsoft SQL Server para Linux	
MySQL 5 y superior	
Oracle 9i y superior	
PostgreSQL 8 y superior	Son compatibles todas las conexiones PostgreSQL, tanto nativas como basadas en controladores, a través de interfaces como ODBC o JDBC. Las conexiones nativas no necesitan controladores.
Progress OpenEdge 11.6	
SQLite 3.x	Las conexiones SQLite son conexiones nativas y directas compatibles con el archivo de base de datos de SQLite. No se precisan controladores separados.
Sybase ASE 15, 16	
Teradata 16	

2 Tutorial de UModel

Este tutorial explica cómo crear varios tipos de diagramas UML con UModel y le ayudará a familiarizarse con la interfaz gráfica del usuario. También aprenderá a generar código a partir de un modelo UML (ingeniería directa) y a importar código a un modelo UML (ingeniería inversa). Con respecto a la ingeniería de código, también aprenderá a realizar ingeniería de ida y vuelta (del modelo al código al modelo o del código al modelo al código). Para entender el tutorial es necesario tener conocimientos básicos de UML.

El tutorial está dividido en varios apartados. En los primeros apartados del tutorial se trabaja con un proyecto de muestra que se instala con UModel. Si desea crear un proyecto de modelado nuevo desde cero y rápidamente, consulte el apartado Ingeniería directa (del modelo al código).

- Introducción
- Casos de uso
- Diagramas de clases
- Crear clases derivadas
- Diagramas de objetos
- Diagramas de componentes
- Diagramas de implementación
- Ingeniería directa (del modelo al código)
- Ingeniería inversa (del código al modelo)

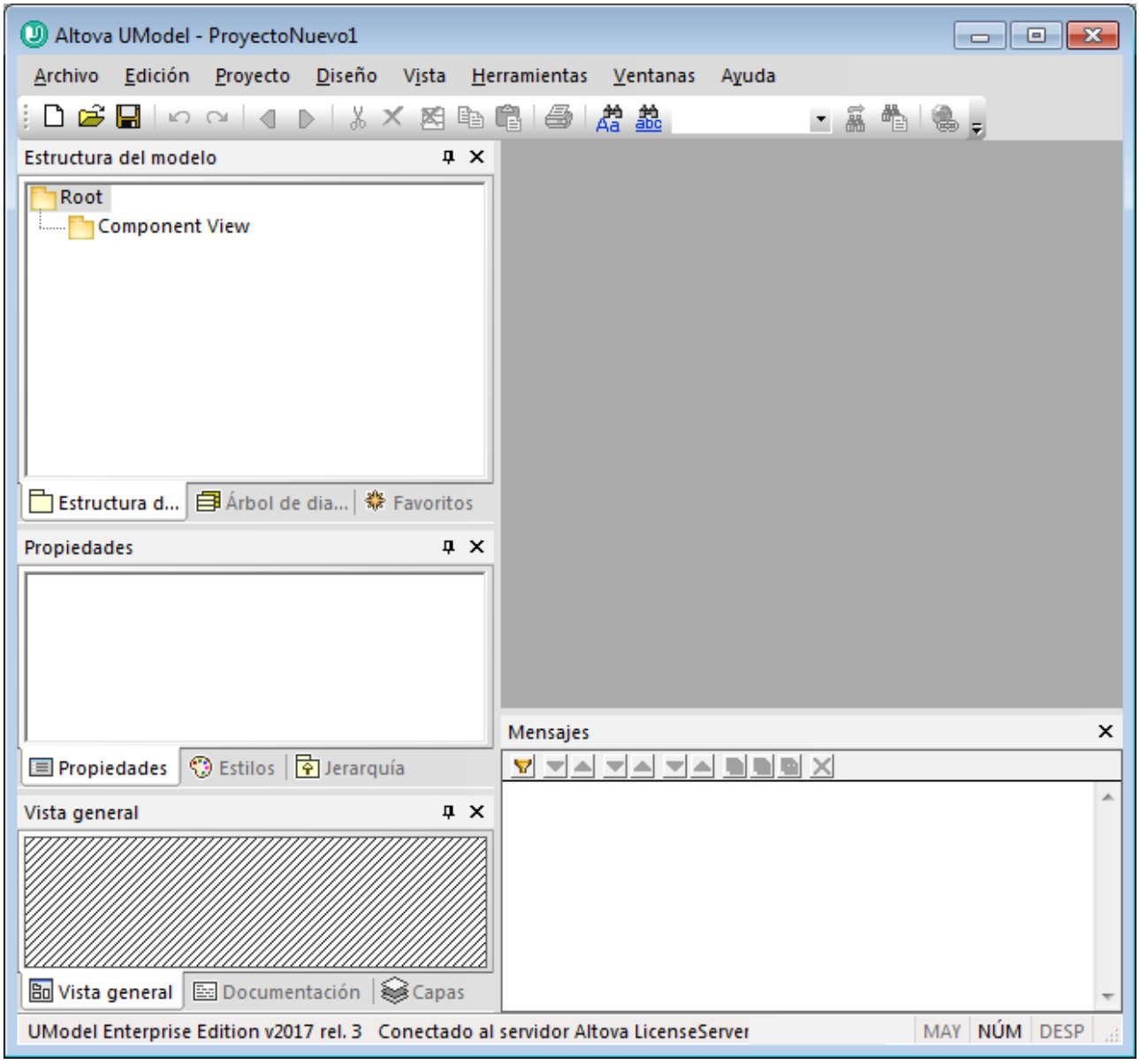
Para el tutorial se utilizan dos archivos de ejemplo que vienen con UModel y que están en el directorio **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Tutorial**:

BankView-start.ump	Se trata del archivo de proyecto de UModel que representa el estado inicial del tutorial. En este proyecto existen varios diagramas de modelado, así como clases, objetos y otros elementos de modelado. A lo largo del tutorial se añadirán elementos y diagramas nuevos y se editarán elementos del proyecto. **Nota:** el proyecto está deliberadamente incompleto, por lo que si usa el comando **Proyecto | Revisar la sintaxis del proyecto** aparecerán errores de validación y advertencias. El tutorial explica cómo solucionar estos errores.
BankView-finish.ump	Se trata del archivo de proyecto de UModel que representa el estado final del tutorial.

Nota: todos los archivos de ejemplo de UModel están en el directorio **C:\ProgramData\Altova\UModel2023**. Cuando se inicia la aplicación por primera vez se crea una copia de esos archivos de ejemplo en el directorio **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**. Por esta razón recomendamos que evite mover, editar o eliminar los archivos de ejemplo del directorio inicial.

2.1 Introducción

Cuando se inicia por primera vez, UModel se abre con un proyecto vacío predeterminado llamado `ProyectoNuevo1`. En las siguientes ejecuciones UModel se iniciará con el último proyecto que estuviera cargado. Para crear, abrir y guardar proyectos de UModel (archivos .ump) puede usar los comandos de Windows estándar del menú **Archivo** o los botones de la barra de herramientas.



UModel, interfaz gráfica del usuario

Observe que la interfaz del usuario se distingue por sus numerosas ventanas de ayuda situadas en el lateral izquierdo y por la ventana de diseño situada a la derecha. En la ventana *Estructura del modelo* pueden verse los dos paquetes predeterminados: "Root" y "Component View". Estos dos paquetes no se pueden eliminar ni renombrar.

La ventana superior izquierda ofrece varias vistas del proyecto de modelado:

- el panel *Estructura del modelo* muestra todos los elementos de modelado del proyecto de UModel. Los elementos se pueden manipular directamente en esta pestaña usando las teclas de edición estándar y operaciones de arrastrar y colocar.

- el panel *Árbol de diagramas* ofrece acceso rápido a los diagramas que componen el proyecto, independientemente de su posición en la estructura del proyecto. Los diagramas aparecen agrupados por tipo.

- el panel *Favoritos* es un repositorio de elementos de modelado que el usuario puede personalizar. En esta pestaña puede colocar todo tipo de elementos de modelado haciendo clic en el comando **Agregar a favoritos** del menú contextual.

La ventana central izquierda ofrece varias vistas de determinadas propiedades del modelo:

- el panel *Propiedades* muestra las propiedades del elemento que está seleccionado en la *Estructura del modelo* o en el *Árbol de diagramas*. Aquí puede definir y actualizar las propiedades de los elementos.

- el panel *Estilos* muestra los atributos de los diagramas o de los elementos que están visibles en el panel *Diagramas*. Estos atributos de estilo se dividen en dos categorías: formato y presentación.

- el panel *Jerarquía* muestra todas las relaciones que tiene el elemento de modelado seleccionado. El elemento de modelado se puede seleccionar en el diagrama o en las pestañas *Estructura del modelo* o *Favoritos*.

La ventana inferior izquierda está compuesta por:

- el panel *Vista general*, que muestra la estructura general del diagrama activo.

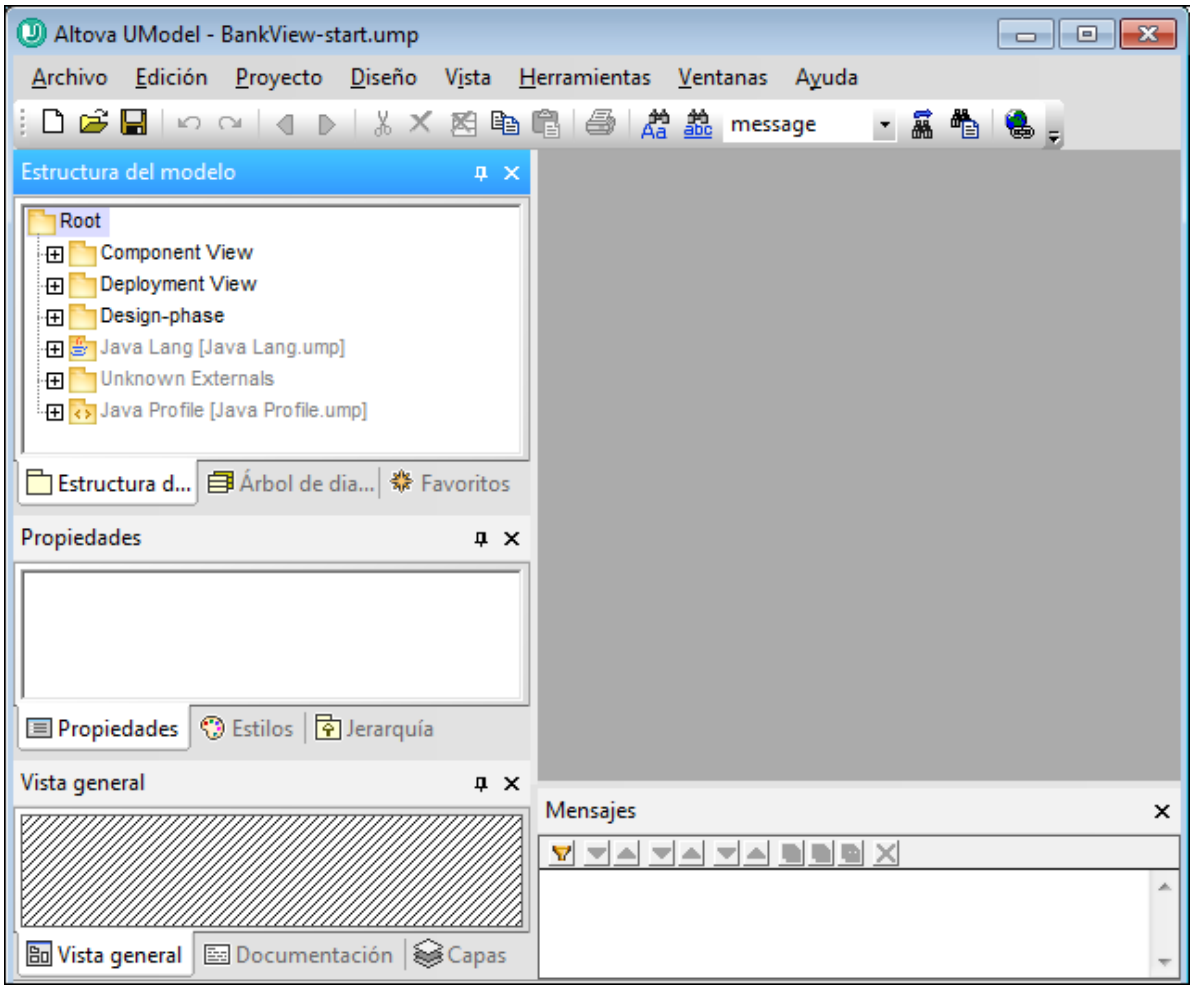
- el panel *Documentación*, que sirve para documentar las clases una a una.

- el panel *Capas*, que sirve para definir las distintas capas de un diagrama, que pueden estar visibles, bloqueadas u ocultas. Las capas sirven para crear agrupaciones lógicas de elementos de modelado en un diagrama.

En este tutorial trabajaremos principalmente en los paneles *Estructura del modelo* y *Árbol de diagramas* y, por supuesto, en la ventana principal. Para más información consulte el apartado Interfaz del usuario.

Para abrir el proyecto del tutorial:

1. Seleccione la opción de menú **Archivo | Abrir** y navegue hasta la carpeta ... \UModelExamples\Tutorial de UModel. Recuerde que también puede abrir archivos *.ump desde una URL (haciendo clic en el botón **Cambiar a URL**).
2. Abra el archivo de proyecto **BankView-start.ump**.
   El archivo de proyecto se carga en UModel. Bajo el paquete Root aparecen ahora varios paquetes predefinidos. Observe que por ahora la ventana principal sigue vacía.



Proyecto BankView-start.ump

2.2 Casos de uso

Este apartado del tutorial explica cómo crear un diagrama de casos de uso y permite familiarizarse con conceptos básicos de la interfaz gráfica del usuario de UModel. Más concretamente, en este tutorial aprenderá a:

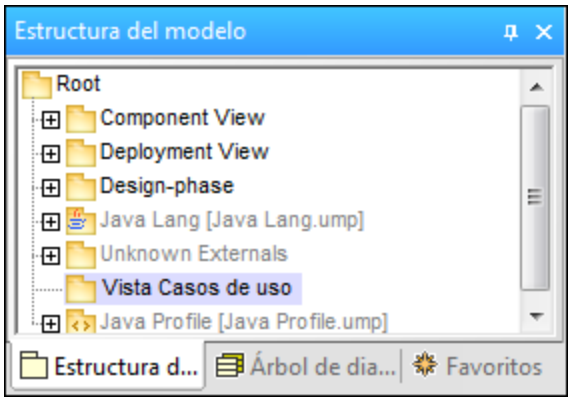
- agregar un **paquete** nuevo al proyecto;
- agregar un **diagrama** de casos de uso nuevo al proyecto;
- agregar **elementos** al diagrama y definir dependencias entre ellos;
- alinear los elementos y ajustar su tamaño;
- cambiar el estilo de todos los diagramas de un proyecto de UModel.

Para continuar debe iniciar UModel y abrir el proyecto **BankView-start.ump** (*véase Abrir el proyecto del tutorial*).

Agregar un paquete nuevo a un proyecto

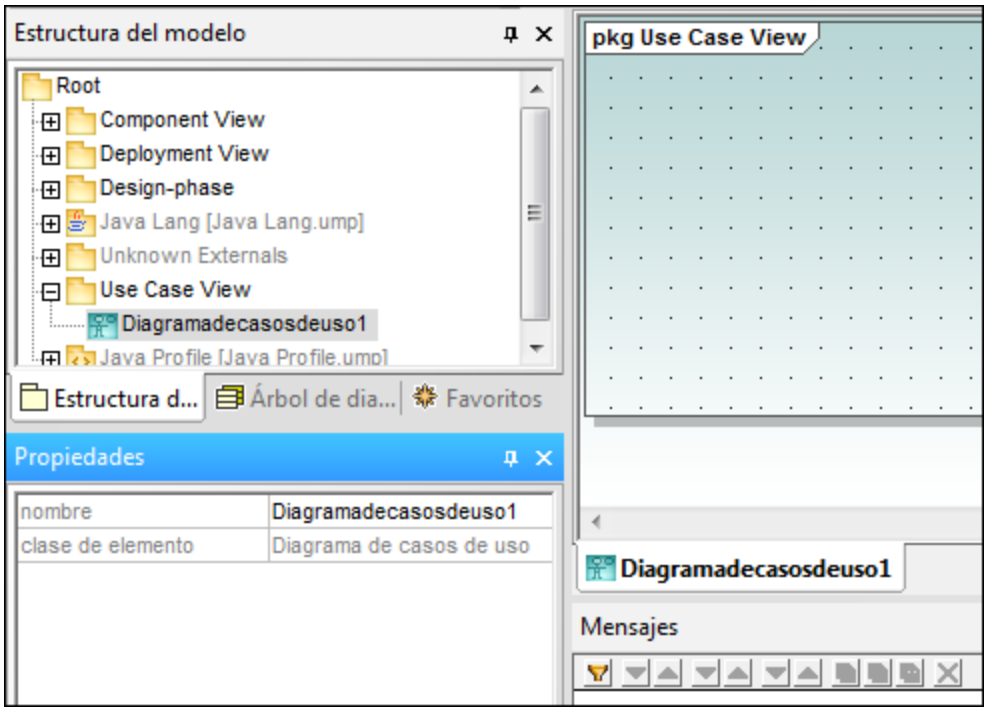
Como ya sabe, un paquete es un contenedor que sirve para organizar clases y otros elementos UML, incluidos los casos de uso. Empezaremos por crear un paquete que contendrá un diagrama de casos de uso nuevo. Aunque UModel no exige que un diagrama en concreto deba encontrarse en un paquete específico, sí se recomienda organizar los diagramas en paquetes.

1. En la ventana *Estructura del modelo* haga clic con el botón derecho en el paquete Root y seleccione **Elemento nuevo | Paquete**.
2. Escriba el nombre del paquete nuevo (p. ej. Vista Casos de uso) y pulse **Entrar**.



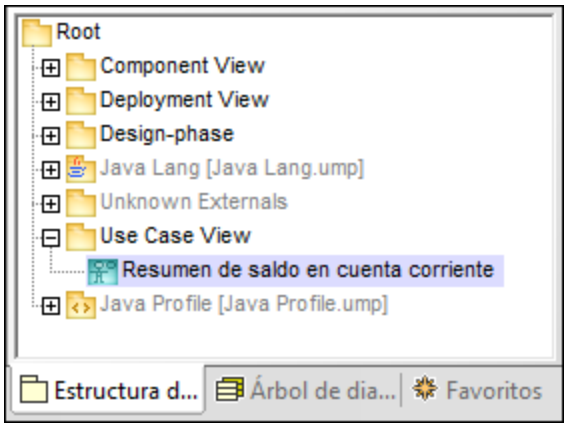
Agregar un diagrama de caso de uso a un proyecto

1. Haga clic con el botón derecho en el paquete Vista Casos de uso que acaba de crear.
2. Seleccione el comando **Diagrama nuevo | Diagrama de casos de uso**.



Ahora se ha añadido al paquete un diagrama de casos de uso (en la *Estructura del modelo*) y en el panel *Diagramas* aparece la pestaña del diagrama que acaba de crear, al que se ha asignado automáticamente un nombre predeterminado.

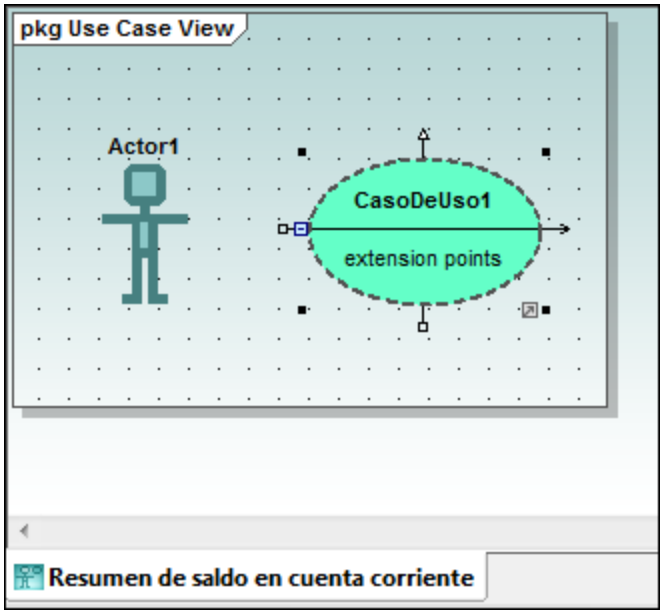
3. En la *Estructura del modelo* haga doble clic en el nombre del diagrama, escriba Resumen de saldo en cuenta corriente y pulse **Entrar** para confirmar.



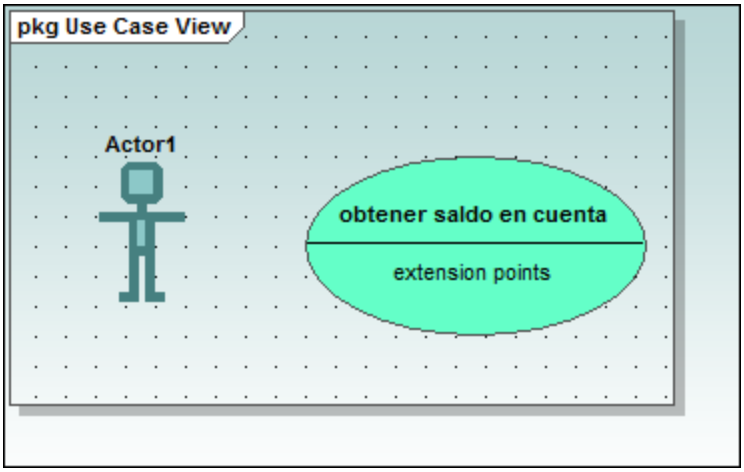
Agregar elementos al diagrama de casos de uso

1. Haga clic con el botón derecho dentro de la ventana del diagrama recién creado y seleccione **Nuevo/a | Actor**. El elemento actor se inserta en el lugar donde se hizo clic.
2. Haga clic en el icono **Caso de uso**  de la barra de herramientas y después en la ventana del diagrama para insertar el elemento. El elemento `CasoDeUso1` se inserta en el diagrama. Observe que

el elemento y su nombre están seleccionados, por lo que sus propiedades se pueden ver en la ventana *Propiedades*.



3. Cambie el nombre de este elemento a **`obtener saldo en cuenta`** y pulse **Entrar** para confirmar. Si el nombre del elemento no está seleccionado, haga doble clic para seleccionarlo.
Observe que el tamaño del caso de uso se ajusta automáticamente a la longitud del texto.



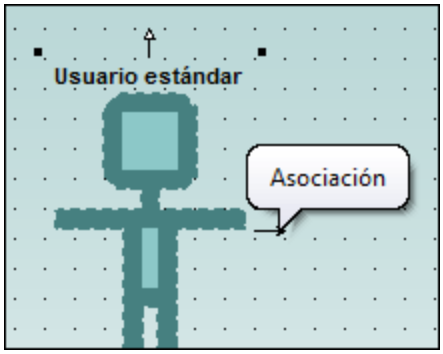
Nota: use **Ctrl+Entrar** para añadir un salto de línea en el nombre del caso de uso.

Manipular elementos de UModel: controladores y compartimentos

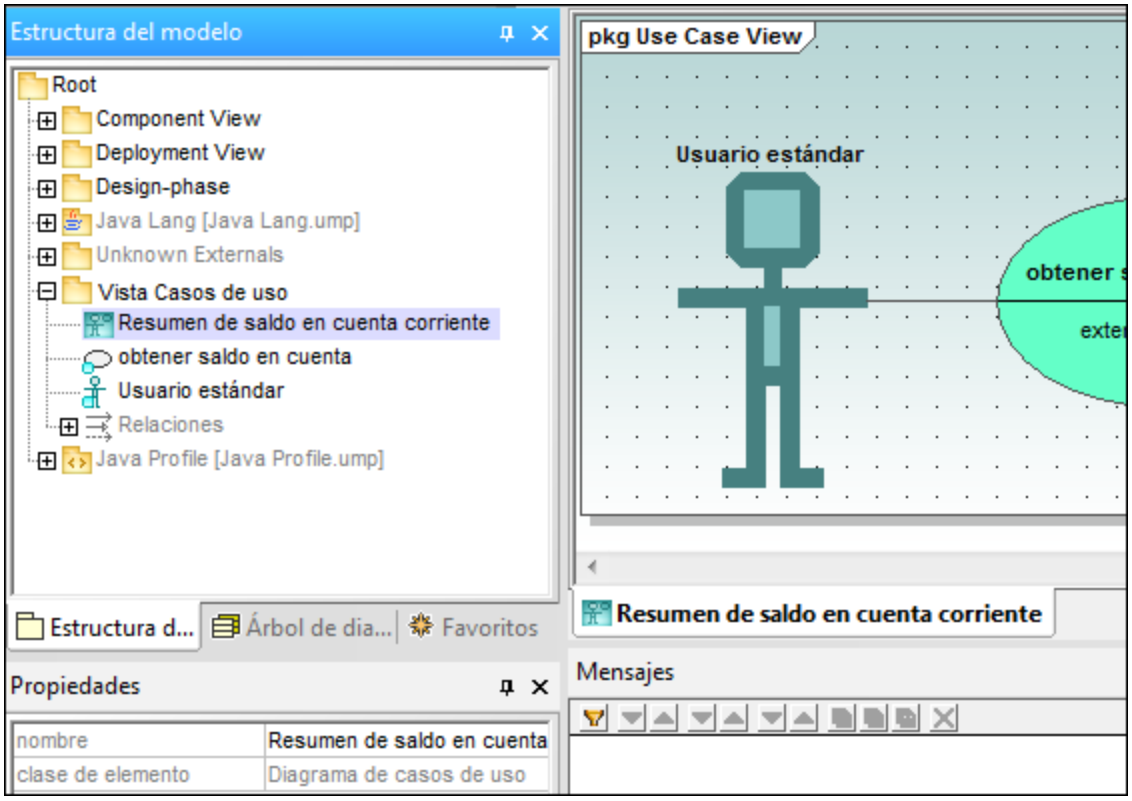
Cuando están seleccionados, los elementos de modelado del diagrama presentan varios controladores de conexión y otros elementos que sirven para manipularlos. Los controladores sirven para crear relaciones entre los elementos o para mostrar/ocultar ciertos compartimentos del elemento.

1. Haga doble clic en el texto **`Actor1`** del elemento actor, cambie el nombre a **`Usuario estándar`** y pulse **Entrar** para confirmar.

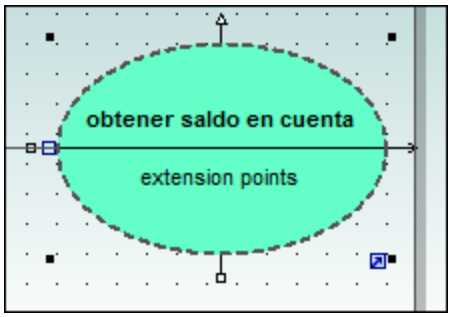
2. Pase el cursor por encima del controlador derecho del actor. Aparece una nota de información rápida que dice Asociación.



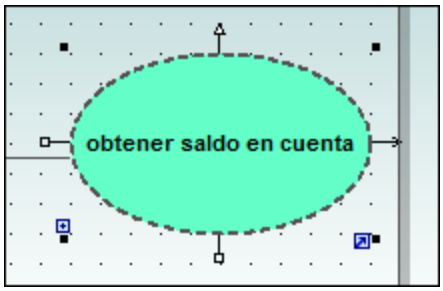
3. Haga clic en el controlador, arrastre la línea de asociación hacia la derecha y suéltela en el caso de uso **`obtener saldo en cuenta`**. Ahora se crea una asociación entre el actor y el caso de uso. Las propiedades de la asociación se pueden ver en la ventana *Propiedades*. La nueva asociación también se añade a la *Estructura del modelo*, bajo el elemento Relaciones del paquete Vista Casos de uso.

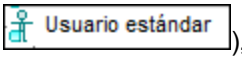


4. Haga clic en el caso de uso y arrástrelo hacia la derecha. Las propiedades de la asociación están visibles en el objeto asociación.
5. Haga clic en el caso de uso para seleccionarlo y después haga clic en el icono de contraer situado en el borde izquierdo del caso de uso.

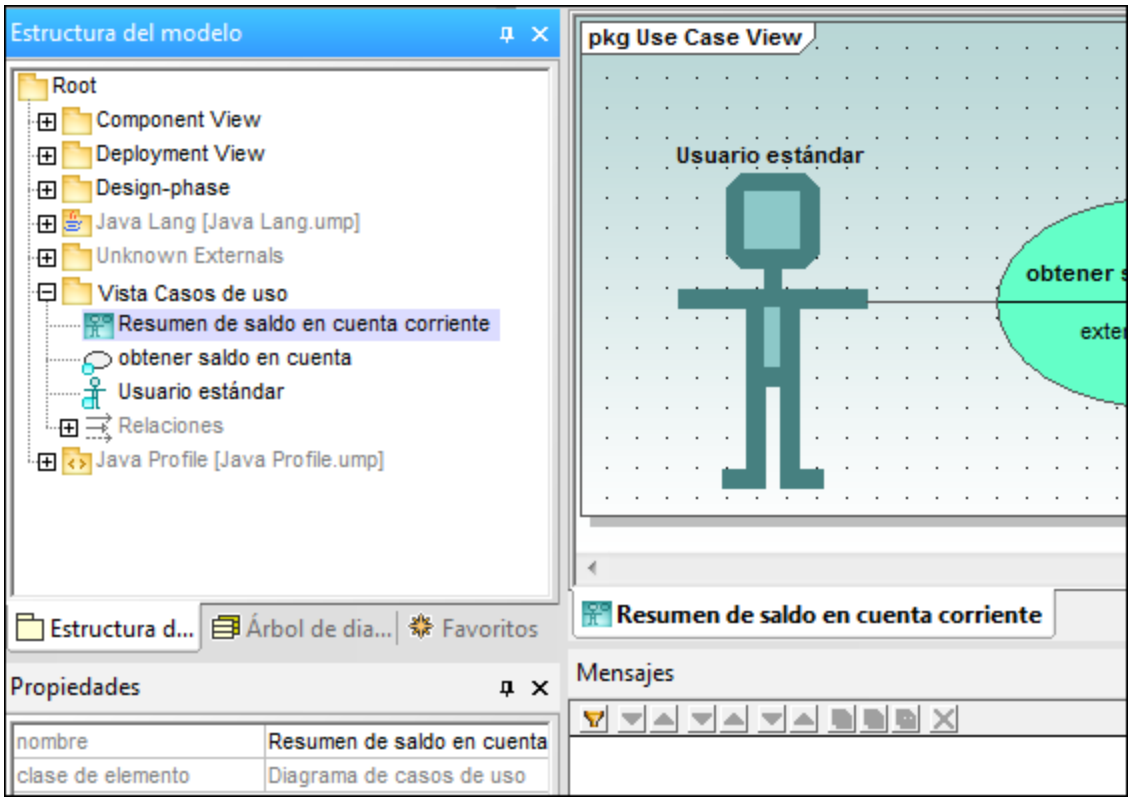


Ahora el compartimento "extension points" está oculto.



Si en la *Estructura del modelo* el icono del elemento incluye un punto azul (p. ej. 

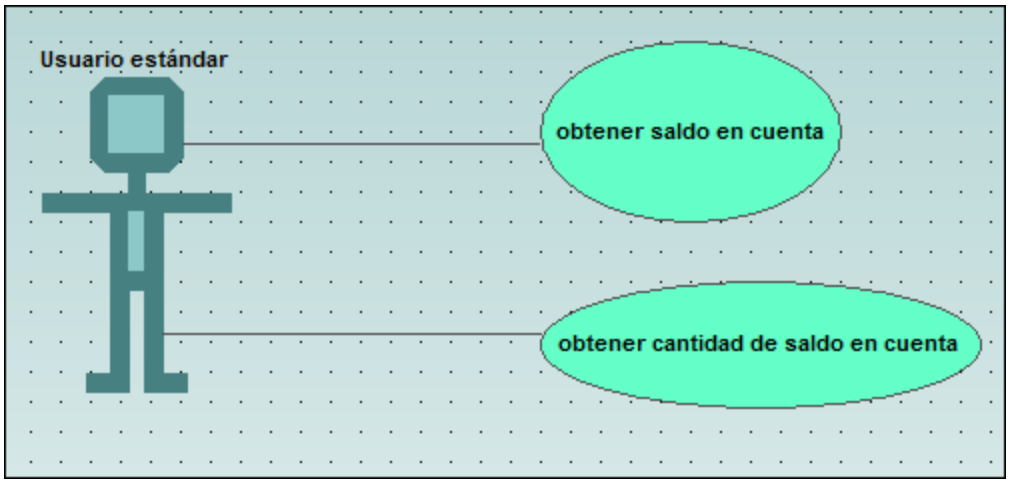
), esto significa que el elemento está visible en la ventana de diagramas actual. Por ejemplo, en la imagen siguiente puede ver que hay tres elementos visibles en el diagrama y por eso los tres elementos están marcados con un punto azul en la *Estructura del modelo*:



Al ajustar el tamaño del actor también se ajusta el campo de texto, que puede ser multilínea. Puede insertar un salto de línea usando **Ctrl+Entrar**.

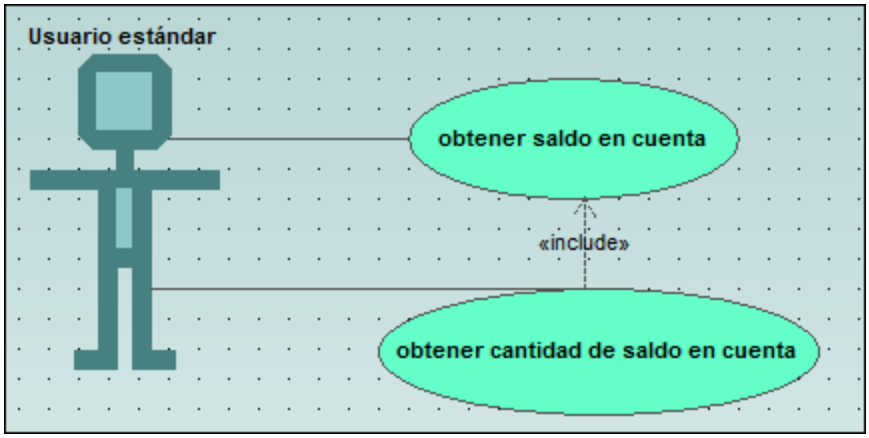
Terminar el diagrama de casos de uso:

1. Haga clic en el icono **Caso de uso**  de la barra de herramientas mientras pulsa la tecla **Ctrl**.
1. Haga clic en dos posiciones verticales distintas en la ventana del diagrama para añadir dos casos de uso más. Cuando termine puede soltar la tecla **Ctrl**.
2. Al primer caso de uso lo llamamos `obtener cantidad de saldo en cuenta` y al segundo `generar informe mensual de ingresos`.
3. Haga clic en el icono **contraer** de ambos casos de uso para ocultar el compartimento de los puntos de extensión.
4. Haga clic en el actor y cree una asociación entre `Usuario estándar` y `obtener saldo en cuenta`.



Para crear una dependencia de inclusión entre los casos de uso (creando un caso de uso subordinado):

- Haga clic en el controlador *Inclusión* del caso de uso **`obtener cantidad saldo en cuenta`**, situado en el borde inferior, y arrástrelo hasta el caso de uso **`obtener saldo en cuenta`**. Se crea la dependencia de inclusión y el estereotipo include aparece en la flecha de línea discontinua.

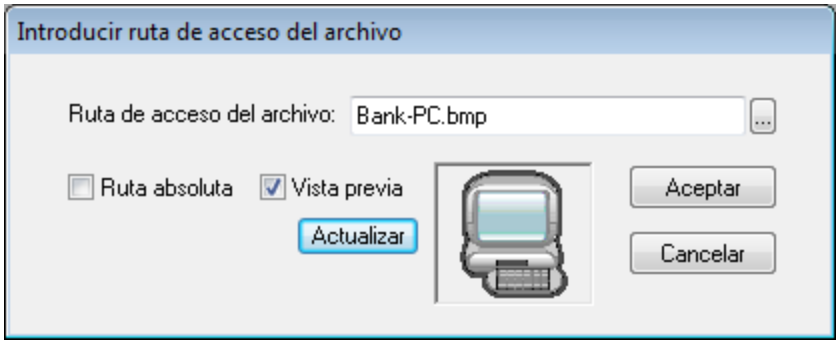


Insertar actores definidos por el usuario

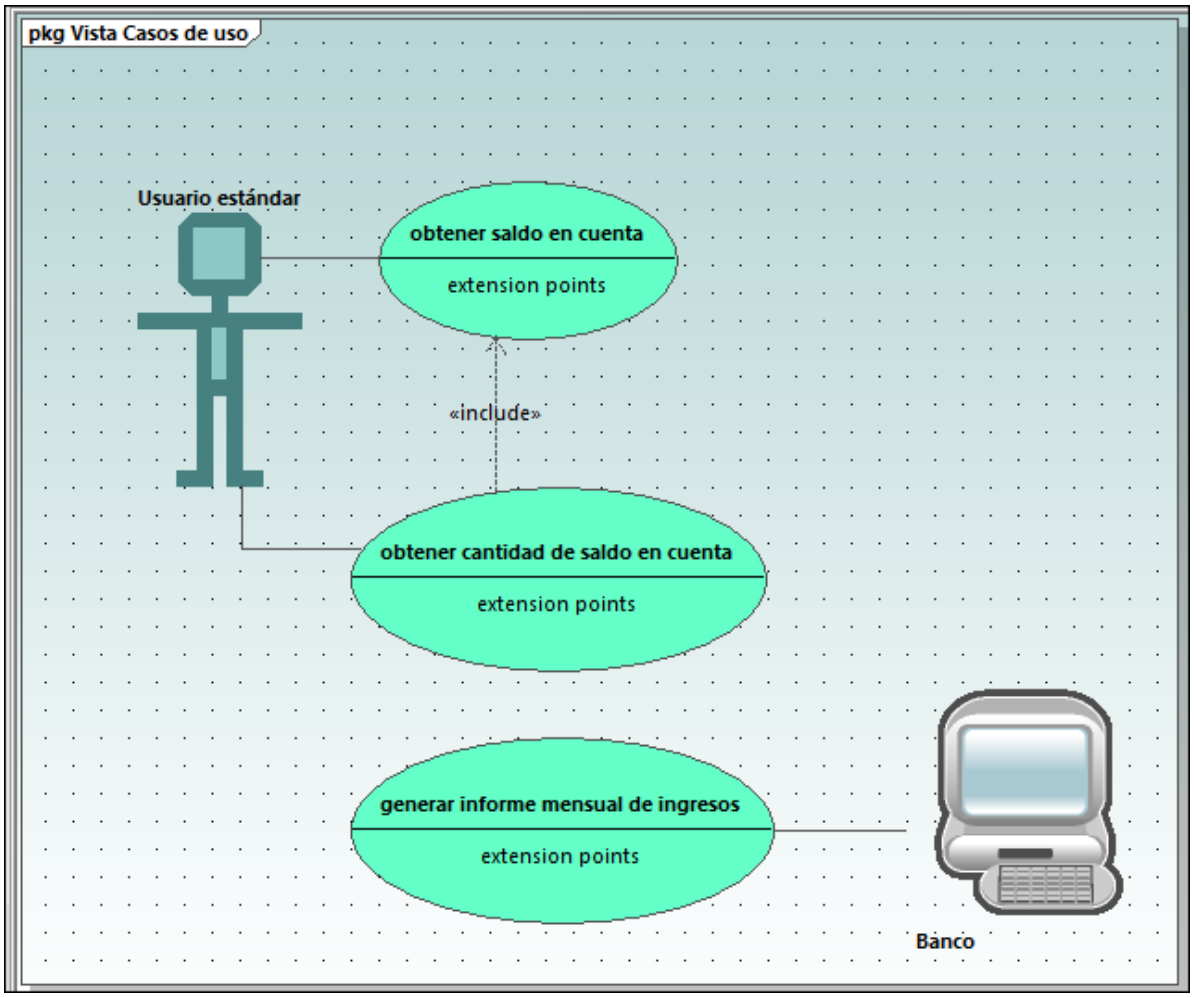
El actor del caso de uso **`generar informe mensual de ingresos`** no es una persona, sino un trabajo automatizado de procesamiento por lotes que ejecuta una computadora del banco. A continuación explicamos cómo añadir un actor nuevo al diagrama y cómo asignarle una imagen personal.

1. Haga clic en el botón **Actor** de la barra de herramientas para insertar un actor en el diagrama.
2. Cambie el nombre del actor por `Banco`.
3. En la ventana *Propiedades* haga clic en el botón **Examinar** situado junto a la entrada *nombre de archivo del icono* y busque el archivo **Bank-PC.bmp**, que se encuentra en la misma carpeta que el proyecto.

4. Desactive la casilla *Ruta absoluta* del cuadro de diálogo para convertir la ruta en relativa. Marque la casilla *Vista previa* del cuadro de diálogo para generar una vista previa del archivo seleccionado en el cuadro de diálogo.



5. Haga clic en **Aceptar** para confirmar la selección e insertar el nuevo actor. Mueva el nuevo actor `Banco` y colóquelo a la derecha del caso de uso más inferior.
6. Haga clic en el botón **Asociación** de la barra de herramientas y cree una conexión entre el actor `Banco` y el caso de uso **`generar informe mensual de ingresos`**. Esta es otra manera de crear asociaciones.



Nota: el color de fondo utilizado para que el mapa de bits sea transparente tiene los valores RGB 82.82.82.

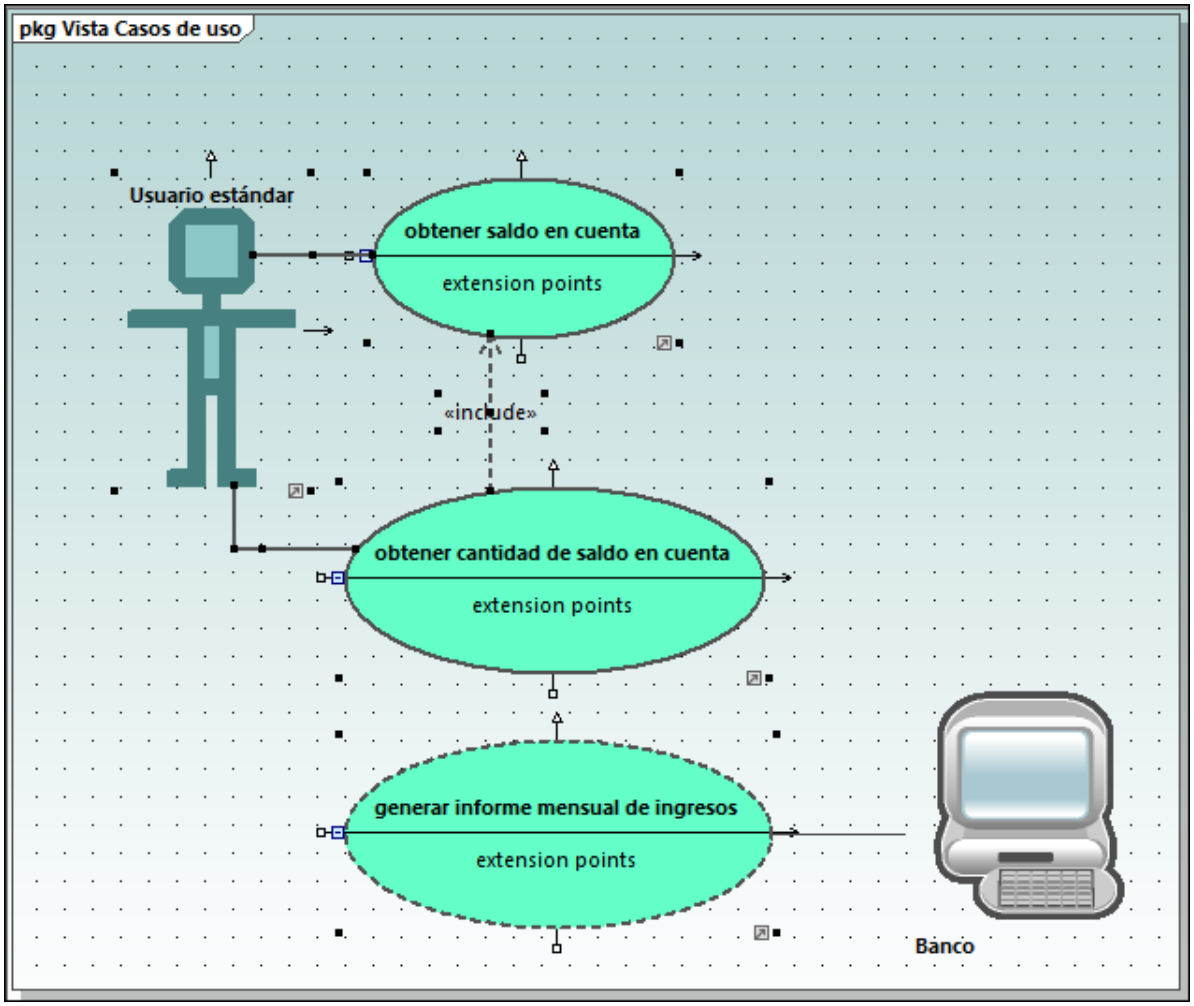
Alinear los elementos del diagrama y ajustar su tamaño

Cuando se arrastran los componentes del diagrama, aparecen líneas guía que facilitan la alineación de los elementos del diagrama. Esta opción se puede habilitar o deshabilitar:

1. En el menú **Herramientas** haga clic en **Opciones**.
2. En la pestaña *Vista* marque la casilla *Habilitar líneas de ajuste* del grupo de opciones *Alineación*.

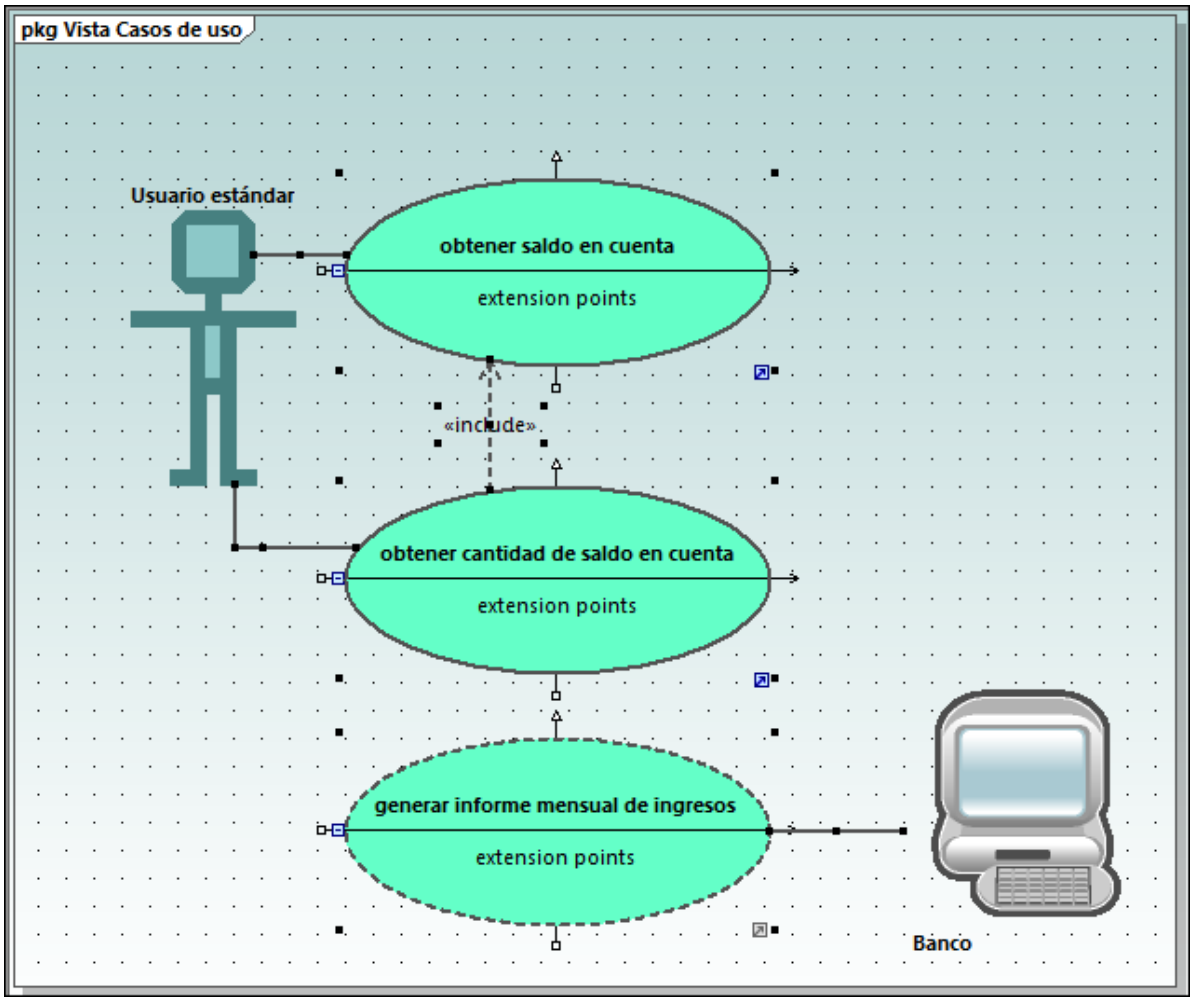
También puede alinear y ajustar el tamaño de varios elementos a la vez:

1. Arrastre el puntero por el diseño para crear un recuadro de selección que abarque los tres casos de uso, empezando por el superior. También puede seleccionar varios elementos haciendo clic en ellos mientras mantiene pulsada la tecla **Ctrl**. Observe que el último caso de uso que se marca aparece con un contorno de guiones en el diagrama y en la ventana *Vista general*.



Todos los casos de uso están seleccionados y el más inferior será el que se utilice como base para los ajustes que vamos a llevar a cabo a continuación.

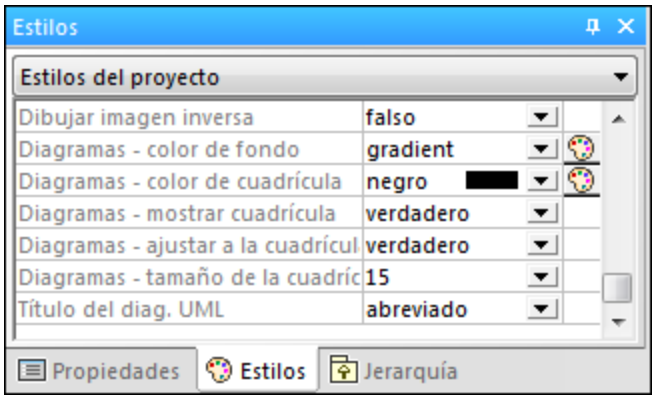
2. Haga clic en el botón **Igualar tamaño** de la barra de herramientas.
3. Para alinear todos los casos de uso haga clic en el botón **Centrar horizontalmente** de la barra de herramientas.



Cambiar el estilo de los diagramas en un proyecto

Todos los diagramas del proyecto de tutorial vienen con un fondo predeterminado que consiste en una cuadrícula de fondo sobre un degradado de colores. El aspecto de los diagramas del proyecto se puede configurar. Por ejemplo, se puede cambiar el color de fondo de todos los diagramas:

1. Abra la ventana *Estilos* (situada junto a la ventana *Propiedades*).
2. En el grupo *Estilos del proyecto* busque la opción `Diagramas - color de fondo`.



3. Cambie el valor **gradient** por el color que prefiera.

Para deshabilitar la cuadrícula de fondo del diagrama:

- Busque la opción `Diagramas - mostrar cuadrícula` y cambie el valor **verdadero** por el valor **falso**. También puede desactivar el botón **Mostrar cuadrícula** de la barra de herramientas.

2.3 Diagramas de clases

Este apartado del tutorial explica las siguientes tareas:

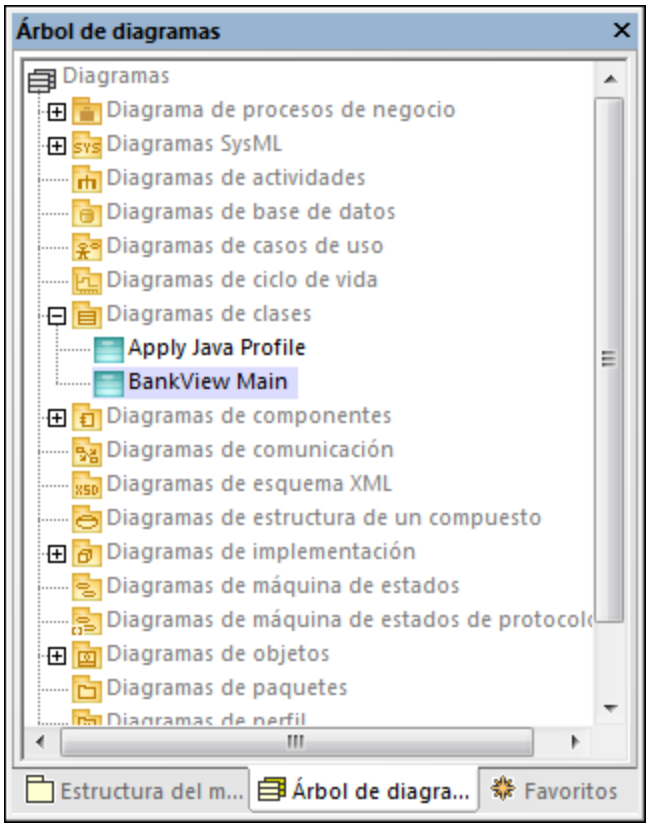
- cómo añadir una clase abstracta a un diagrama de clases,
- cómo añadir propiedades de clase y operaciones y definir parámetros, su dirección y su tipo,
- cómo añadir un tipo de retorno a una operación,
- cómo cambiar iconos por símbolos UML estándar,
- cómo eliminar y ocultar propiedades de clase y operaciones, y
- cómo crear una asociación compuesta entre dos clases.

Para continuar ejecute UModel y abra el proyecto **BankView-start.ump** (*véase* *Abrir el proyecto del tutorial*).

Para agregar una clase abstracta

El diagrama al que debemos añadir la clase abstracta se llama "BankView Main" y se puede abrir así:

1. En la ventana *Árbol de diagramas* expanda el paquete "Diagramas de clases" para ver todos los diagramas de clases que contiene el proyecto.



2. Ahora tiene dos opciones para abrir el diagrama:

- hacer doble clic en el icono del diagrama "BankView Main"
- o hacer clic con el botón derecho y seleccionar **Abrir el diagrama** en el menú contextual.

Nota: el diagrama también se puede abrir desde la ventana *Estructura del modelo*. Primero debe buscar el diagrama dentro del paquete "Root | Design-phase | BankView | com | altova | bankview" y después usar uno de los dos métodos anteriores para abrirlo.

En el diagrama de clases se pueden ver dos clases concretas unidas por una asociación compuesta.

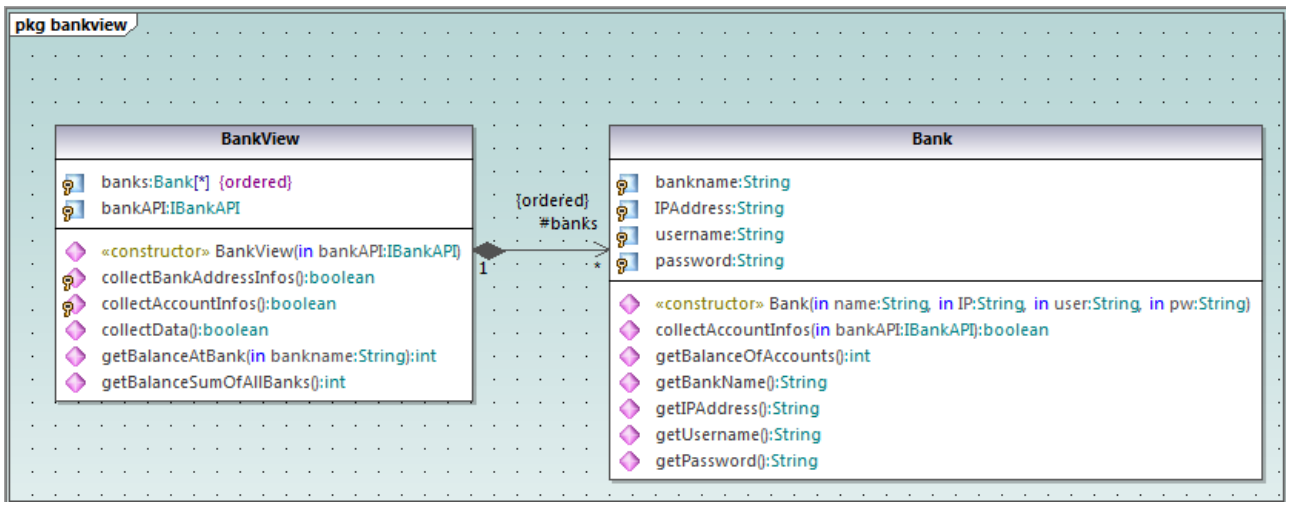
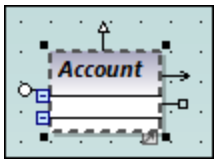


Diagrama "Bank View Main"

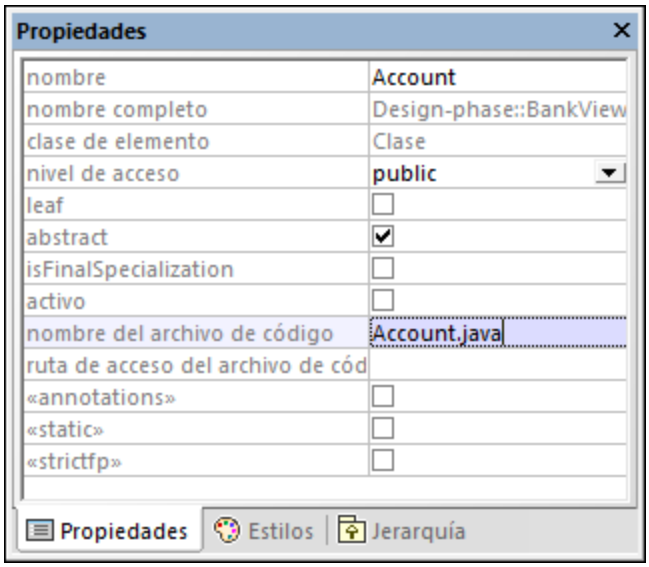
Para añadir la nueva clase abstracta:

1. Haga clic en el botón **Clase**  de la barra de herramientas y después haga clic a la derecha de la clase `Bank` para insertar la clase nueva.
2. Haga doble clic en el nombre de la clase nueva y cámbielo por `Account`.

3. En la ventana *Propiedades* marque la casilla *abstract* para que la clase sea abstracta. El título de la clase aparecerá en cursiva, lo cual la identifica como clase abstracta.

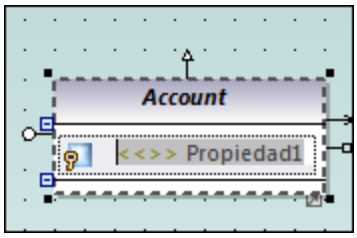


4. En la casilla *nombre del archivo de código* introduzca "Account.java" para definir la clase Java.

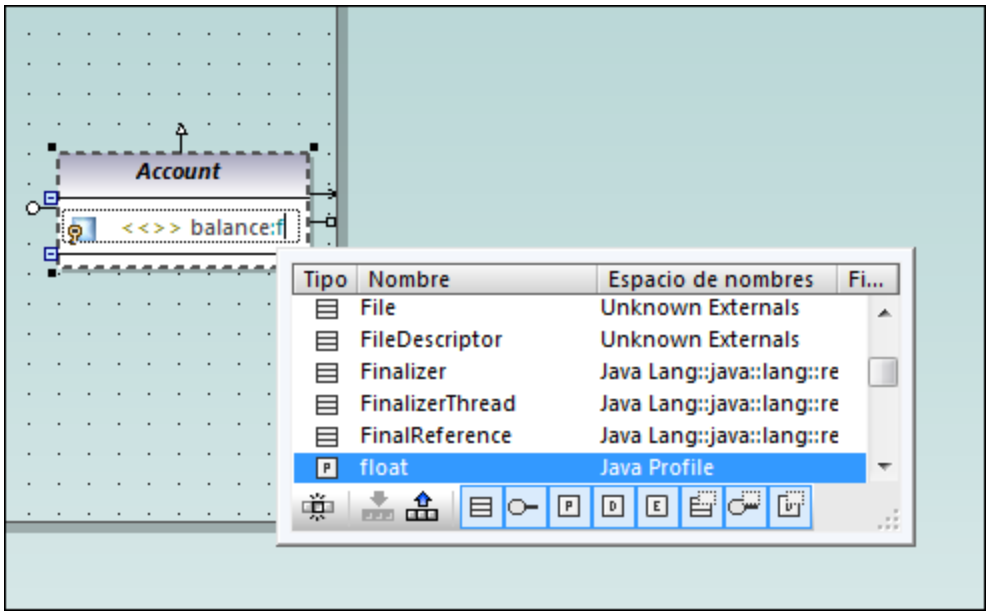


Agregar propiedades a una clase

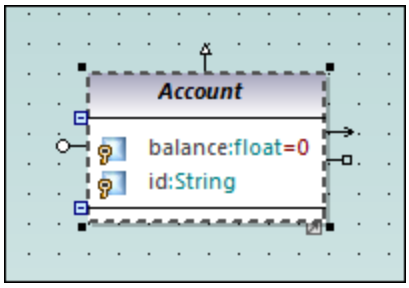
1. Haga clic con el botón derecho en la clase "Account" y seleccione **Nuevo/a | Propiedad** o pulse **F7**. Esto inserta una propiedad predeterminada llamada `Property1` con los identificadores de estereotipo `<< >>`.



2. Cambie el nombre de la propiedad por `balance` y después introduzca dos puntos (:). Aparece una lista desplegable con todos los tipos válidos.
3. Teclee la letra "f" y pulse **Entrar** para insertar el tipo devuelto `float`. Recuerde que las listas desplegables tienen en cuenta el uso de mayúsculas y minúsculas.

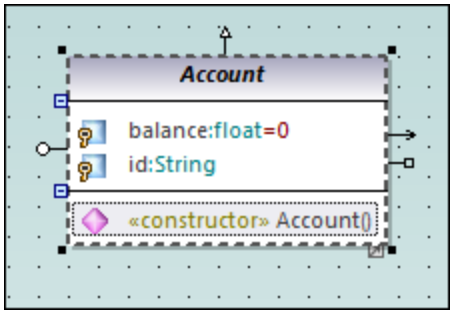


4. Siga en la misma línea y anexe "=0" para definir el valor predeterminado.
5. Para terminar (y usando el mismo método), cree una propiedad nueva llamada `id` de tipo `String`.

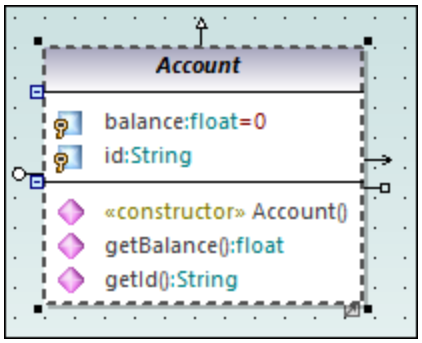


Agregar operaciones a una clase

1. Haga clic con el botón derecho en la clase `Account` y seleccione **Nuevo/a | Operación** o pulse **F8**.
2. Introduzca el nombre de operación "Account()". Observe que el estereotipo se cambió por `<<constructor>>` porque el nombre de la operación es igual que el de la clase.

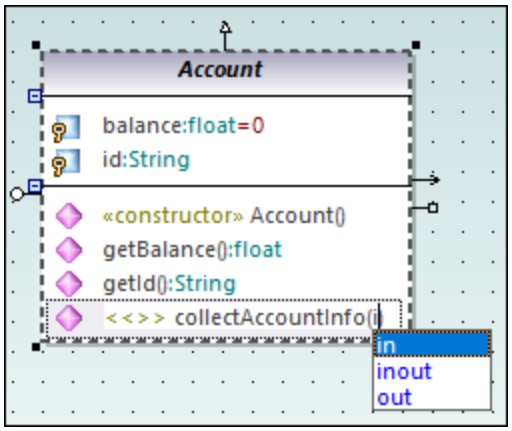


3. Para terminar (y usando el mismo método), añada dos operaciones más llamadas `getBalance():float` y `getId():String`.

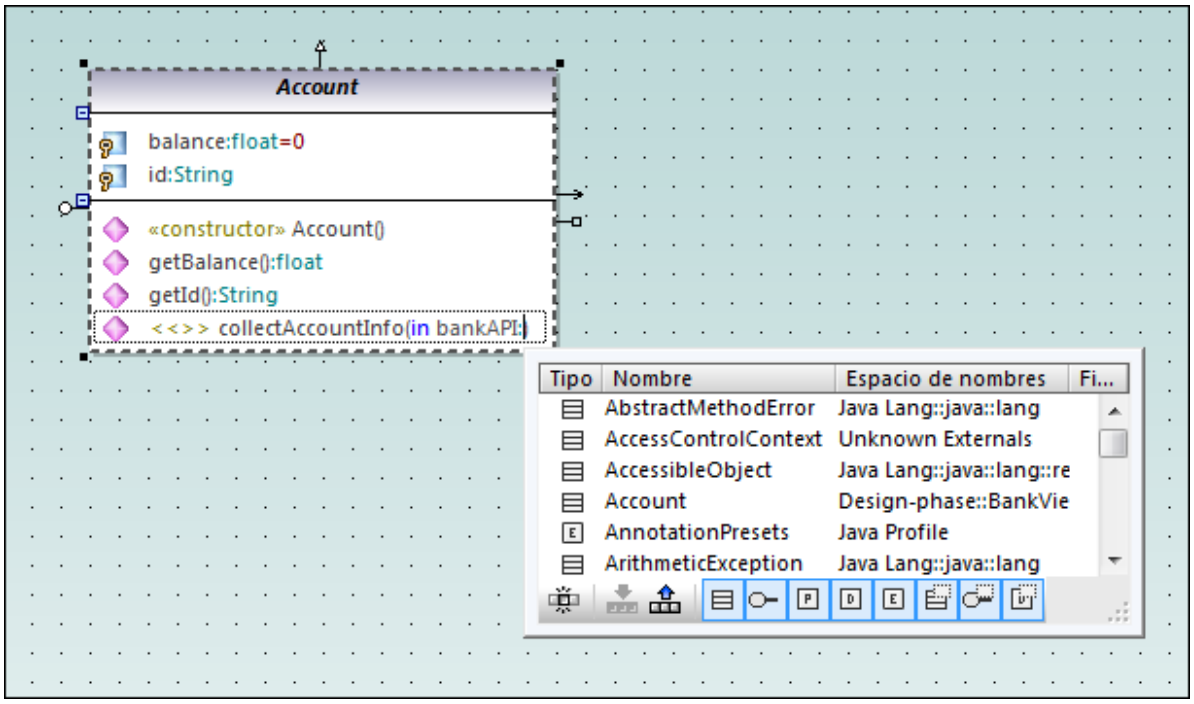


Ahora añadiremos una operación nueva que toma un parámetro. También especificaremos la dirección y el tipo de este parámetro.

1. Pulse **F8** para crear otra operación que llamaremos `collectAccountInfo()`.
2. Ponga el cursor del ratón entre los paréntesis y teclee la letra "i". Aparece una lista desplegable donde puede seleccionar la dirección del parámetro: `in`, `inout` o `out`.



3. Seleccione `in` en la lista desplegable, introduzca un espacio y siga editando el parámetro en la misma línea.
4. Introduzca el nombre de parámetro "bankAPI" y después dos puntos (:). Aparece una lista desplegable donde puede seleccionar el tipo del parámetro.

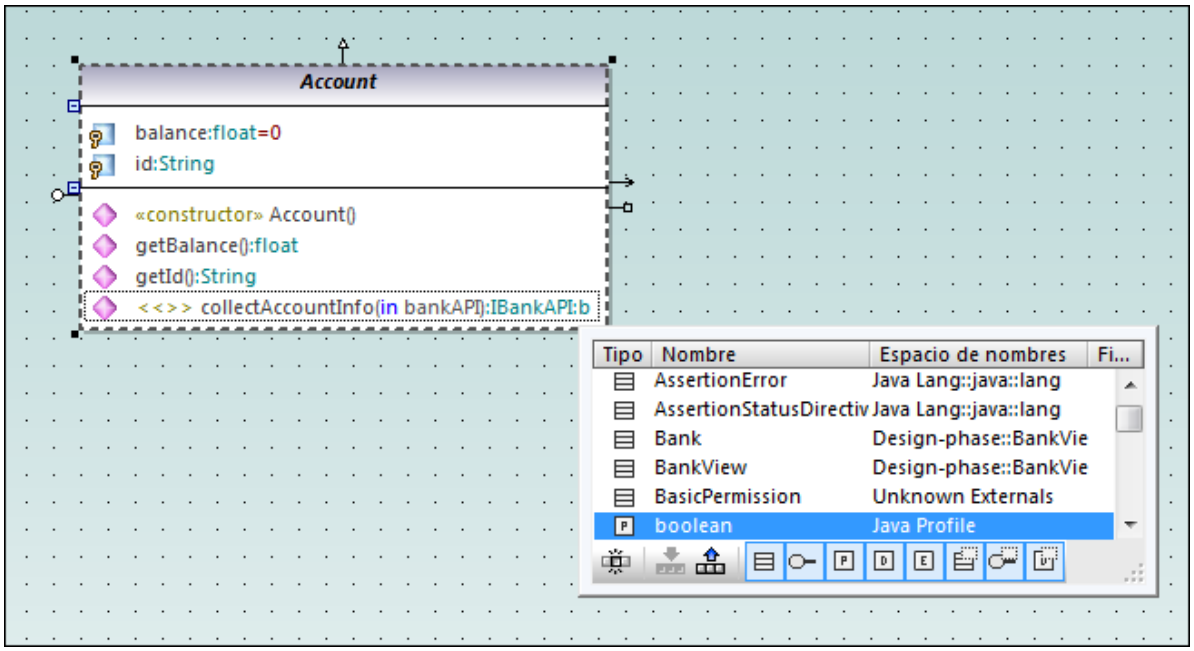


5. Seleccione **IBankAPI** en la lista desplegable.

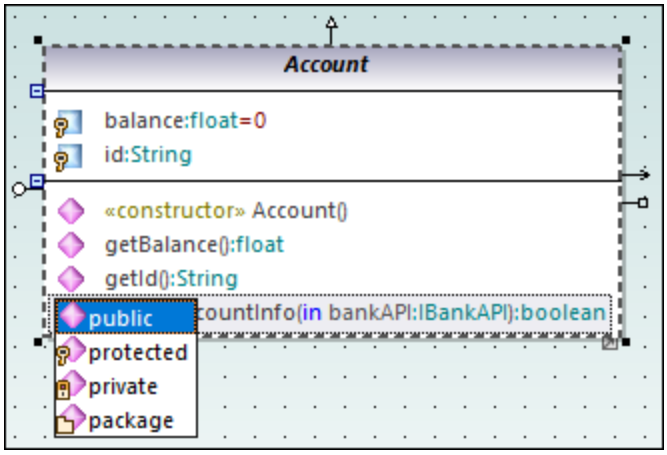
Agregar un tipo devuelto a una operación

Llegados a este punto se añadió el parámetro de la operación pero todavía no tiene un tipo devuelto.

1. Ponga el cursor del ratón detrás del paréntesis de cierre ")" e introduzca dos puntos (:). Aparece una lista desplegable donde puede seleccionar un tipo devuelto.
2. Teclee la letra "b" y seleccione el tipo de datos `boolean`.



Para especificar el nivel de acceso de una operación (privado, protegido o público) haga clic en el icono que precede al nombre de la operación y seleccione el valor correspondiente. Por ejemplo:



El "paquete" de visibilidad se puede aplicar a Java. En C# debe usar el elemento "paquete" para indicar que el nivel de acceso es interno. Para más información sobre cómo se asignan los elementos de UModel a las construcciones de cada lenguaje consulte UModel Element Mappings.

Cambiar los iconos por símbolos UML estándar

Los iconos de nivel de acceso se pueden cambiar por símbolos UML estándar:

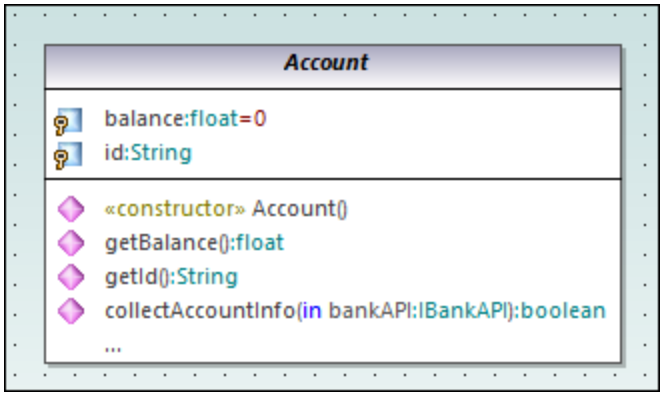
1. En la ventana *Estilos* seleccione **Estilos del proyecto** en la lista desplegable.
2. Desplácese hasta la opción *Mostrar nivel de acceso* y seleccione **Estilo de UML**.

Eliminar y ocultar propiedades y operaciones de clase en un diagrama de clases

Pulse **F8** para agregar una operación de ejemplo llamada `Operación1` a la clase `Account`.

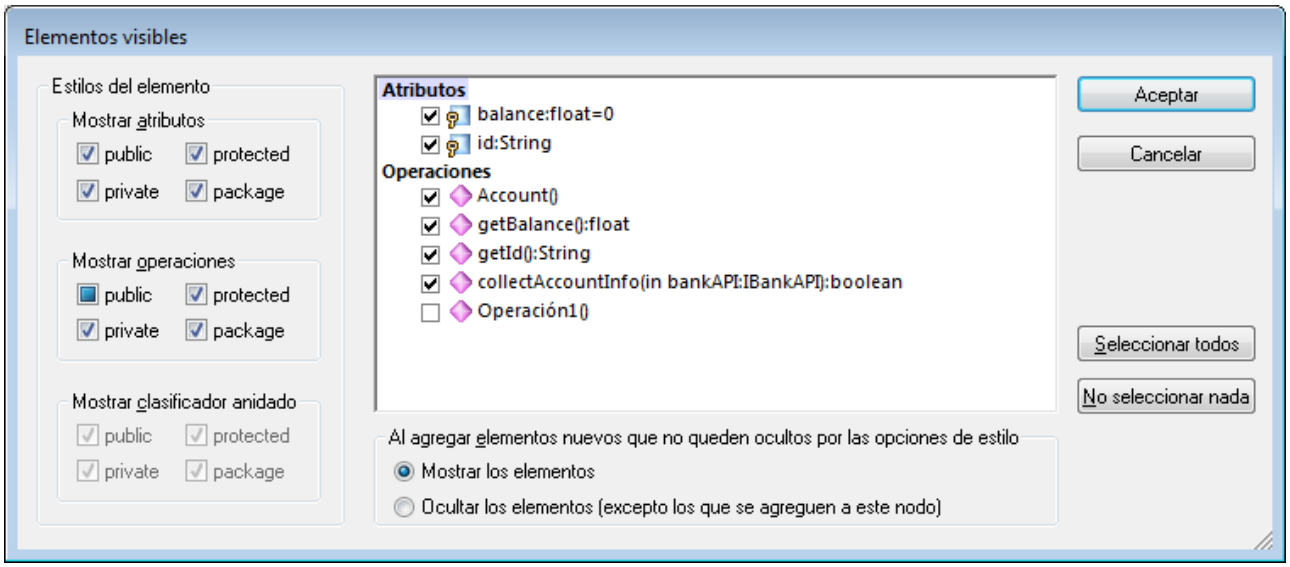
Para eliminar esta operación de ejemplo basta con seleccionarla y pulsar **Supr**. También puede hacer clic con el botón derecho en la operación y seleccionar **Eliminar** en el menú contextual. Aparecerá un cuadro de mensaje preguntando si desea eliminar el elemento del proyecto. Haga clic en **Sí** para eliminar `Operación1` del diagrama de clases y del proyecto.

Para eliminar una operación de una clase pero no del proyecto pulse **Ctrl+Supr**. Así, la operación no aparece en el diagrama pero sigue existiendo en el proyecto. Las clases que tienen miembros ocultos incluyen tres puntos suspensivos al final del recuadro. Por ejemplo:



Una clase con operaciones ocultas

Para dejar de ocultar la operación haga doble clic en los puntos suspensivos que aparecen al final del recuadro de la clase. Esto abre un cuadro de diálogo donde puede elegir qué elementos pueden verse en el diagrama:



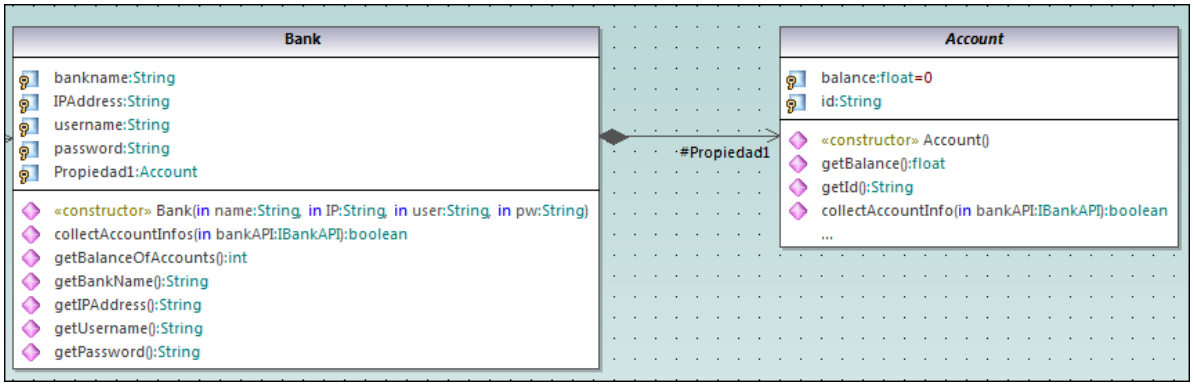
Cuadro de diálogo "Elementos visibles"

Si lo prefiere, puede configurar UModel para que no muestre un cuadro de mensaje cada vez que se intenta eliminar un objeto del diagrama:

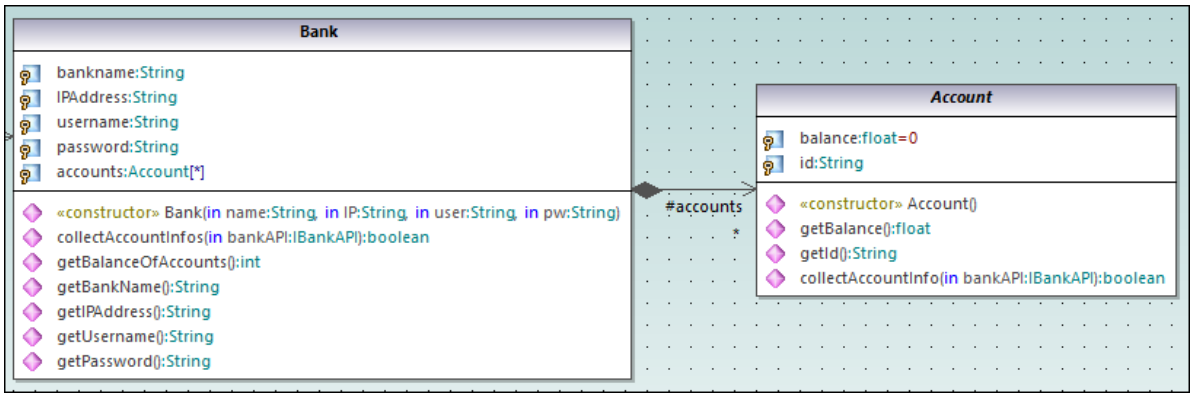
1. En el menú **Herramientas** haga clic en el comando **Opciones**.
2. Haga clic en la pestaña *Edición*.
3. En el grupo de opciones *Al eliminar del proyecto preguntar siempre*, desactive la casilla *desde los diagramas*.

Crear una asociación de composición entre las clases Bank y Account

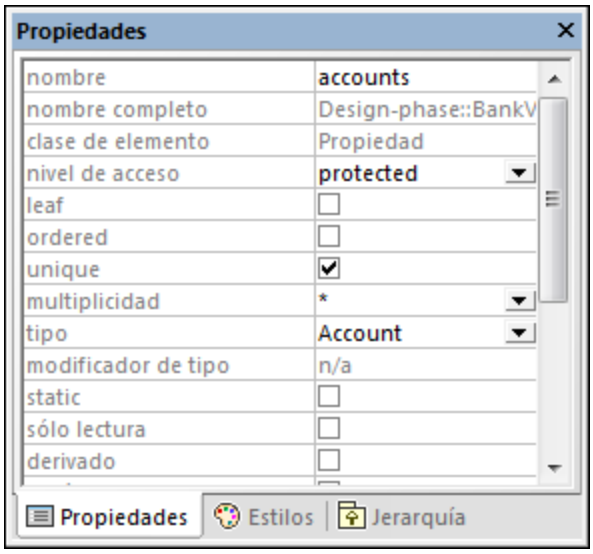
1. Haga clic en el botón **Composición** de la barra de herramientas y después arrastre el puntero desde la clase `Bank` hasta la clase `Account`. La clase se resalta cuando la asociación se puede crear. Como resultado se crea una propiedad nueva llamada `Propiedad1:Account` en la clase `Bank` y una flecha de asociación compuesta une las dos clases.



2. Haga doble clic en la nueva propiedad `Propiedad1` de la clase `Bank` y llámela `accounts` (con cuidado de no eliminar la definición de tipo `Account` que aparece en color esmeralda).
3. Pulse la tecla **Fin** para colocar el cursor de texto al final de la línea.
4. Introduzca un corchete de apertura ([ ) y seleccione el asterisco (*) en la lista desplegable. Esto define la multiplicidad, es decir, el hecho de que un banco puede tener muchas cuentas.



Recuerde que el rango de multiplicidad añadido previamente al diagrama también se puede ver en la ventana *Propiedades*:



2.3.1 Crear clases derivadas

Este apartado del tutorial explica:

- cómo añadir un diagrama de clases nuevo al proyecto,
- cómo añadir clases que ya existen al diagrama,
- cómo añadir una clase nueva al diagrama y
- cómo crear clases derivadas desde una clase abstracta por medio de generalizaciones.

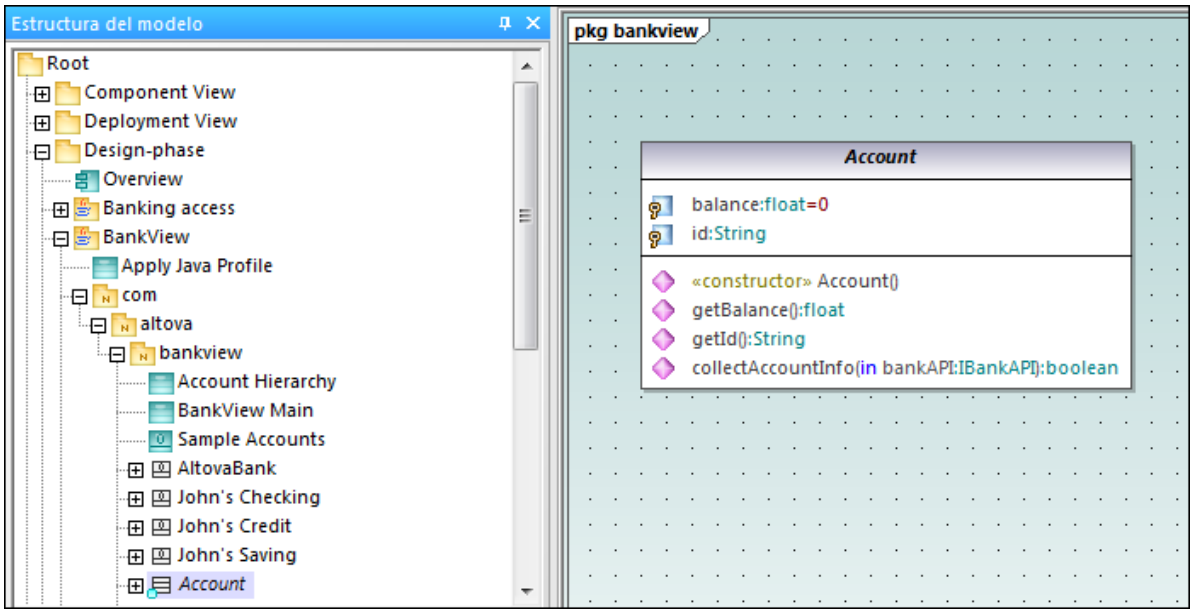
Nota: el requisito para este apartado del tutorial es haber completado el apartado anterior (*Diagramas de clases)* para crear la clase abstracta `Account`.

Crear un diagrama de clases nuevo

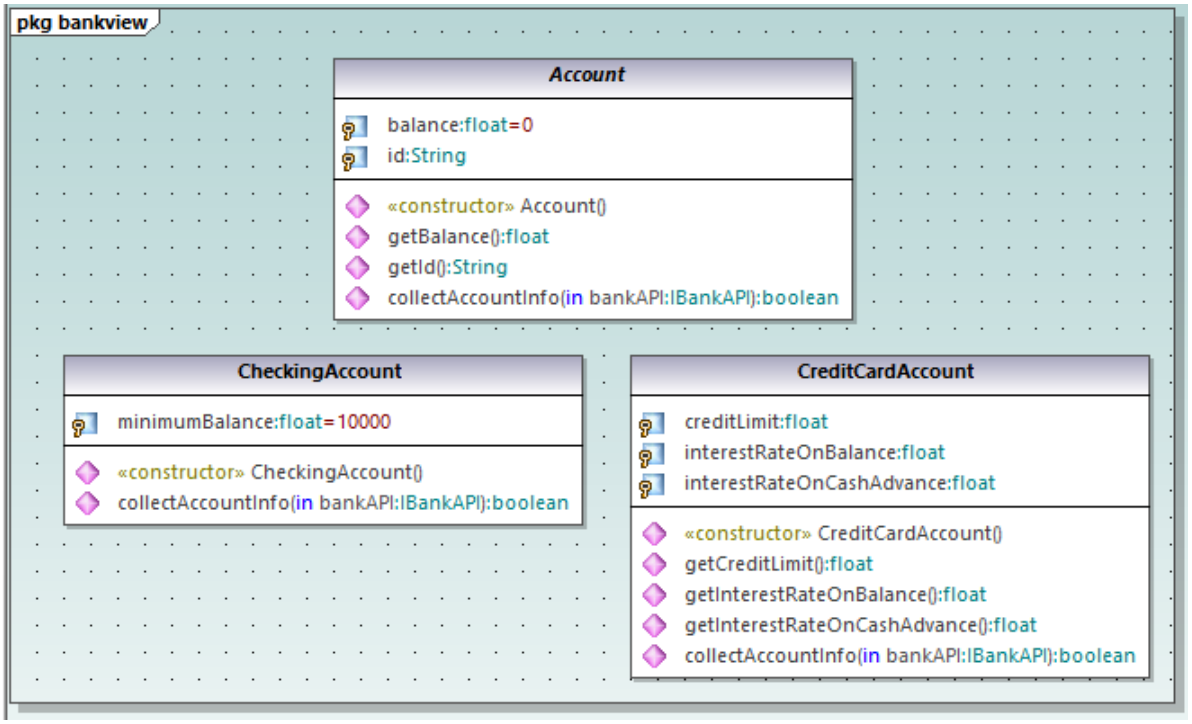
1. En la ventana *Estructura del modelo* haga clic con el botón derecho en el paquete `bankview` (situado dentro de **Root | Design-phase | BankView | com | altova**) y seleccione **Diagrama nuevo | Diagrama de clases**.
2. Haga doble clic en la nueva entrada `DiagramaDeClases1`, cambie el nombre por "Account Hierarchy" y pulse **Entrar** para confirmar. El nuevo diagrama "Account Hierarchy" puede verse en el área de trabajo.

Agregar clases que ya existen a un diagrama

1. En la ventana *Estructura del modelo* haga clic en la clase `Account` del paquete `bankview` (situado dentro de **com | altova | bankview**) y arrástrelo hasta el diagrama.



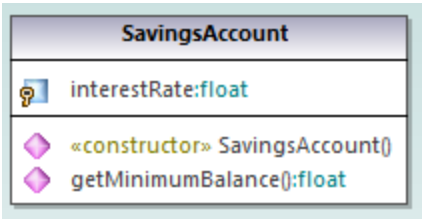
2. Haga clic en la clase `CheckingAccount` (del mismo paquete) y arrástrela hasta el diagrama. Ponga la clase debajo y a la izquierda de la clase `Account`.
3. Use el mismo método para insertar la clase `CreditCardAccount`. Póngala a la derecha de la clase `CheckingAccount`.



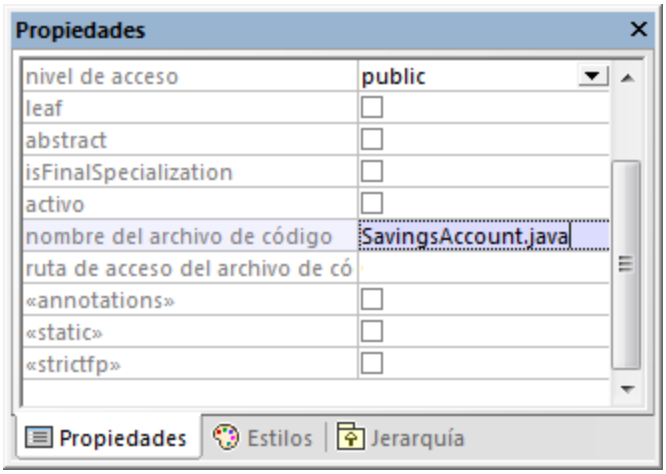
Agregar una clase nueva

La tercera clase derivada (`SavingsAccount`) se añadirá manualmente al diagrama.

1. Haga clic con el botón derecho en el diagrama y seleccione **Nuevo/a | Clase**. Esto añade automáticamente una clase nueva al paquete (`bankview`) que contiene el diagrama de clases actual "Account Hierarchy".
2. Haga doble clic en el nombre de la clase y cámbielo por `SavingsAccount`.
3. Cree la estructura de clase que aparece en la siguiente imagen. Para añadir propiedades y operaciones siga los métodos que se explican en el apartado anterior del tutorial (*Diagramas de clases*).



3. En la ventana *Propiedades* navegue hasta el cuadro de texto *nombre del archivo de código* e introduzca "SavingsAccount.java" para definir la clase Java.



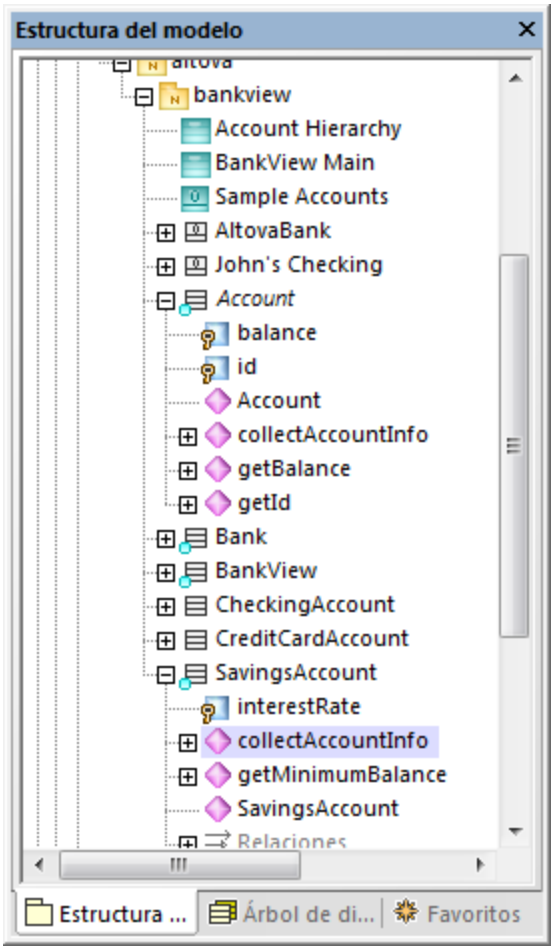
Estas son las propiedades y operaciones que se pueden copiar o mover de una clase a otra directamente:

- propiedades y operaciones de una clase del diagrama actual,
- propiedades y operaciones de clases distintas del mismo diagrama,
- propiedades y operaciones de la ventana *Estructura del modelo*,
- propiedades y operaciones de diagramas UML distintos (colocando los datos copiados en un diagrama diferente).

Esto se puede hacer mediante una operación de arrastrar o colocar o con una operación de corta y pega con **Ctrl + C** y **Ctrl + V** (*véase* *Renombrar, mover y copiar elementos*). Por ejemplo, puede copiar la operación `collectAccountInfo()` de la clase `Account` a la nueva clase `SavingsAccount` de la siguiente manera:

1. En la ventana *Estructura del modelo* expanda la clase `Account`.
2. Haga clic con el botón derecho en la operación `collectAccountInfo` y seleccione **Copiar**.
3. Haga clic con el botón derecho en la clase `SavingsAccount` y seleccione **Pegar**.

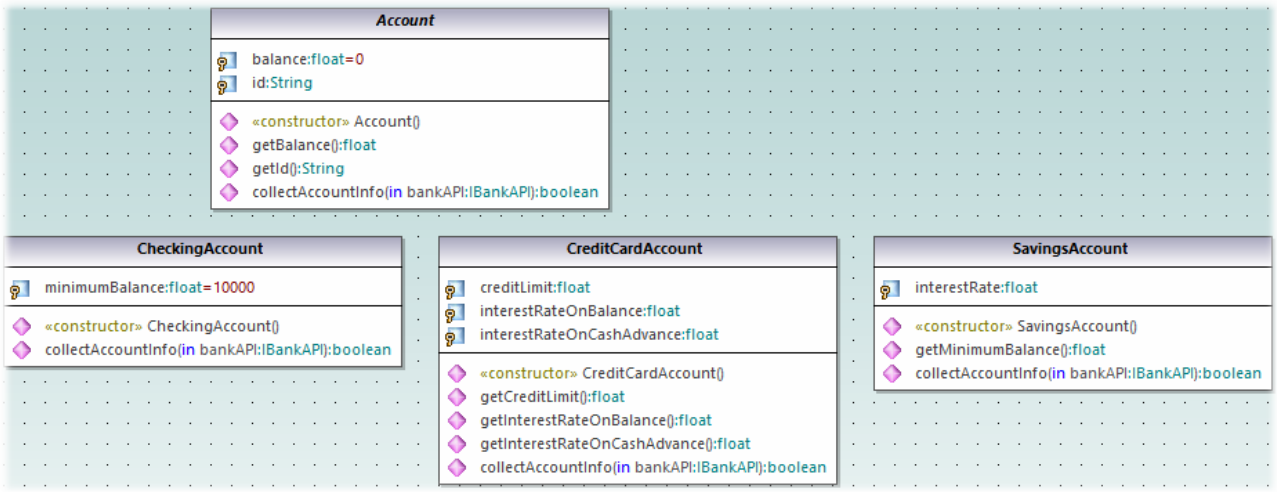
La operación se copia en la clase `SavingsAccount`, que automáticamente se expande para mostrar la nueva operación.



La nueva operación también se puede ver en la clase `SavingsAccount` del diagrama de clases.

Crear clases derivadas mediante generalización/especialización

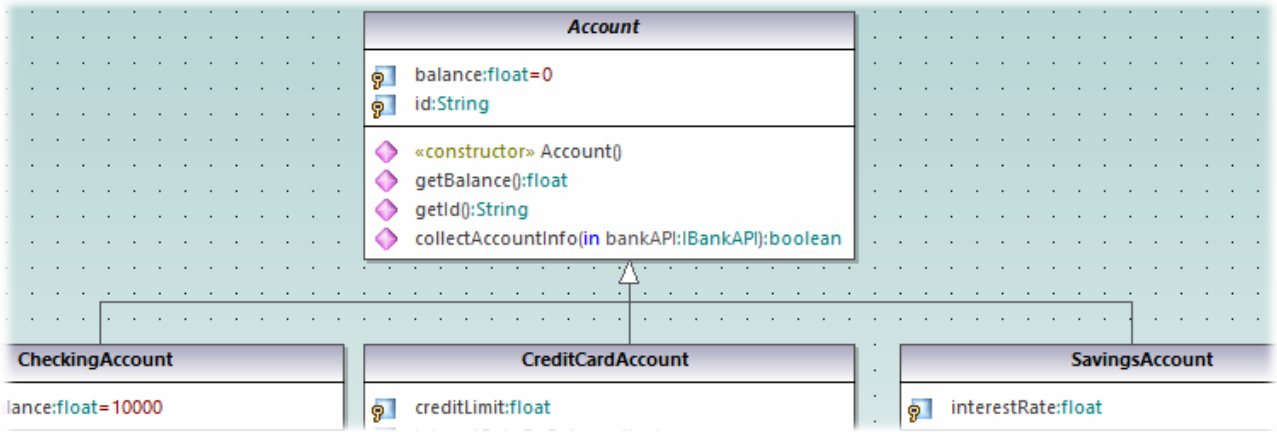
Llegados a este punto el diagrama de clases contiene la clase abstracta `Account` y tres clases específicas.



Ahora vamos a crear una relación de generalización/especialización entre `Account` y las clases específicas (es decir, crearemos tres clases derivadas concretas).

1. Haga clic en el botón **Generalización**  de la barra de herramientas mientras mantiene pulsada la tecla **Ctrl**.
2. Arrastre el puntero del ratón desde la clase `CreditCardAccount` hasta la clase `Account`.
3. Arrastre el puntero del ratón desde la clase `CheckingAccount` hasta la *punta de flecha* de la generalización que acaba de crear.
4. Arrastre el puntero del ratón desde la clase `SavingsAccount` hasta la *punta de flecha* de la generalización que creó en el paso anterior (ya puede soltar la tecla **Ctrl**).

Como resultado se crearon flechas de generalización entre las tres subclases y la superclase `Account`.



2.4 Diagramas de objetos

En este apartado del tutorial aprenderá a:

- crear un diagrama a partir de clases y de objetos,
- crear objetos/instancias y definir relaciones entre ellos,
- dar formato a asociaciones/vínculos y
- a introducir datos reales dentro de objetos/instancias.

Para continuar ejecute UModel y abra el proyecto **BankView-start.ump** (*véase* *Abrir el proyecto del tutorial*). Este proyecto incluye un diagrama de objetos predefinido llamado "Sample Accounts" que nos servirá de ejemplo para este apartado.

Crear un diagrama a partir de objetos y clases

En la ventana *Estructura del modelo* navegue hasta **Root | Design-phase | BankView | com | altova | bankview**. Después haga doble clic en el icono situado junto al diagrama "Sample Accounts".

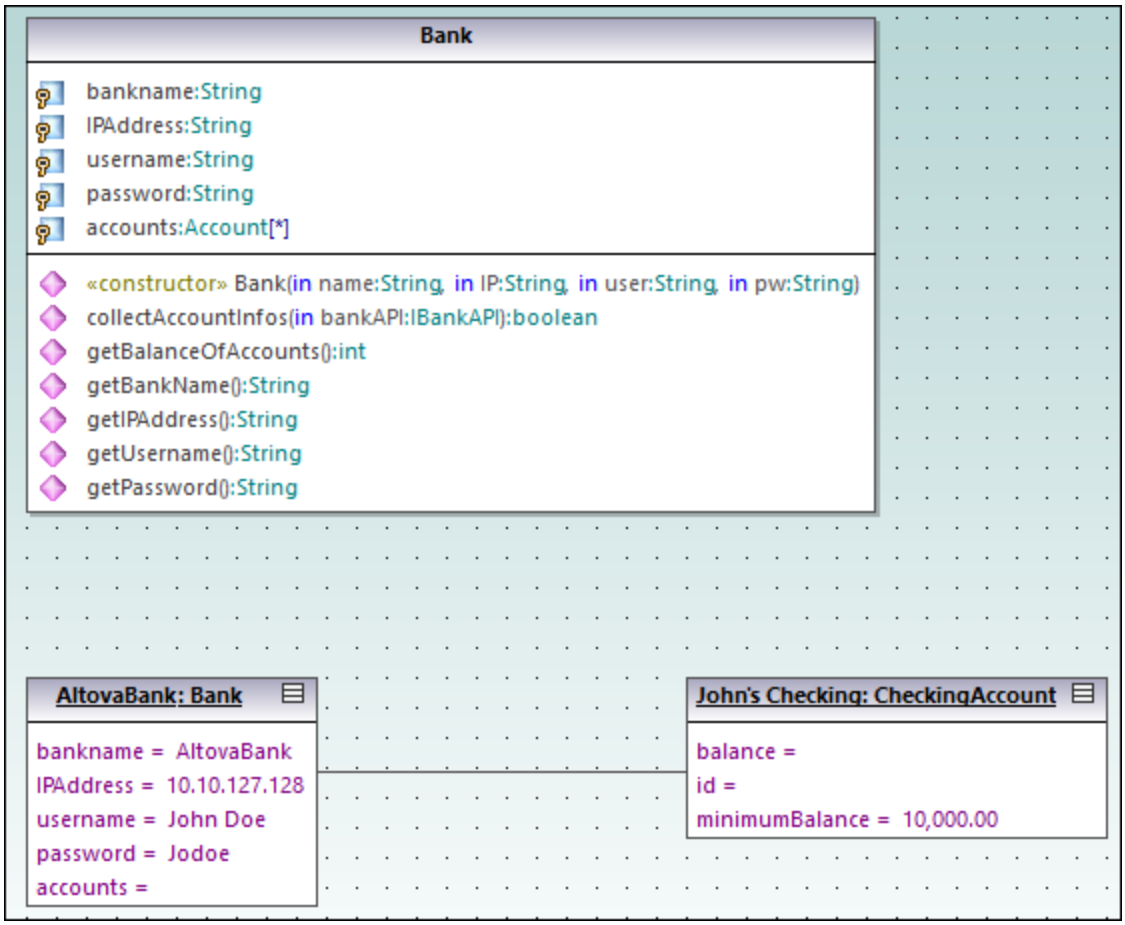
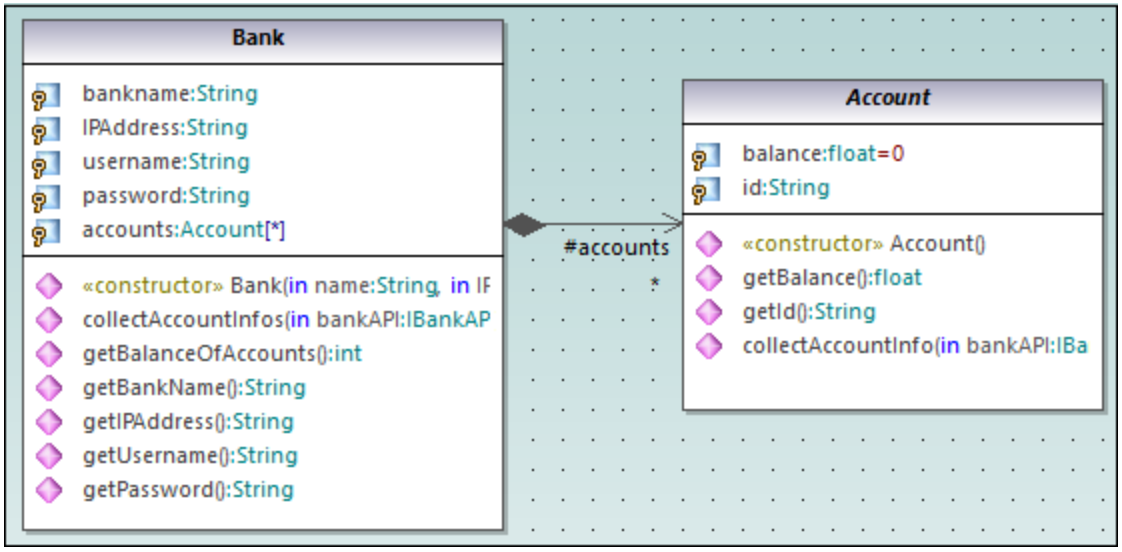


Diagrama "Sample Accounts"

Este diagrama de objetos está compuesto por clases e instancias de dichas clases (objetos). Concretamente, `AltovaBank:Bank` es el objeto/la instancia de la clase `Bank`, mientras que `John's checking: CheckingAccount` es una instancia de la clase `CheckingAccount` (que todavía no se añadió al diagrama).

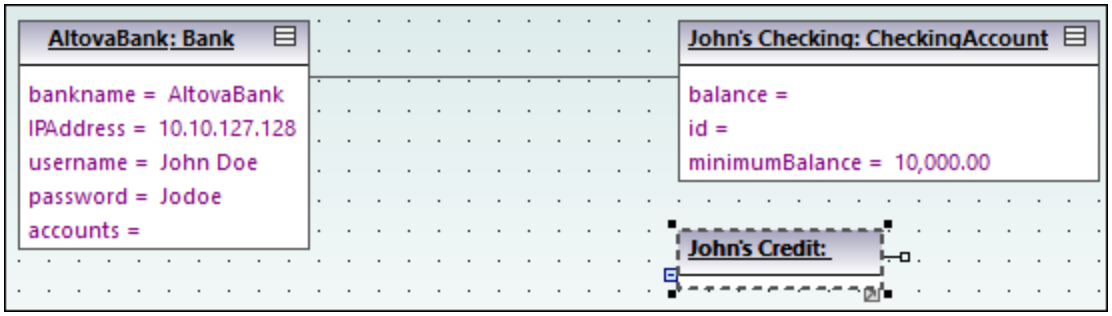
Ahora vamos a agregar la clase `Account` que falta. Para ello la seleccionamos y la arrastramos desde la *Estructura del modelo* hasta el diagrama. Observe que aparece automáticamente la asociación compuesta entre `Bank` y `Account` (esta asociación se definió en una de las fases anteriores del tutorial, en el apartado Diagramas de clases).



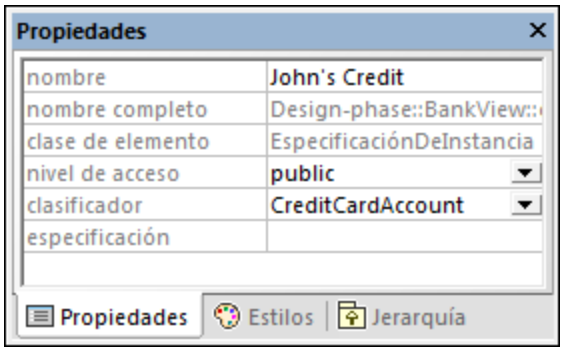
Agregar un objeto/una instancia nuevos (Método 1)

Ahora vamos a agregar un objeto nuevo llamado `John's Credit` al diagrama. Este objeto creará una instancia de la clase `CreditCardAccount`.

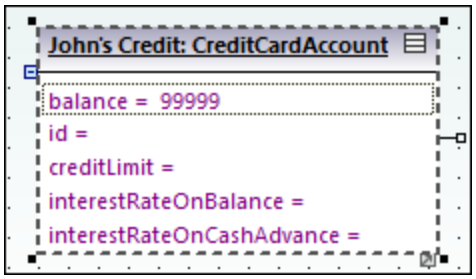
1. Haga clic en el botón **EspecificaciónDeInstancia** de la barra de herramientas y después haga clic dentro del diagrama, justo debajo del objeto `John's Checking: Checking Account`.
2. Cambie el nombre de la nueva instancia y llámela `John's Credit`. Para confirmar pulse **Entrar**.



3. Seleccione la instancia nueva para ver sus propiedades en la ventana *Propiedades*.
4. En la ventana *Propiedades*, junto al campo `clasificador`, seleccione **CreditCardAccount** en la lista desplegable.



El aspecto de la instancia cambia y ahora muestra todas las propiedades de la clase. Haga doble clic en cualquier propiedad para introducir un valor. Por ejemplo:

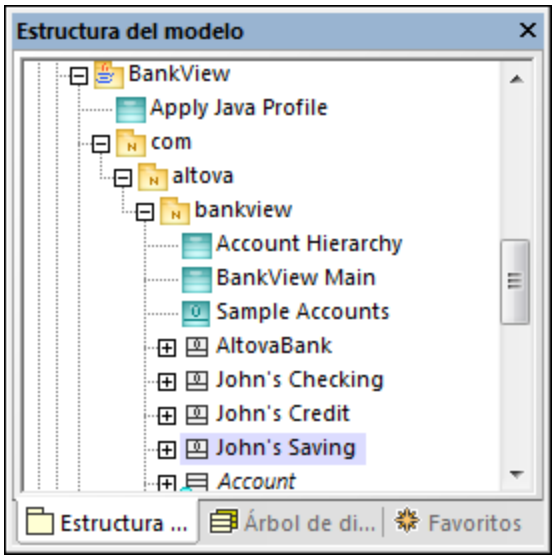


Para mostrar u ocultar nodos concretos, haga clic con el botón derecho en la instancia y seleccione **Mostrar u ocultar el contenido del nodo** (**Ctrl+Mayús+H)** en el menú contextual.

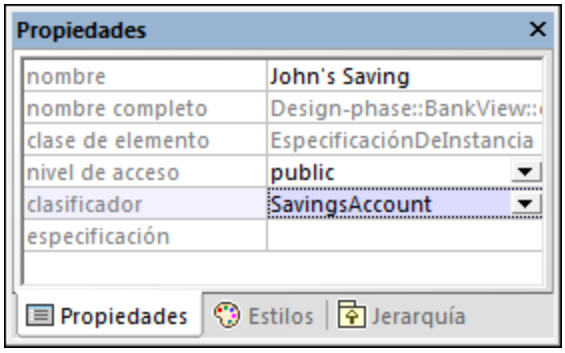
Agregar un objeto/una instancia nuevos (Método 2)

Ahora vamos a agregar una instancia nueva de la clase `SavingsAccount` pero utilizaremos un método distinto:

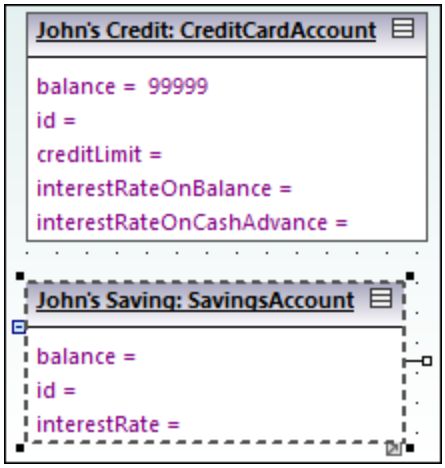
1. En la ventana *Estructura del modelo*, haga clic con el botón derecho en el paquete `bankview` y seleccione **Elemento nuevo | EspecificaciónDeInstancia**.
2. Cambie el nombre de la instancia nueva por `John's Saving` y pulse **Entrar** para confirmar. El objeto nuevo se añade al paquete y se coloca en la posición correcta.



3. Con el objeto todavía seleccionado en la ventana *Estructura del modelo*, seleccione ahora **SavingsAccount**, que se encuentra junto al campo `clasificador` de la ventana *Propiedades*.



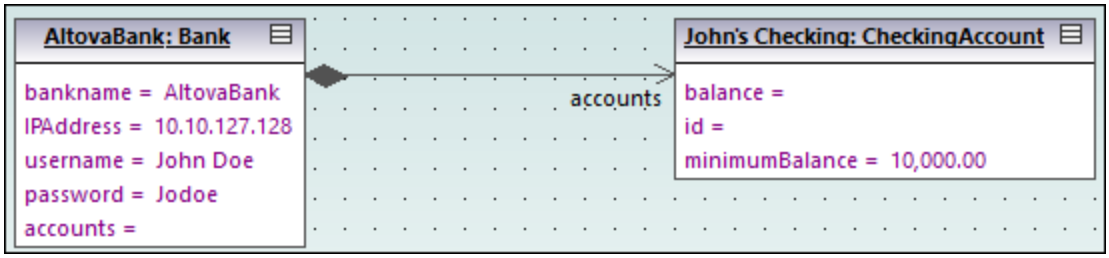
4. Arrastre el objeto `John's Saving` de la ventana *Estructura del modelo* hasta el diagrama y colóquelo justo debajo del objeto `John's Credit`.



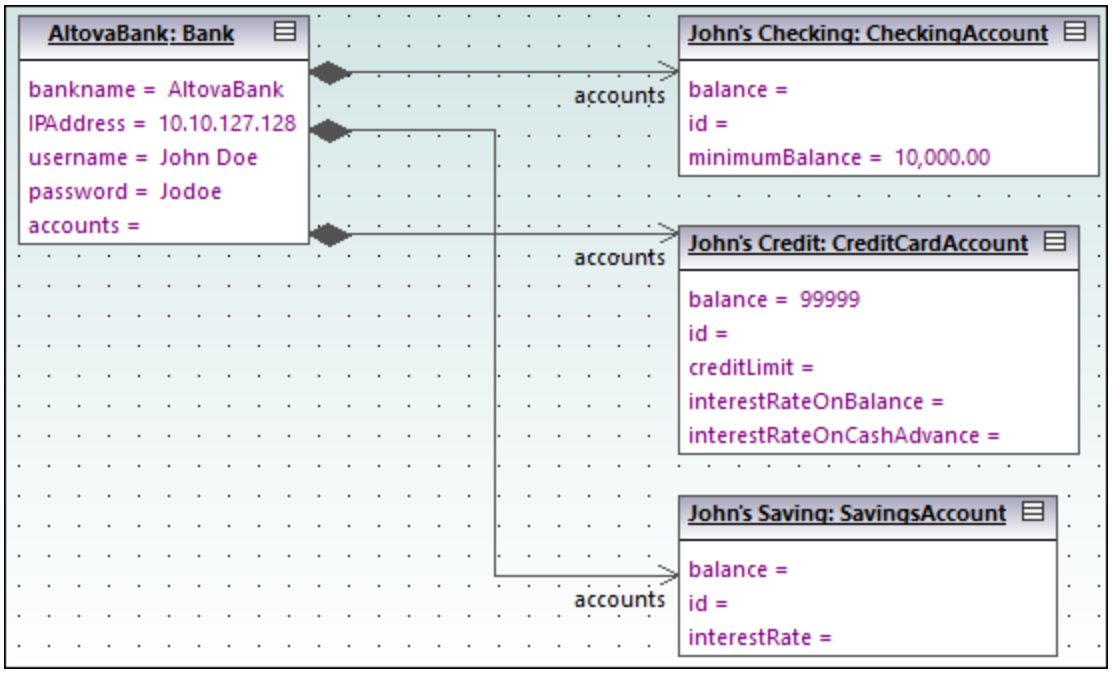
Crear vínculos entre objetos

Los vínculos son las instancias de asociaciones de clase y describen la relación existente entre objetos/instancias en un momento dado.

1. Haga clic en el vínculo existente (en la asociación) entre el objeto `AltovaBank: Bank` y el objeto `John's Checking: CheckingAccount`.
2. En la ventana *Propiedades*, junto al campo `clasificador`, seleccione la entrada **Account - Bank**. El vínculo se convierte en una asociación compuesta, de acuerdo con las definiciones de la clase.



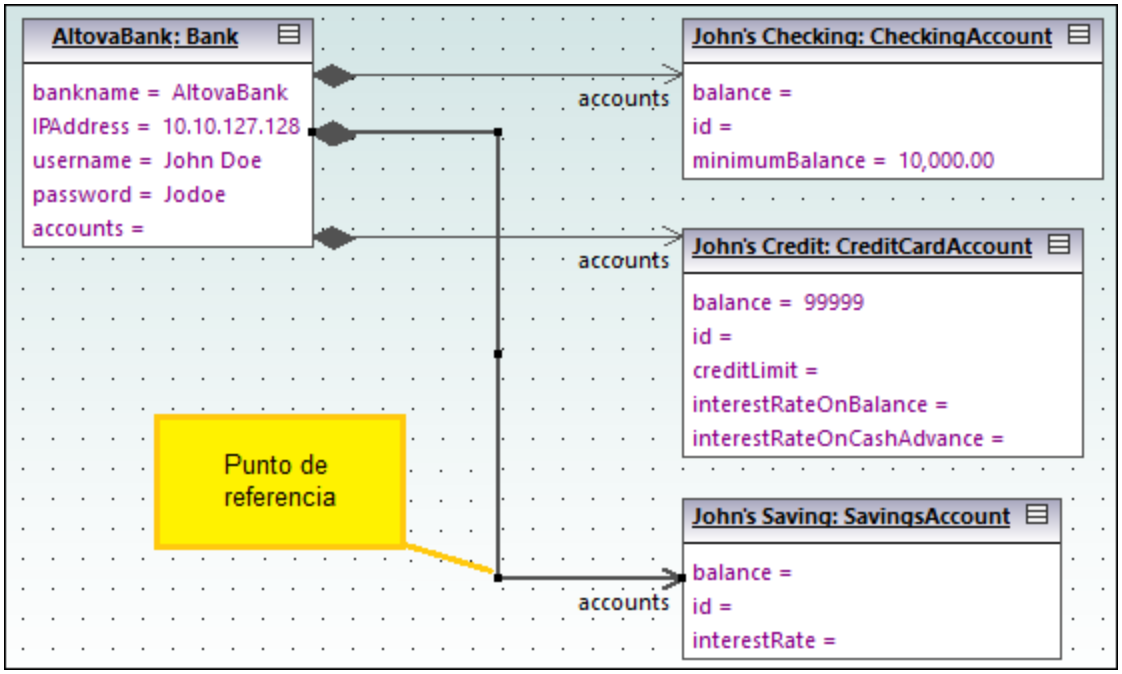
3. Haga clic en el botón **EspecificaciónDeInstancia**  de la barra de herramientas y pase el puntero por encima del objeto `John's Credit: CreditAccount`. Verá que el puntero adopta la forma del signo +.
4. Arrastre el puntero desde el objeto `John's Credit: CreditAccount` hasta `AltovaBank: Bank` para crear un vínculo entre los dos.
5. En la ventana *Propiedades*, junto a `clasificador`, seleccione la entrada **Account - Bank**.
6. Por último, usando los métodos descritos anteriormente, cree un vínculo entre el objeto `AltovaBank: Bank` y el objeto `John's Saving: SavingsAccount`.



Observe que los cambios aplicados al tipo de asociación en cualquier diagrama de clases se reflejan automáticamente en el diagrama de objetos.

Cambiar el formato de las líneas de asociación/vínculo en un diagrama

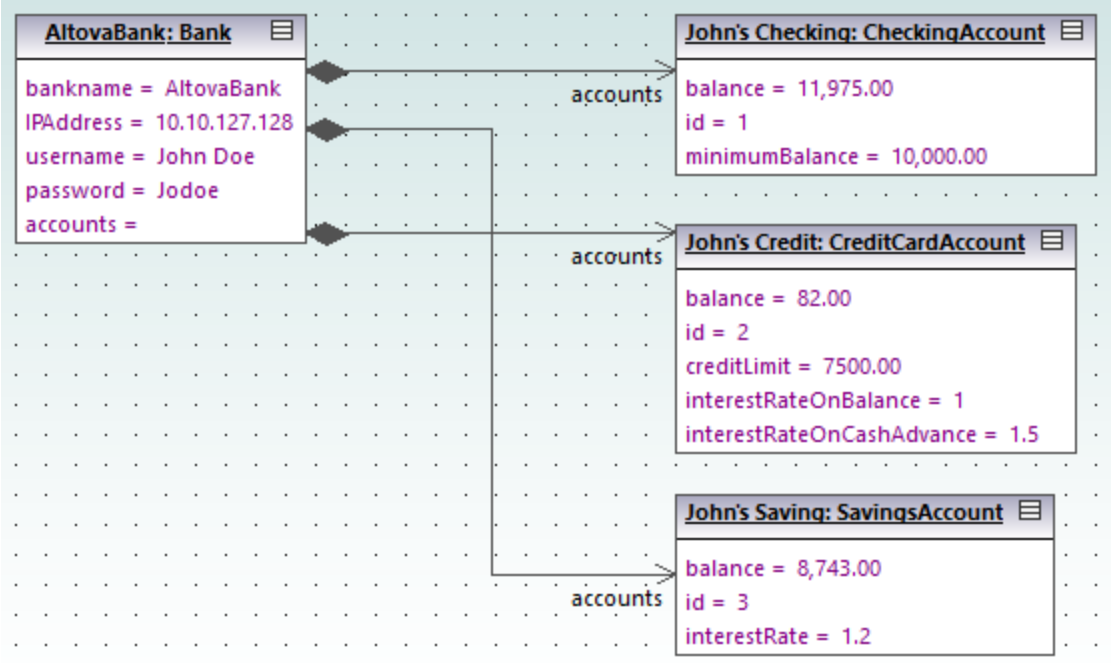
Para cambiar el formato de los vínculos que unen los objetos haga clic en la línea y arrástrela según corresponda. Para cambiar la posición de la línea (horizontal y verticalmente) arrastre el punto de referencia de la línea (*imagen siguiente*).



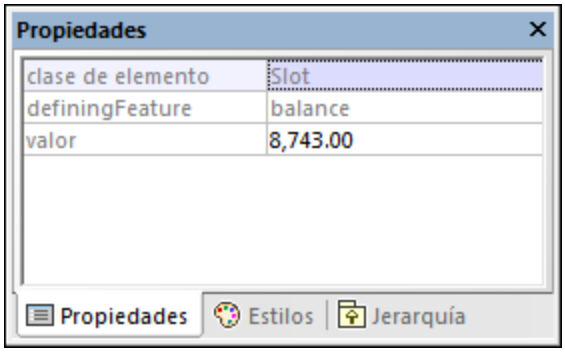
Vínculos en un diagrama de objetos

Introducir datos de muestra en objetos

El valor de instancia de un atributo/una propiedad en un objeto se denomina *slot*. Para describir el estado de un objeto haga doble clic en los slots e introduzca datos de instancia de muestra después del carácter "=". Por ejemplo:



Los slots de los objetos se pueden rellenar en la ventana *Propiedades*. Basta con seleccionar el objeto en el diagrama y después introducir el texto correspondiente junto al campo `valor` en la ventana *Propiedades*.



2.5 Diagramas de componentes

En este apartado del tutorial aprenderá a:

- crear dependencias de realización entre clases y componentes,
- cambiar el aspecto de las líneas utilizadas en el diagrama,
- agregar dependencias de uso a una interfaz y
- a usar la notación de interfaz de tipo bola.

Para continuar ejecute UModel y abra el proyecto **BankView-start.ump** (*véase* *Abrir el proyecto del tutorial*). Este proyecto incluye varios diagramas de objetos predefinidos que nos servirán de ejemplo para este tutorial. El requisito para este apartado del tutorial es haber completado el apartado Crear clases derivadas para crear la clase `SavingsAccount`.

Crear dependencias de realización entre clases y componentes

En la ventana *Árbol de diagramas*, expanda la entrada `Diagramas de componentes` y haga doble clic en el icono del diagrama "BankView realization". Este diagrama ya contiene el componente `BankView` y varias clases que están conectadas a él con dependencias de tipo "RealizaciónDeComponente". El texto `(desde bankview)` que aparece dentro de cada clase indica el nombre del paquete al que pertenece la clase.

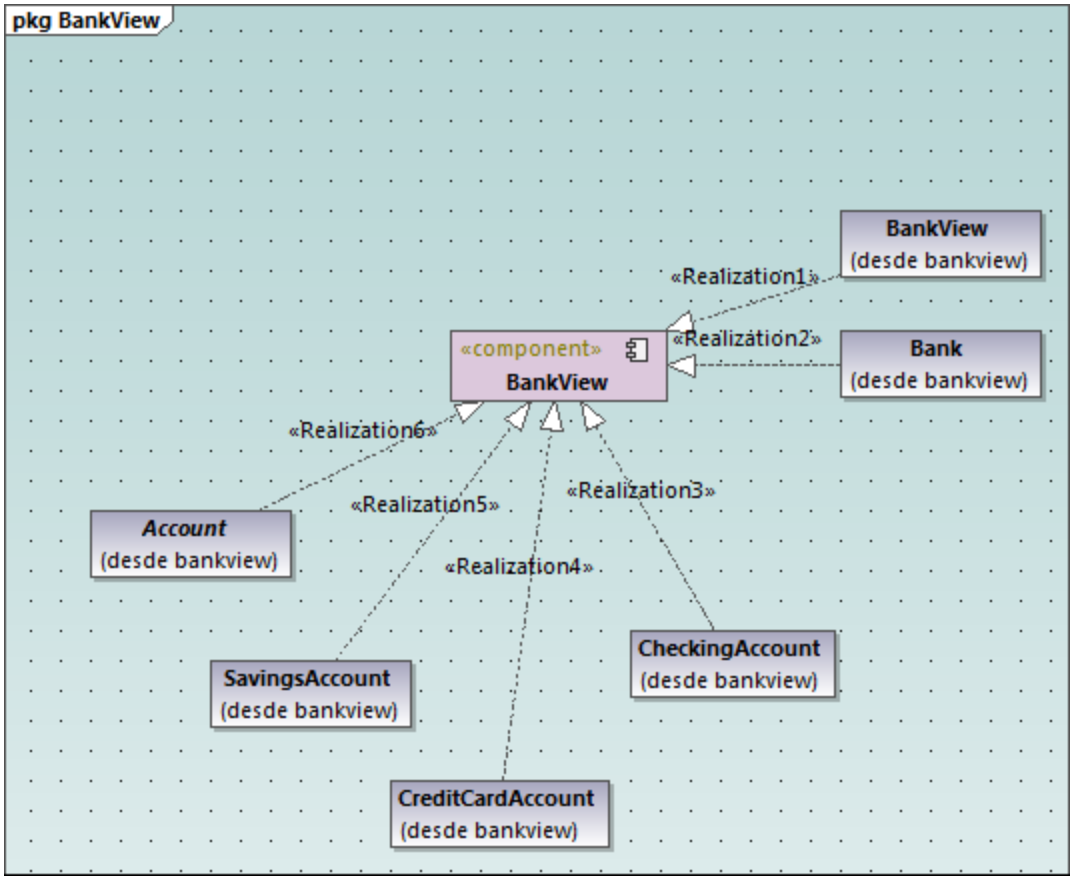
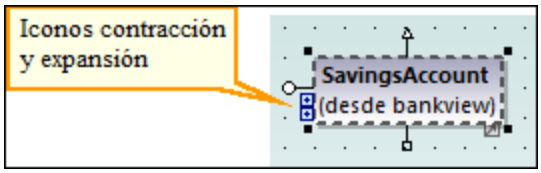


Diagrama "Bank View realization"

Ahora vamos a agregar una clase nueva al diagrama y a crear una dependencia de realización entre la clase nueva y el componente `BankView`.

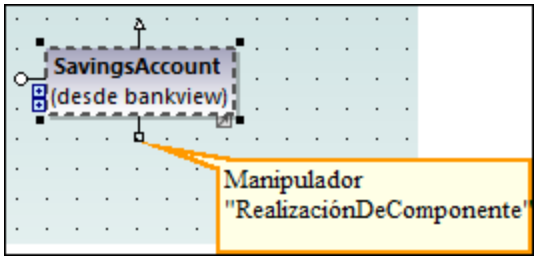
1. En la ventana *Estructura del modelo*, navegue hasta la clase `SavingsAccount` del paquete `bankview`. Si falta esta clase, entonces debe completar las instrucciones del apartado Crear clases derivadas del tutorial para crearla.
2. Arrastre la clase `SavingsAccount` desde la ventana *Estructura del modelo* hasta el diagrama.

La clase aparece por defecto con todos sus compartimentos expandidos. Haga clic en los iconos de contracción/expansión (-/+) situados en el borde izquierdo de la clase para mostrar u ocultar sus propiedades y operaciones.

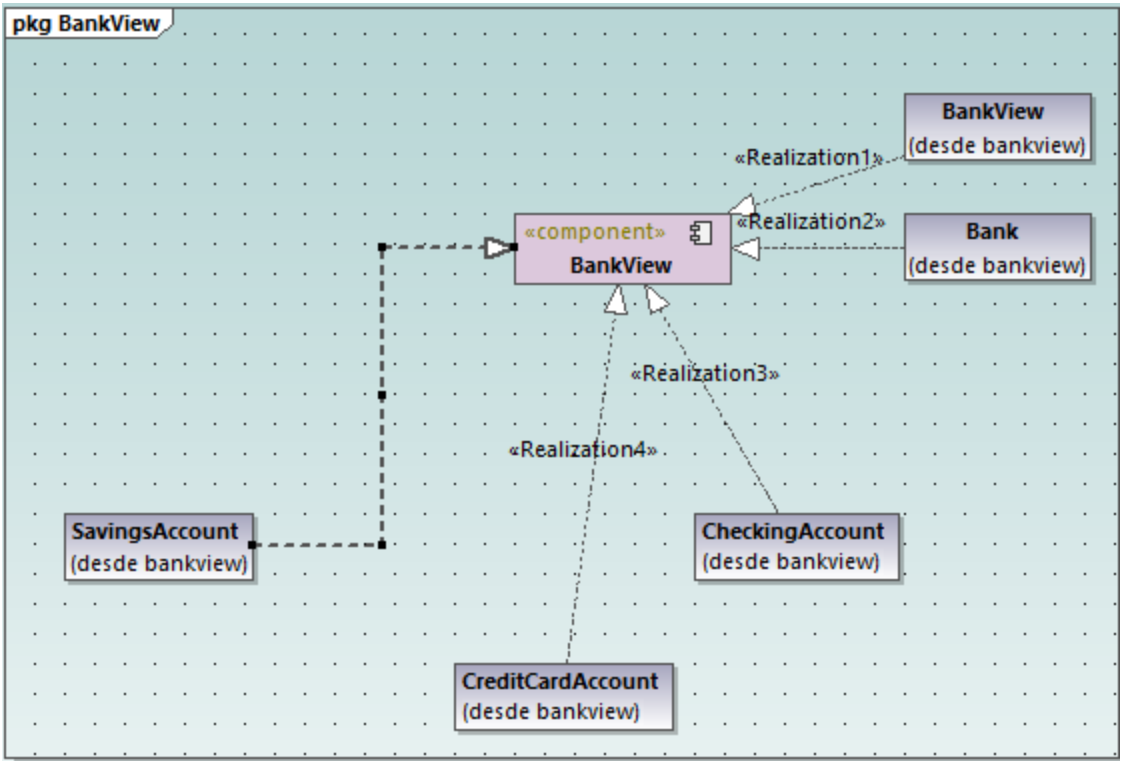


Hay dos maneras de crear una dependencia de realización entre la clase y el componente:

- haciendo clic en el botón **Realización**  de la barra de herramientas y arrastrando el puntero desde la clase `SavingsAccount` hasta el componente `BankView`.
- haciendo clic en el manipulador "RealizaciónDeComponente" de la clase y arrastrando el puntero hasta el componente `BankView`.



Como resultado se crea una dependencia de realización entre `SavingsAccount` y `BankView`.



Para poner nombre a la nueva línea de dependencia (p. ej. "Realization5") primero debe seleccionar la línea y después podrá teclear el nombre nuevo. Otra opción es seleccionar la línea y editar el valor de la propiedad `nombre` en la ventana *Propiedades*.

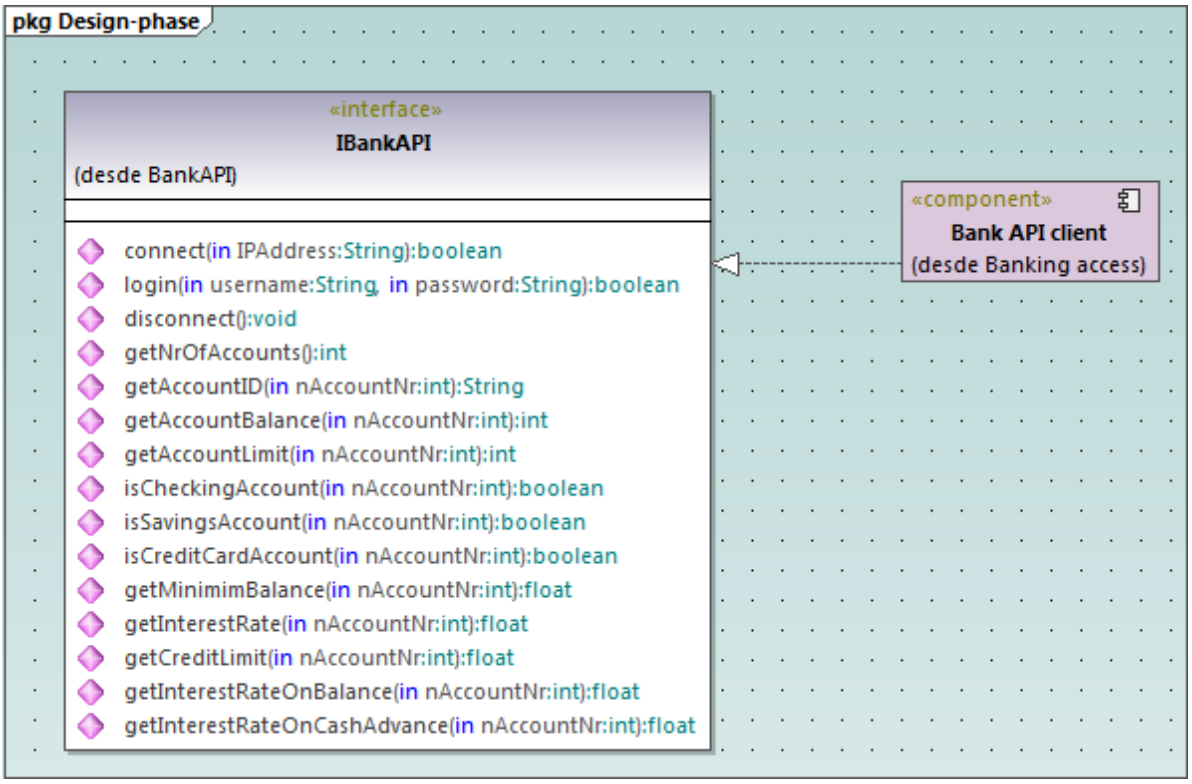
Cambiar el aspecto de las líneas del diagrama

Ahora aprenderemos a cambiar el aspecto de las líneas, que pueden ser curvas o rectas:

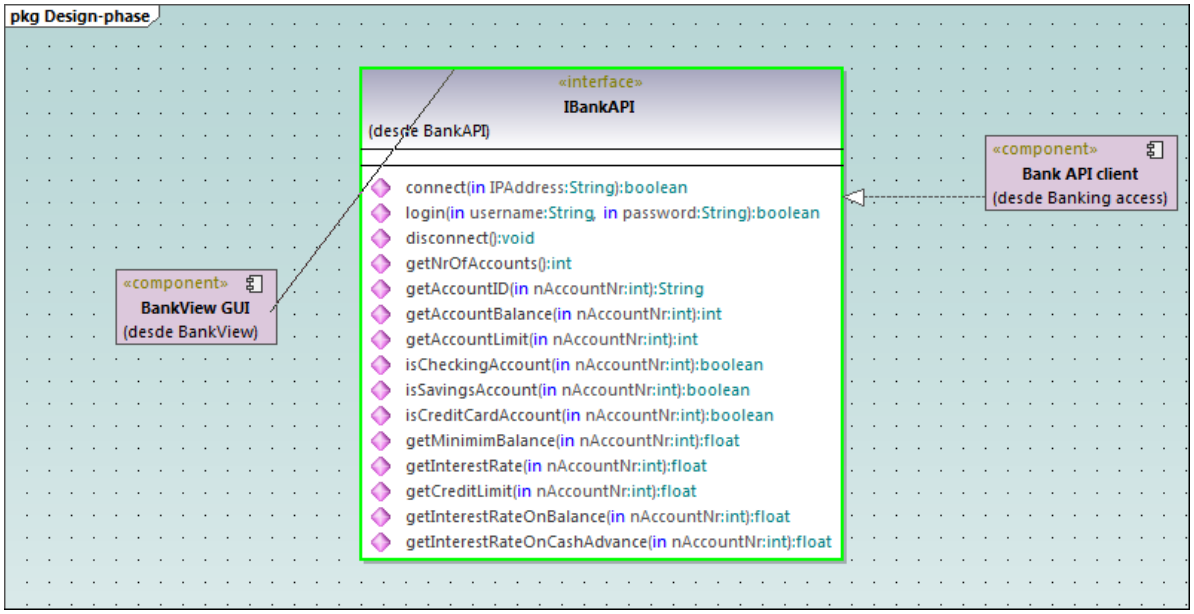
1. Seleccione la línea que acabamos de crear para conectar `SavingsAccount` y `BankView`.
2. Haga clic en el botón **Línea directa** de la barra de herramientas.

Agregar dependencias de uso a una interfaz

1. En la ventana *Estructura del modelo* navegue hasta **Root | Design-phase** y haga doble clic en el icono del diagrama "Overview". Esto abre el diagrama de componentes "Overview", que muestra las dependencias entre componentes e interfaces definidas en el sistema.

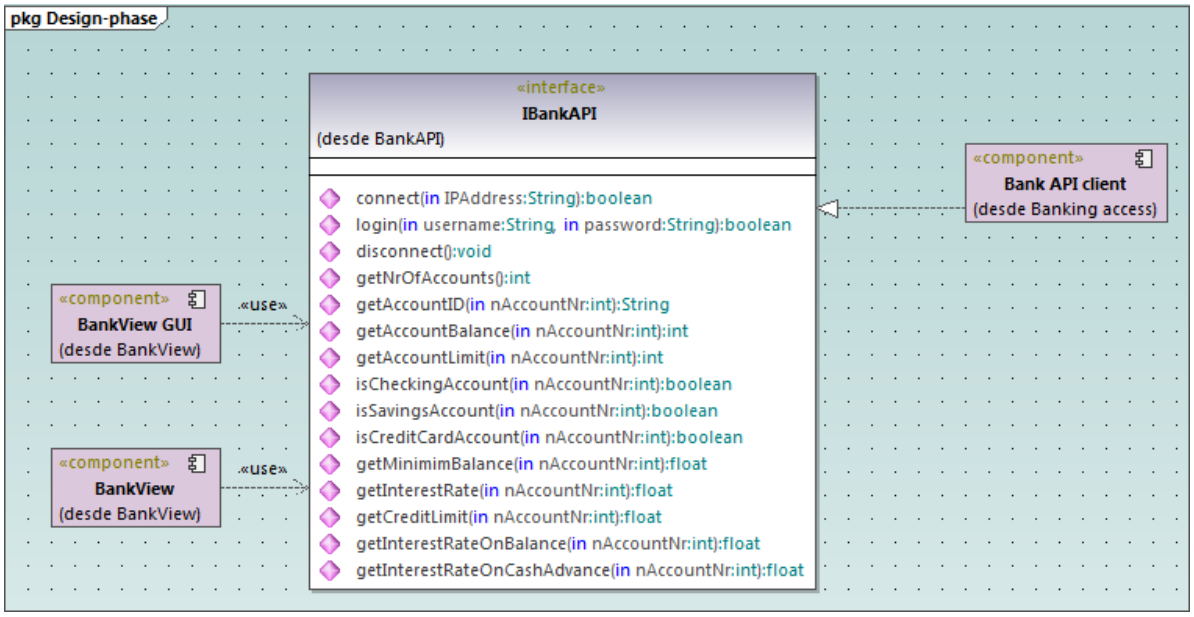


2. En la ventana *Estructura del modelo* navegue hasta **Root | Component View | BankView** y arrastre el paquete `BankView GUI` hasta el diagrama de componentes "Overview".
3. Ahora arrastre también el paquete `BankView` hasta el diagrama de componentes "Overview".
4. Haga clic en el botón **Utilización**  de la barra de herramientas y arrastre el puntero desde el paquete `BankView GUI` hasta la interfaz `IBankAPI`.



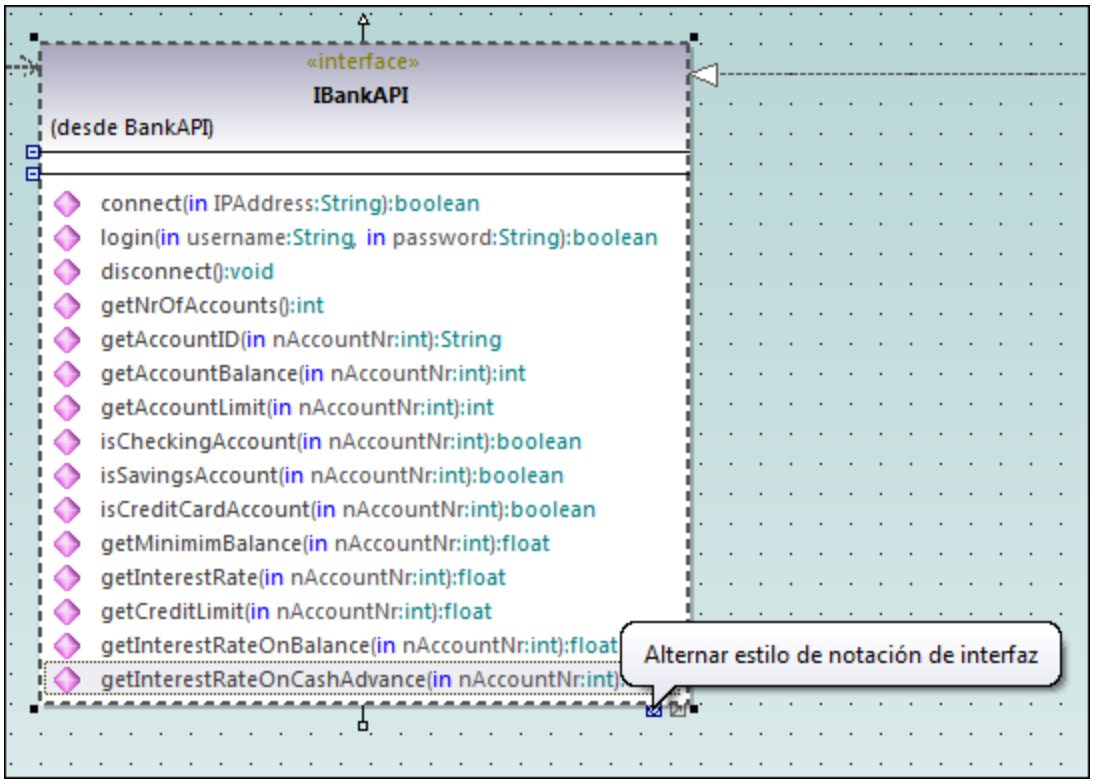
5. Haga lo mismo con el paquete `BankView`.

Como puede ver en la imagen siguiente, ahora los dos paquetes tienen una dependencia de utilización con la interfaz. Es decir, los paquetes `BankView` y `BankView GUI` necesitan a la interfaz `IBankAPI`. Esta interfaz la aporta el paquete `Bank API Client`.

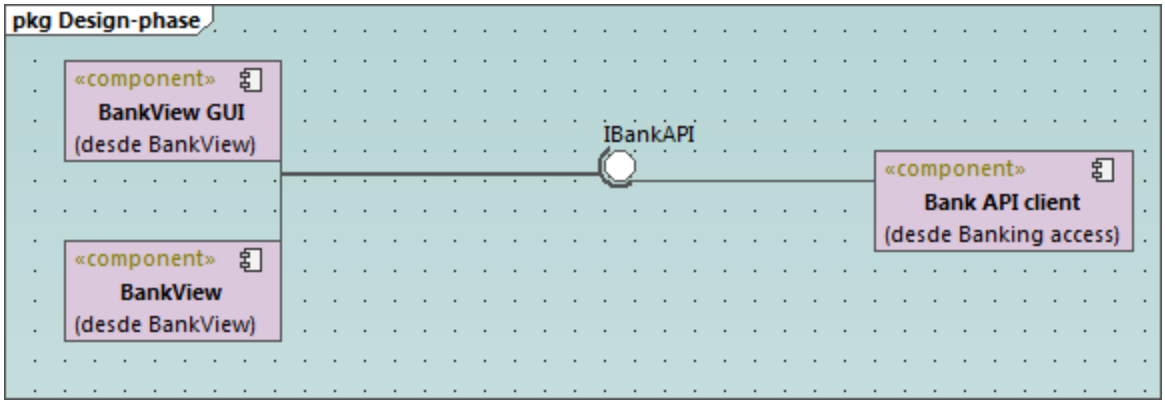


Usar notaciones de tipo bola

Si quiere puede convertir la notación del diagrama en una notación de tipo bola. Esto se hace seleccionando la interfaz y después haciendo clic en el botón **Alternar estilo de notación de interfaz** situado en la esquina inferior derecha.



Cuando se hace clic en ese botón, el diagrama se presenta con notación de tipo bola.



Para volver al estilo de notación anterior basta con seleccionar la interfaz y hacer clic otra vez en el botón **Alternar estilo de notación de interfaz**.

2.6 Diagramas de implementación

Este apartado del tutorial explica:

- cómo agregar una dependencia entre dos artefactos en un diagrama de implementación,
- cómo agregar elementos en un diagrama de implementación,
- cómo incrustar artefactos en un nodo de un diagrama de implementación y
- cómo crear elementos de artefacto (p. ej. propiedades, operaciones, artefactos anidados, etc.).

Para continuar, ejecute UModel y abra el proyecto **BankView-start.ump** (*véase* *Abrir el proyecto del tutorial*).

Agregar una dependencia entre dos artefactos en un diagrama de implementación

En la ventana *Árbol de diagramas*, dentro de `Diagramas de implementación`, haga doble clic en el icono situado junto al diagrama "Artifacts" para abrirlo. Como se ve en la imagen siguiente, este diagrama muestra la manifestación de los componentes `Bank API client` y `BankView` a sus correspondientes archivos Java `.jar` compilados.

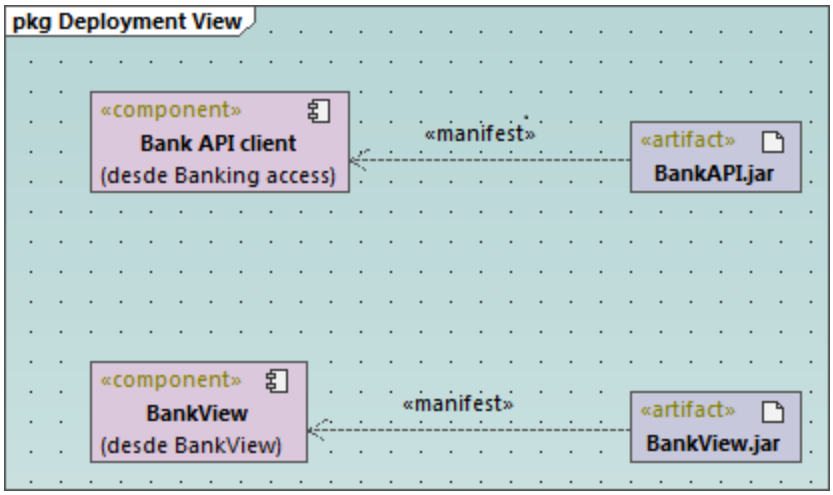


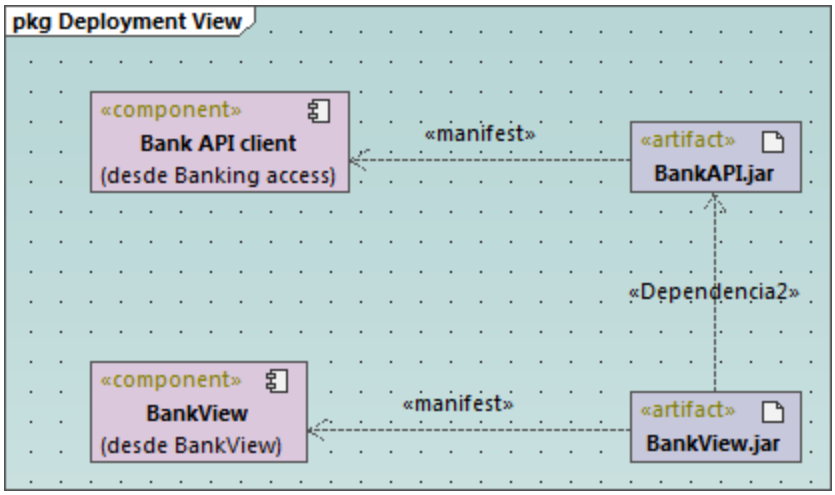
Diagrama "Artifacts"

Estas manifestaciones se crearon con una técnica parecida a la utilizada en apartados previos del tutorial para crear relaciones:

1. Haga clic en el botón **Manifestación** de la barra de herramientas.
2. Arrastre el puntero desde el artefacto hasta el componente.

Usando esta misma técnica añadiremos ahora una dependencia entre los dos archivos `.jar`:

1. Haga clic en el botón **Dependencia** de la barra de herramientas.
2. Arrastre el puntero desde el artefacto `BankView.jar` hasta el artefacto `BankAPI.jar`.
3. Seleccione la línea de dependencia y escriba "Dependencia2".



Agregar elementos a un diagrama de implementación

En la ventana *Árbol de diagramas*, dentro de `Diagramas de implementación`, haga doble clic en el icono situado al diagrama "Deployment" para abrir este diagrama. Observará que este diagrama está incompleto y solamente contiene un nodo que representa un equipo doméstico (*Home PC*). Más adelante añadiremos más elementos al diagrama.

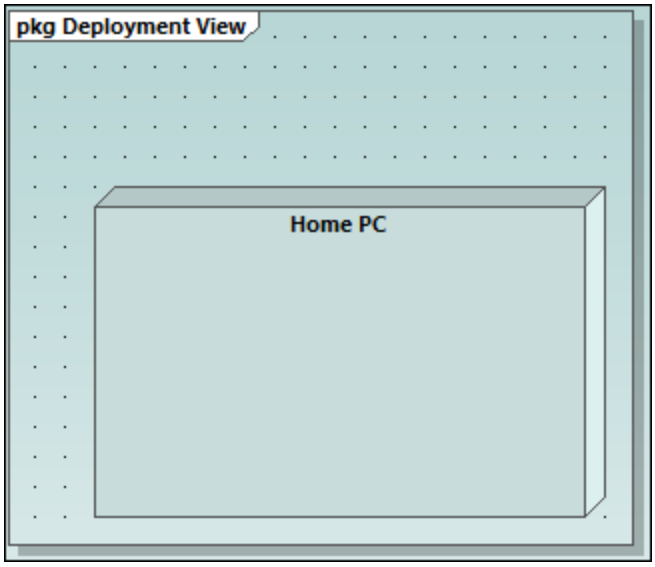
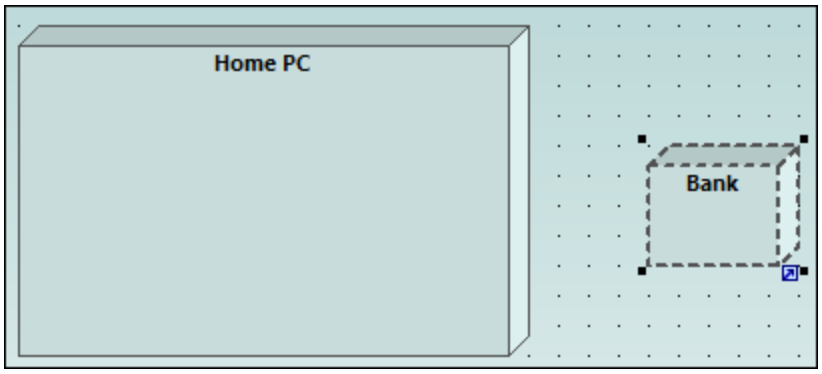


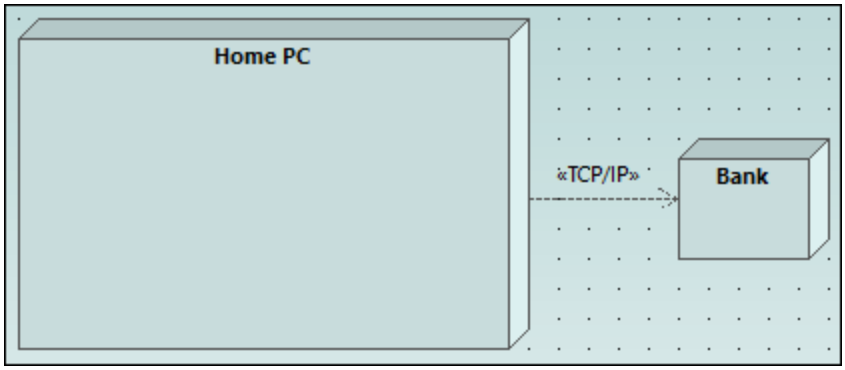
Diagrama "Deployment"

Imaginemos que nuestro objetivo es ilustrar una conexión TCP/IP entre el equipo doméstico y un banco. Para ello deberemos añadir los elementos necesarios:

1. Haga clic en el botón **Nodo** de la barra de herramientas y haga clic a la derecha del nodo "Home PC" para insertar el nuevo nodo.
2. Cambie el nombre del nuevo nodo por "Bank" y arrastre sus bordes para agrandarlo.

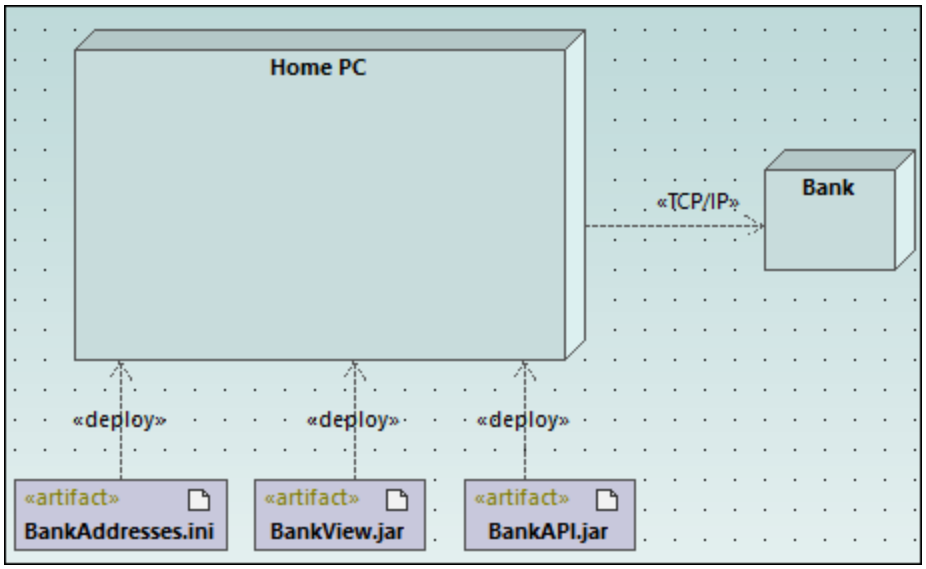


3. Haga clic en el botón **Dependencia** de la barra de herramientas y arrastre el puntero desde el nodo "Home PC" hasta el nodo "Bank". Esto crea una dependencia entre los dos nodos.
4. Seleccione la línea de dependencia y póngale el nombre "TCP/IP" (esto también puede hacerse editando el valor de la propiedad `nombre` en la ventana *Propiedades*).

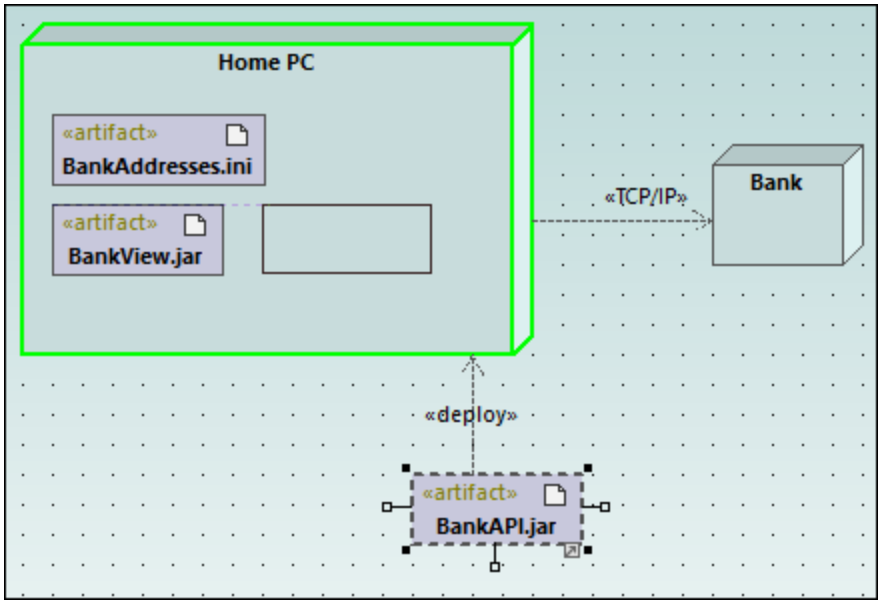


Incrustar artefactos

En la ventana *Estructura del modelo* expanda el paquete "Deployment View" y después arrastre estos tres artefactos hasta el diagrama: **BankAddresses.ini**, **BankAPI.jar** y **BankView.jar**. El proyecto está preconfigurado para que incluya dependencias de implementación entre estos artefactos y el nodo "Home PC", por lo que todas estas dependencias se pueden ver en el diagrama:

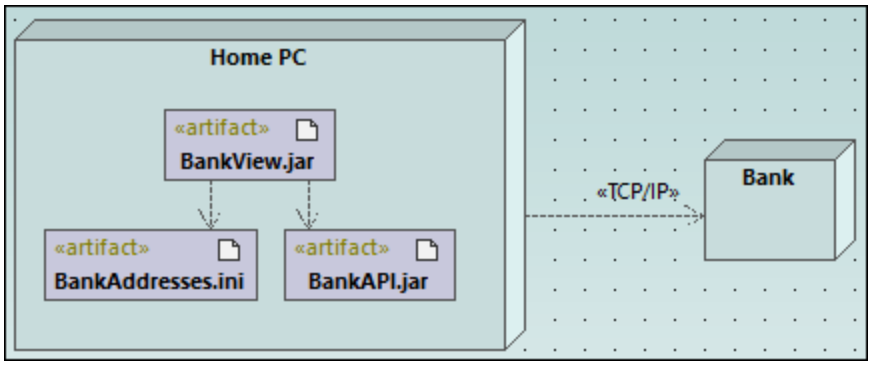


También puede incrustar los artefactos en el nodo "Home PC" (arrastrando uno a uno todos los artefactos hasta el nodo). Observe que las dependencias de implementación ya no se ven en el diagrama pero siguen existiendo lógicamente.



Los artefactos incrustados en el nodo también pueden compartir dependencias:

1. Haga clic en el botón **Dependencia** de la barra de herramientas y, sin dejar de pulsar la tecla **Ctrl**, arrastre el puntero desde el artefacto "BankView.jar" hasta "BankAddresses.ini".
2. Ahora, sin dejar de pulsar la tecla **Ctrl**, arrastre el puntero desde el artefacto "BankView.jar" hasta el artefacto "BankAPI.jar".



Nota: cuando se saca un artefacto de un nodo arrastrándolo hasta la superficie del diagrama se crea automáticamente una dependencia de implementación.

Crear elementos de artefacto (propiedades, operaciones, artefactos anidados)

En UML los artefactos pueden estar compuestos de propiedades, operaciones y otros elementos, como artefactos anidados. Para crear estos elementos, haga clic con el botón derecho en el artefacto en la ventana *Estructura del modelo* y seleccione el comando correspondiente en el menú contextual (p. ej. **Elemento nuevo | Operación** o **Elemento nuevo | Propiedad**). El elemento nuevo aparecerá anidado debajo del artefacto seleccionado en la ventana *Estructura del modelo*.

2.7 Ingeniería directa (del modelo al código)

Este apartado explica cómo crear un proyecto de UModel nuevo y generar código de programa a partir de él (un proceso conocido como *ingeniería directa*). El proyecto que vamos a crear será muy sencillo y estará formado por una sola clase. También aprenderemos a preparar el proyecto para la generación de código y a comprobar que la sintaxis del proyecto es correcta. Tras generar el código de programa, lo modificaremos fuera de UModel, añadiendo un método nuevo a la clase. Por último, en el siguiente apartado, se explica cómo combinar cambios realizados en el código con el proyecto de UModel original (un proceso conocido como *ingeniería inversa*).

El lenguaje de generación de código utilizado en este tutorial es Java, pero para los demás lenguajes de generación de código pueden seguirse instrucciones parecidas.

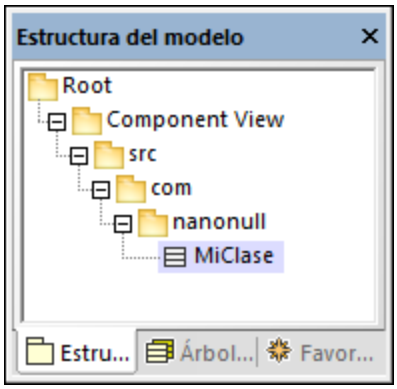
Crear un proyecto de UModel nuevo

Para crear un proyecto de UModel nuevo haga clic en el comando **Archivo | Nuevo** (o pulsando **Ctrl+N** o el botón **Nuevo** de la barra de herramientas).

El proyecto recién creado solamente contiene los paquetes predeterminados "Root" y "Component View". Estos dos paquetes no se pueden eliminar ni renombrar. "Root" es el nivel superior de agrupación para todos los demás paquetes y elementos del proyecto. El paquete "Component View", por su parte, es necesario para los procesos de ingeniería de código y suele almacenar componentes UML que serán realizados por clases o interfaces del proyecto. Sin embargo, hasta ahora no hemos creado ninguna clase.

Para empezar vamos a diseñar la estructura de nuestro programa:

1. Haga clic con el botón derecho en el paquete "Root" de la ventana *Estructura del modelo* y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "src".
2. Haga clic con el botón derecho en "src" y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "com"
3. Haga clic con el botón derecho en "com" y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "nanonull".
4. Haga clic con el botón derecho en "nanonull" y seleccione **Elemento nuevo | Clase** en el menú contextual. Cambie el nombre de la nueva clase por "MiClase".



Preparar el proyecto para la generación de código

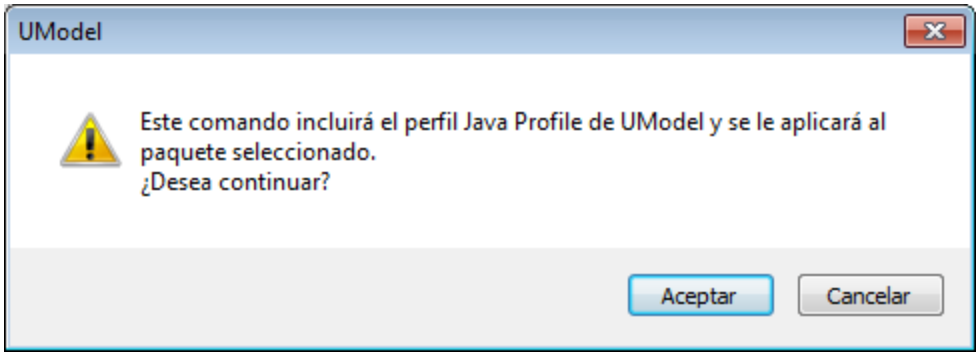
Para generar código a partir de un modelo de UModel es necesario cumplir varios requisitos:

- Debe tener definido un paquete raíz de espacio de nombres Java, C# o VB.NET.
- Debe existir un componente que sea realizado por todas las clases o interfaces para las que se debe generar código.
- El componente debe tener asignada una ubicación física (un directorio). El código se generará en dicha ubicación.
- El componente debe tener habilitada la propiedad `usar para ingeniería de código`.

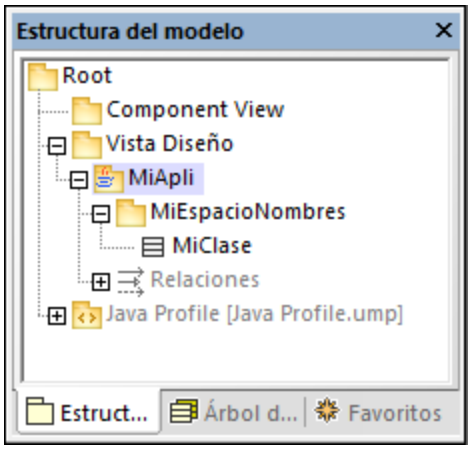
Estos requisitos se explican más abajo con más detenimiento, pero recuerde que puede comprobar en todo momento si el proyecto cumple con estos requisitos con solo usar la función de validación: haciendo clic en el comando **Proyecto | Revisar la sintaxis del proyecto** (**F11**)

Si se valida el proyecto en este momento, la ventana Mensajes emite el error de validación "`No se encontró la raíz de espacio de nombres. Utilice el menú contextual (submenú "Ingeniería de código") en la estructura del modelo para definir un paquete como raíz de espacio de nombres.`" Para resolver este problema asignaremos el paquete src como raíz de espacio de nombres:

1. Haga clic con el botón derecho en el paquete "src" y seleccione **Ingeniería de código | Establecer como raíz de espacio de nombres de Java** en el menú contextual.
2. Cuando la aplicación pida confirmación para incluir el perfil Java Profile de UModel en el paquete, haga clic en **Aceptar**.

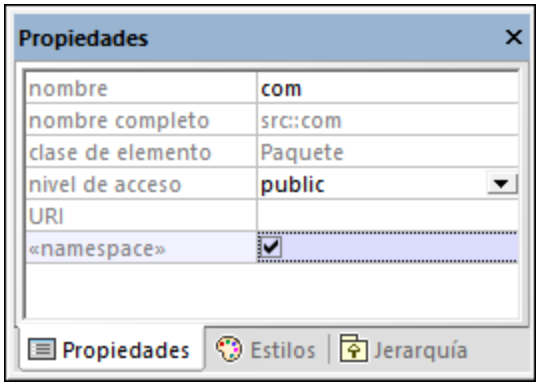


Observe que ahora el paquete tiene un icono distinto ( ) que nos indica que este paquete es una raíz de espacio de nombres de Java. Además, se añadió el perfil Java Profile al proyecto.



El espacio de nombres propiamente dicho se puede definir de la siguiente manera:

1. Seleccione el paquete "com" en la ventana *Estructura del modelo*.
2. En la ventana *Propiedades* habilite la propiedad `<<namespace>>`.

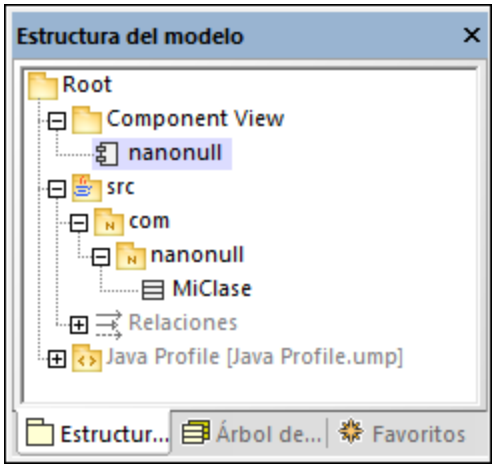


3. Repita el paso anterior con el paquete "nanonull".

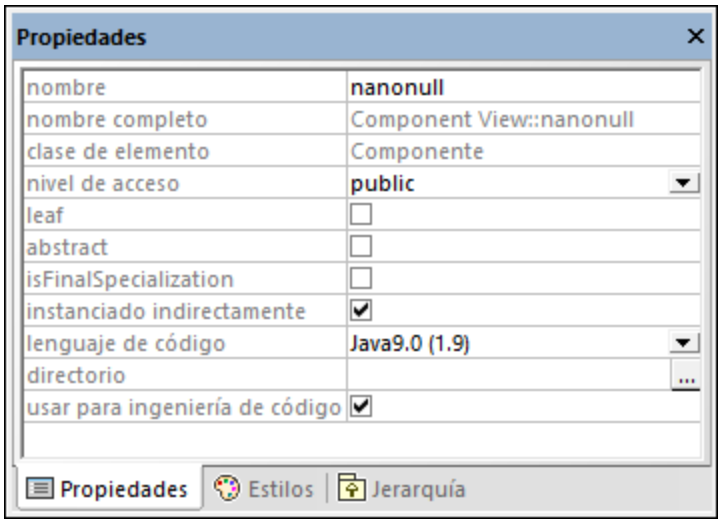
Observe que los paquetes "com" y "nanonull" ahora tienen un icono distinto ( ) que nos indica que estos paquetes son espacios de nombres.

Otro requisito para la generación de código es que los componentes sean realizados por una clase o una interfaz como mínimo. En UML un componente es una pieza del sistema. En UModel el componente nos permite especificar el directorio de generación de código y otras opciones de configuración. Si validamos el proyecto en este momento, aparecerá un mensaje de advertencia en la ventana Mensajes: "*MiClase no tiene una RealizaciónDeComponente para un componente. No se generará código*". Para resolver este problema basta con agregar un componente al proyecto:

1. En la ventana *Estructura del modelo* haga clic con el botón derecho en "Component View" y seleccione **Elemento nuevo | Componente** en el menú contextual.
2. Cambie el nombre del nuevo componente por "nanonull".



3. En la ventana *Propiedades* cambie el valor de la propiedad `directorio` por el directorio donde se debe generar el código (p. ej. "src\com\nanonull"). Observe que la propiedad `usar` para ingeniería de código está habilitada, lo cual es otro requisito imprescindible para la generación de código.



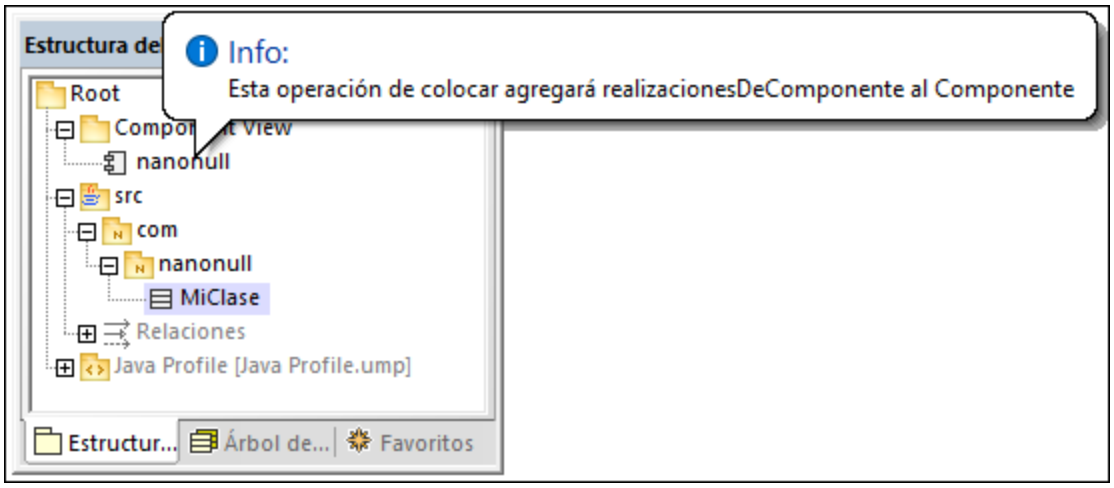
4. Elija un nombre para el proyecto de UModel y guárdelo en un directorio (en este ejemplo: **C:\UModelDemo\Tutorial.ump**).

Nota: el directorio de generación de código puede ser absoluto o relativo al proyecto .ump. Si es relativo, como en este ejemplo, una ruta como **src\com\nanonull** crearía todos los directorio en el mismo directorio en el que se guardó el proyecto de UModel.

En este caso hemos preferido generar código en una ruta de acceso que incluye el nombre de espacio de nombres para evitar mensajes de advertencia porque UModel muestra advertencias de validación si el componente está configurado para generar código Java en un directorio cuyo nombre no coincida con el nombre del espacio de nombres. En este ejemplo, el componente "nanonull" tiene la ruta de acceso "C:\UModelDemo\src\com\nanonull", por lo que no se emitirán advertencias de validación. Si quiere que UModel realice la misma comprobación con C# o VB.NET o si quiere deshabilitar la validación del espacio de nombres para Java, debe seguir estas instrucciones:

1. En el menú **Herramientas** haga clic en el comando **Opciones**.
2. Haga clic en la pestaña *Ingeniería de código*.
3. Marque la casilla correspondiente en el grupo de opciones *Usar espacio de nombres para la ruta del archivo de código*.

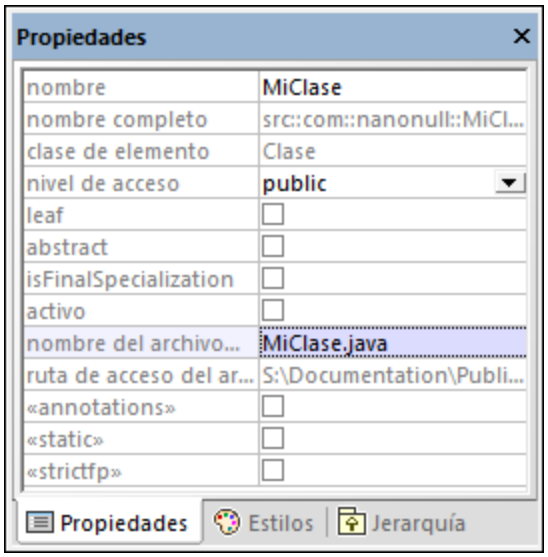
Ahora veamos cómo se puede crear la relación de realización de componente. En la ventana *Estructura del modelo* haga clic en la clase "MiClase" y arrástrela hasta el componente `nanonull`.



De este modo, al componente lo realiza la única clase del proyecto (MiClase). También puede crear la realización de componente desde un diagrama de componentes (*véase* *Diagramas de componentes*).

Lo siguiente que deberíamos hacer es dar un nombre de archivo a las clases o interfaces que participarán en la generación de código. De lo contrario, UModel generará el archivo correspondiente con un nombre de archivo predeterminado y la ventana Mensajes emitirá una advertencia (*no se configuró el nombre del archivo de código. Se generará un nombre predeterminado*). Para solucionar este problema:

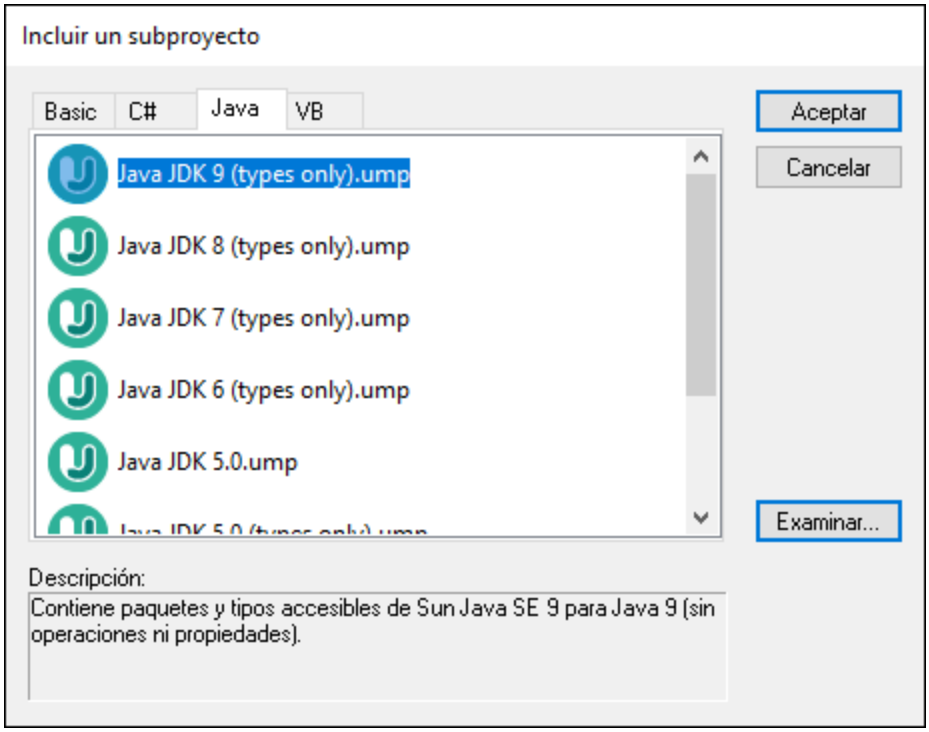
1. Seleccione la clase "MiClase" en la ventana *Estructura del modelo*.
2. En la ventana *Propiedades* cambie el valor de la propiedad `nombre del archivo de código` por el nombre correspondiente (p. ej. `MiClase.java`).



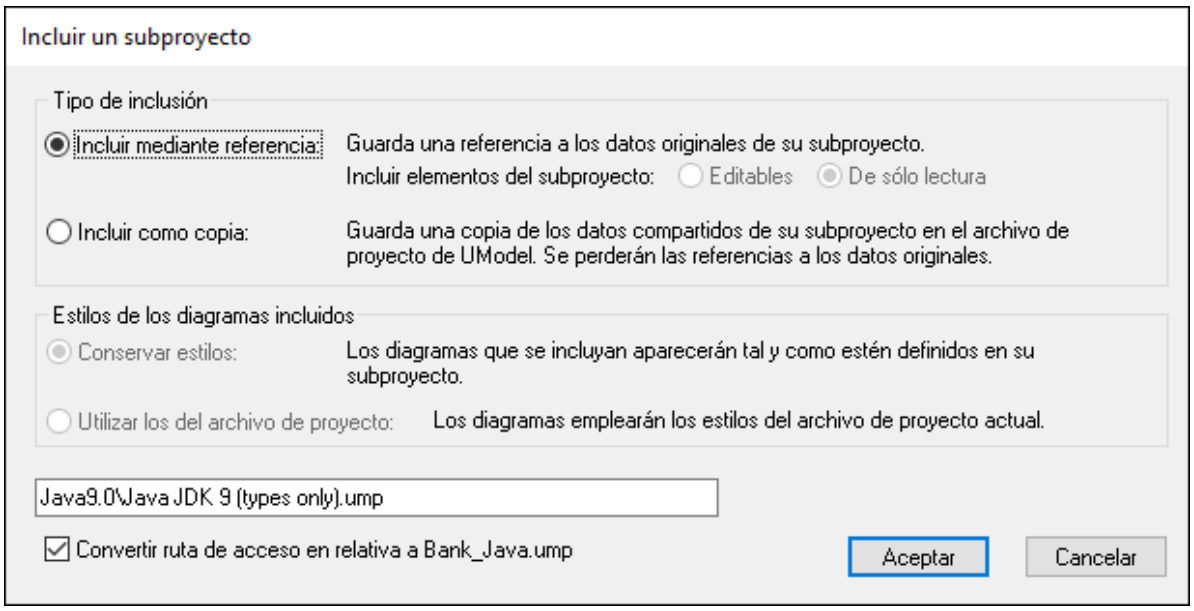
Incluir los tipos JDK

Aunque este paso es opcional, recomendamos que incluya los tipos del lenguaje JDK (Jave Development Kit) como subproyecto de su proyecto de UModel. De lo contrario, los tipos JDK no estarán disponibles cuando cree las clases o interfaces. Esto se puede hacer de la siguiente manera (las mismas instrucciones pueden seguirse para C#, C++ y VB.NET):

1. En el menú **Proyecto** haga clic en **Incluir un subproyecto**.
2. Haga clic en la pestaña **Java** seleccione el proyecto **Java JDK 9 (types only)**.



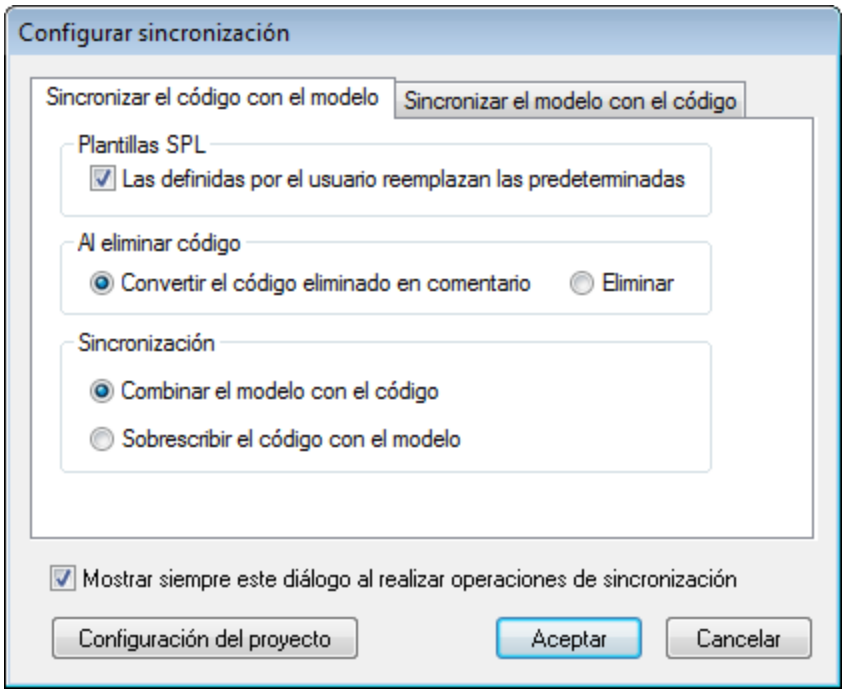
3. Cuando la aplicación pregunte si se incluye mediante referencia o como copia, elija la opción *Incluir mediante referencia*.



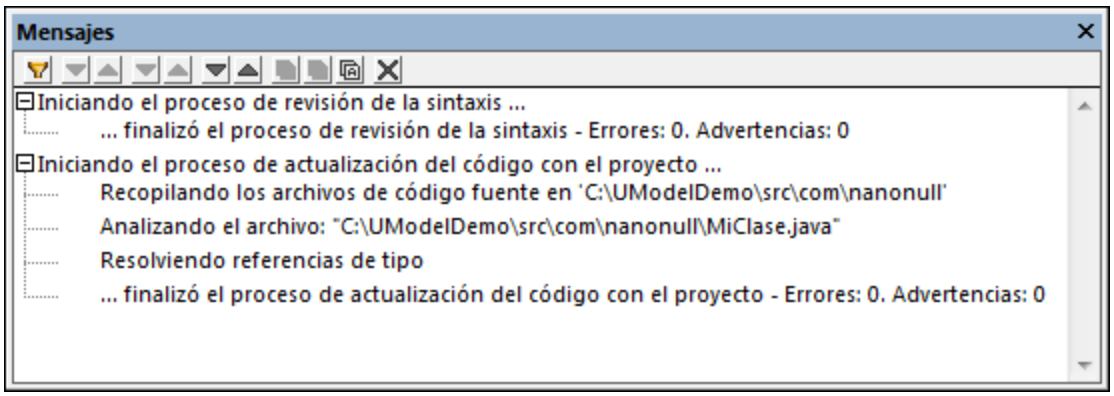
Generar código

Ahora que se cumplen todos los requisitos para la generación de código, podemos empezar el proceso:

1. En el menú **Proyecto** haga clic en el comando **Combinar el código de programa con el proyecto de UModel** (o pulse **F12**).



2. Deje las opciones de sincronización predeterminadas como están y haga clic en **Aceptar**. UModel revisa la sintaxis del proyecto automáticamente y en la ventana Mensajes aparecen los resultados de la revisión:



Modificar el código fuera de UModel

La generación de código de programa es el primero paso para empezar a desarrollar una aplicación o un sistema de software. En un proyecto real, el código sufrirá un gran número de modificaciones antes de llegar a ser un programa completo. Siguiendo con nuestro ejemplo, ahora abriremos el archivo generado **MiClase.java**

en un editor de texto y añadiremos un método nuevo a la clase, como se muestra a continuación. El archivo **MiClase.java** tendrá este aspecto:

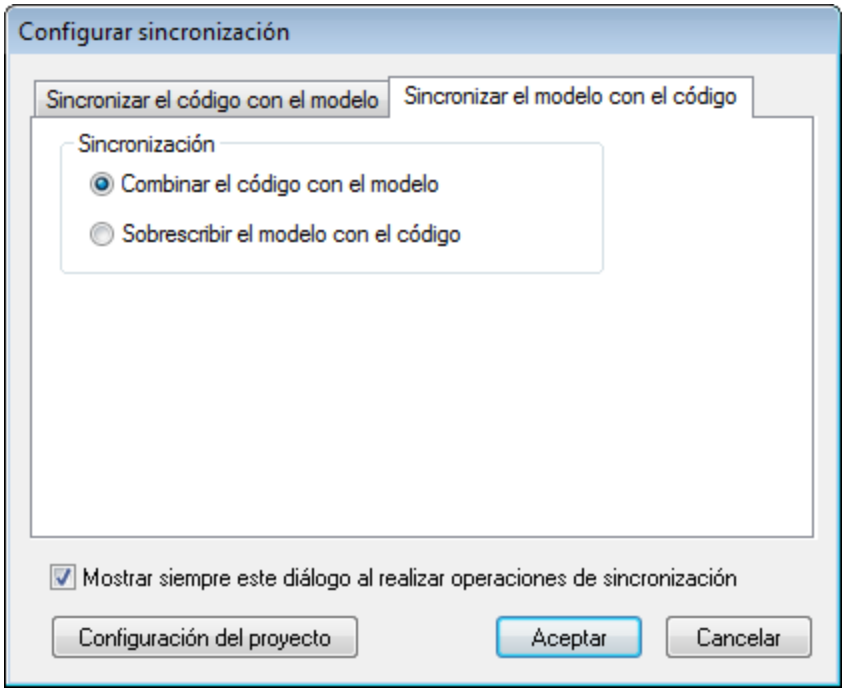
```
package com.nanonull;
public class MyClass{
   public float sum(float num1, float num2){
     return num1 + num2;
   }
}
```

MiClase.java

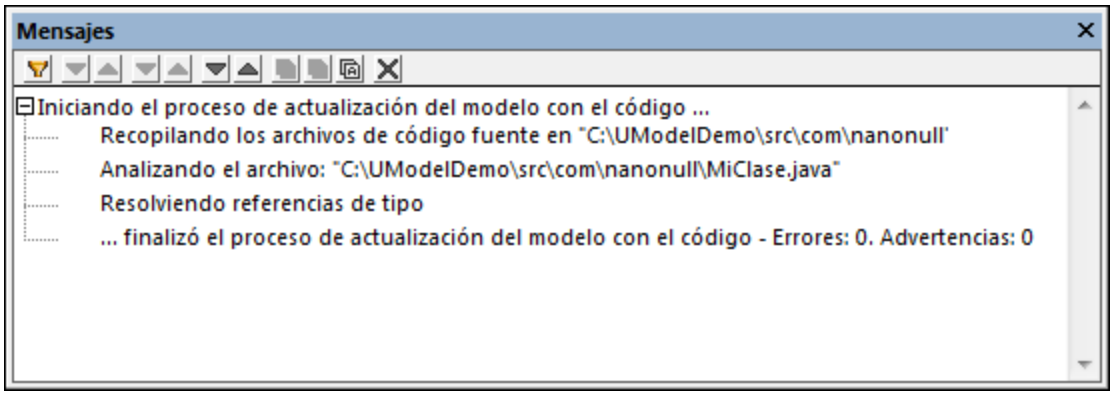
Combinar cambios realizados en el código con el modelo original

Ahora podemos combinar los cambios realizados en el código con el código original:

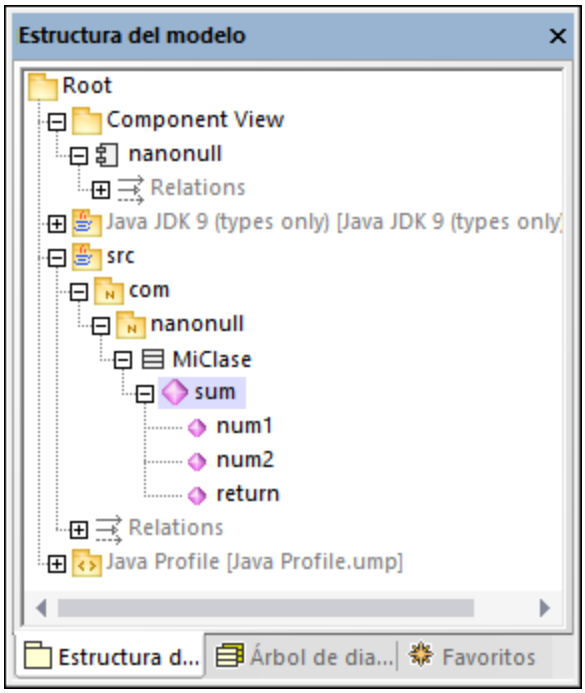
1. En el menú **Proyecto** haga clic en el comando **Combinar el proyecto de UModel con el código de programa** (o pulse **Ctrl+F12**).



2. Deje las opciones de sincronización predeterminadas como están y haga clic en **Aceptar**. UModel revisa la sintaxis del proyecto automáticamente y en la ventana Mensajes aparecen los resultados de la revisión:



La clase `sum` (que se obtuvo mediante ingeniería inversa desde el código) aparece ahora en la ventana *Estructura del modelo*.



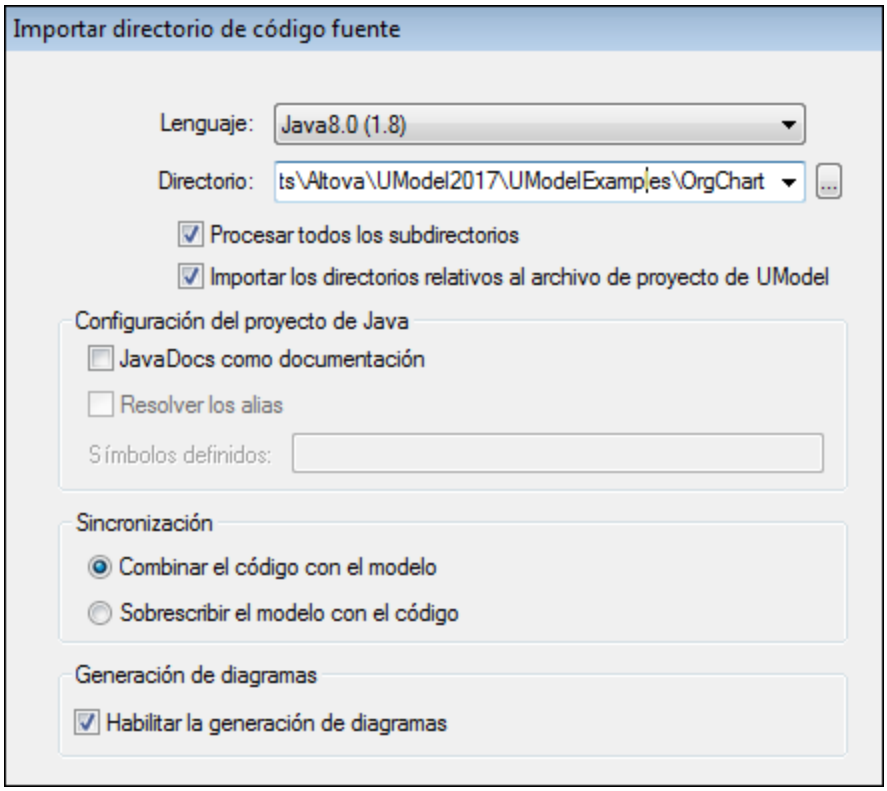
2.8 Ingeniería inversa (del código al modelo)

En este apartado del tutorial explicamos cómo importar código de programa desde un directorio hasta un proyecto de UModel nuevo (ingeniería inversa). También aprenderá cómo agregar una clase nueva al modelo, prepararla para la generación de código y combinar los cambios con el código Java original (ingeniería directa). Aunque en este apartado trabajaremos con código Java, el proceso es el mismo cuando se importa código C# o VB.NET.

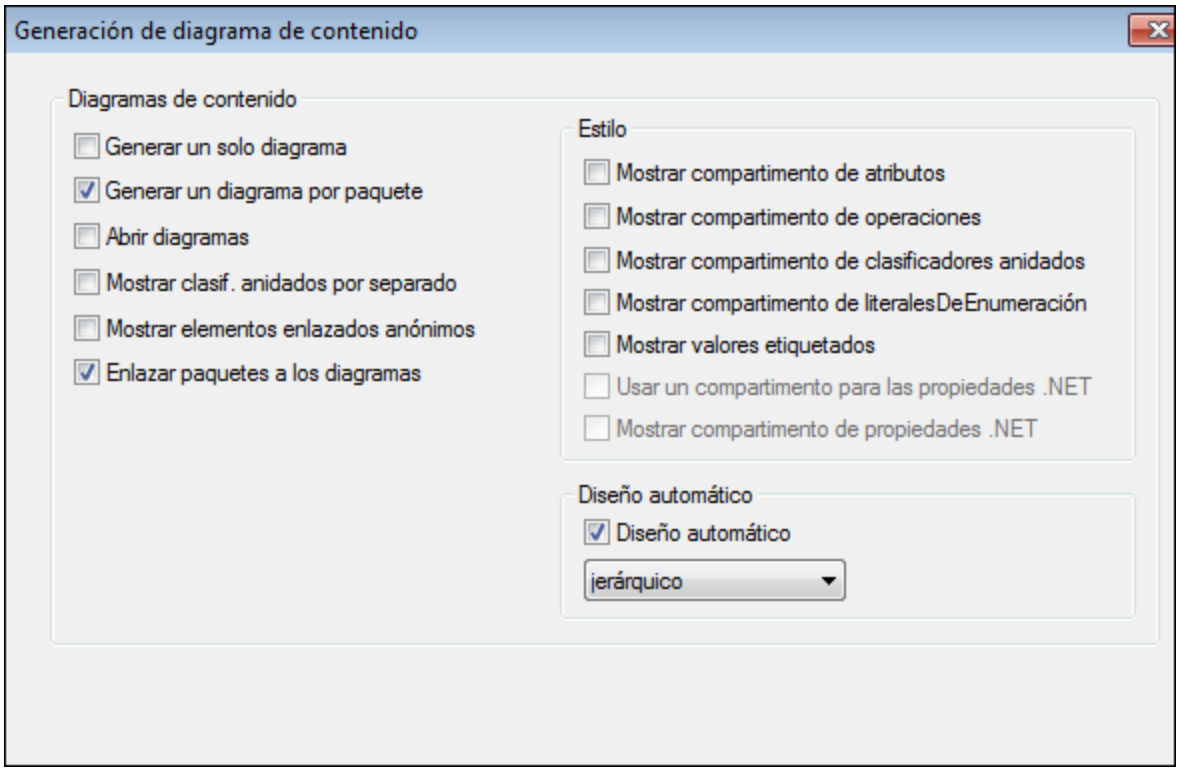
Nota: el código Java de muestra que se utiliza en este apartado está en un archivo ZIP en la ruta de acceso **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\OrgChart.zip**. Antes de empezar el tutorial deberá descomprimirlo en el mismo directorio.

Importar código desde un directorio

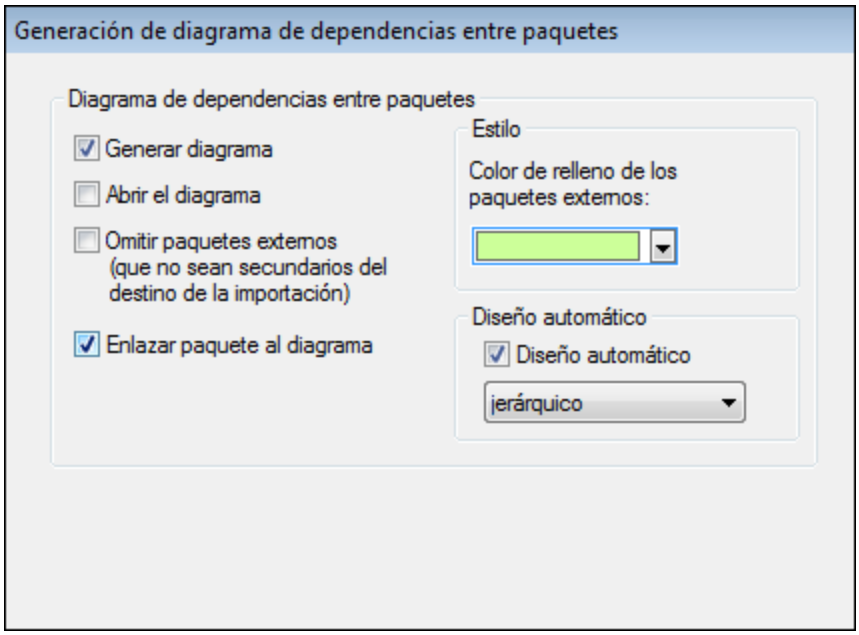
1. En el menú **Archivo** haga clic en el comando **Nuevo**.
2. En el menú **Proyecto** haga clic en **Importar directorio de código fuente**.
3. Seleccione el lenguaje del código fuente (en nuestro ejemplo es Java).
4. Haga clic en el botón **Examinar** ..., seleccione el directorio **OrgChart** que descomprimió antes de empezar el tutorial y haga clic en **Siguiente**. Observe que está marcada la casilla *Habilitar la generación de diagramas*, lo cual ordena a UModel que genere Diagramas de clases y Diagramas de paquetes a partir del código fuente.



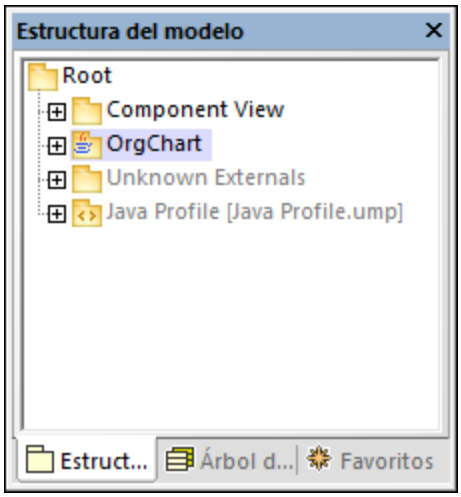
5. En la siguiente pantalla marque la casilla *Generar un diagrama por paquete*. Esto ordena a UModel que cree un diagrama nuevo por cada paquete. Las opciones de estilo de los diagramas se pueden configurar aquí o más adelante.



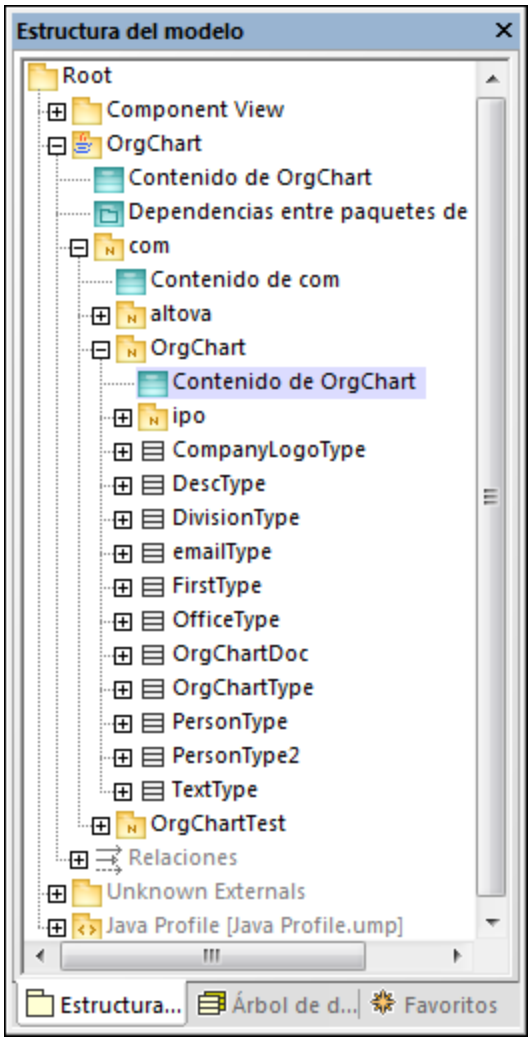
6. Haga clic en **Siguiente** para continuar. En la siguiente pantalla puede definir las opciones de generación de dependencias entre paquetes.



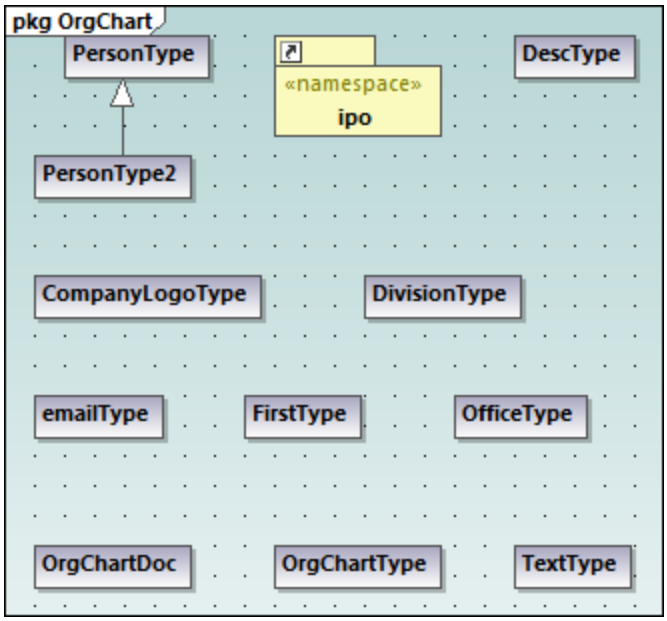
7. Haga clic en **Finalizar**. Cuando la aplicación lo solicite, guarde el modelo nuevo en un directorio del sistema. Los datos se analizan y se crea un paquete nuevo llamado "**OrgChart**".



8. Expanda el paquete nuevo y siga expandiendo los subpaquetes hasta llegar al paquete **OrgChart** (**com | OrgChart**). Haga doble clic en el icono del diagrama **Contenido de OrgChart**:



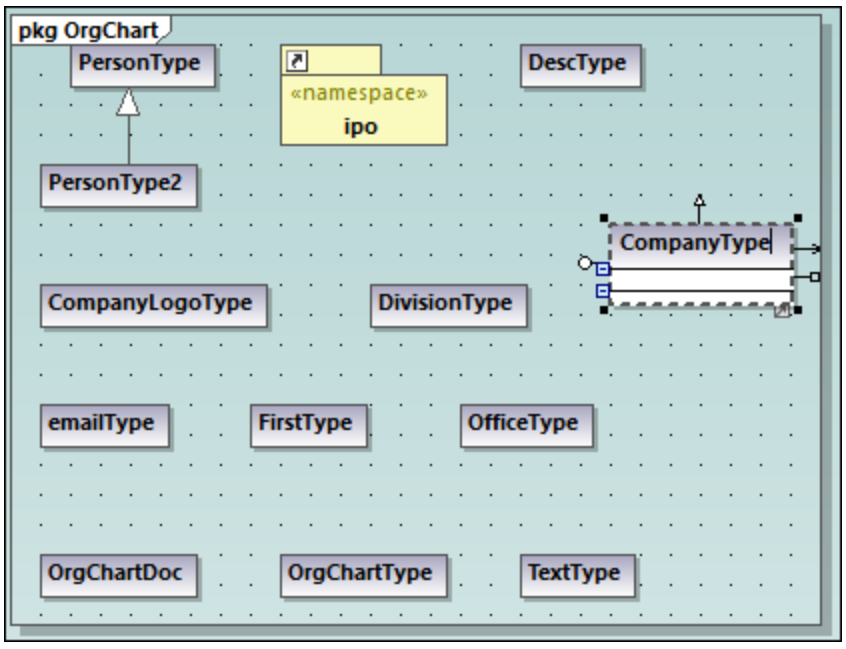
El diagrama "Contenido de OrgChart" aparece ahora en la ventana principal.



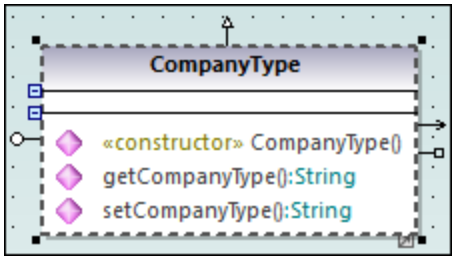
Agregar una clase nueva al diagrama OrgChart

Hasta ahora hemos creado un modelo mediante ingeniería inversa a partir de código Java que ya teníamos. El modelo que hemos creado también incluye varios diagramas que se generaron automáticamente. Ahora vamos a ampliar el modelo con una clase nueva.

1. Haga clic con el botón derecho dentro del diagrama "Contenido de OrgChart" y seleccione **Nuevo/a | Clase** en el menú contextual.
2. Haga clic en el título de la nueva clase e cámbiele el nombre por **CompanyType**.



3. Añada tres operaciones nuevas a la clase (pulsando **F8**): `CompanyType()`, `getCompanyType():String`, `setCompanyType():String`.

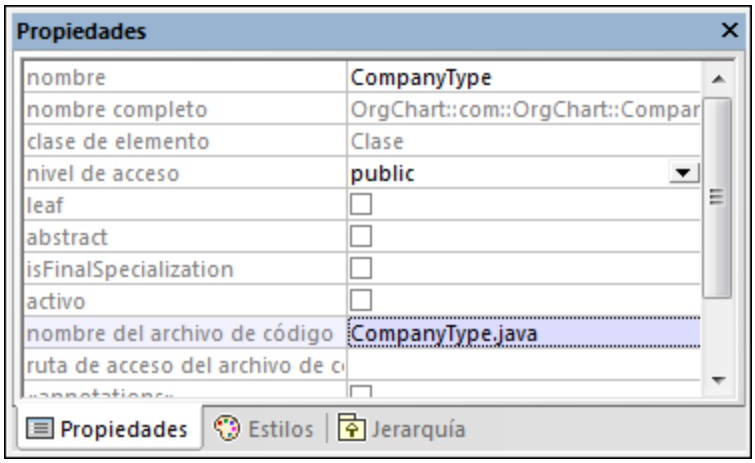


Nota: como el nombre de la clase es `CompanyType`, a la operación `CompanyType()` se le asigna automáticamente el estereotipo `<<constructor>>`.

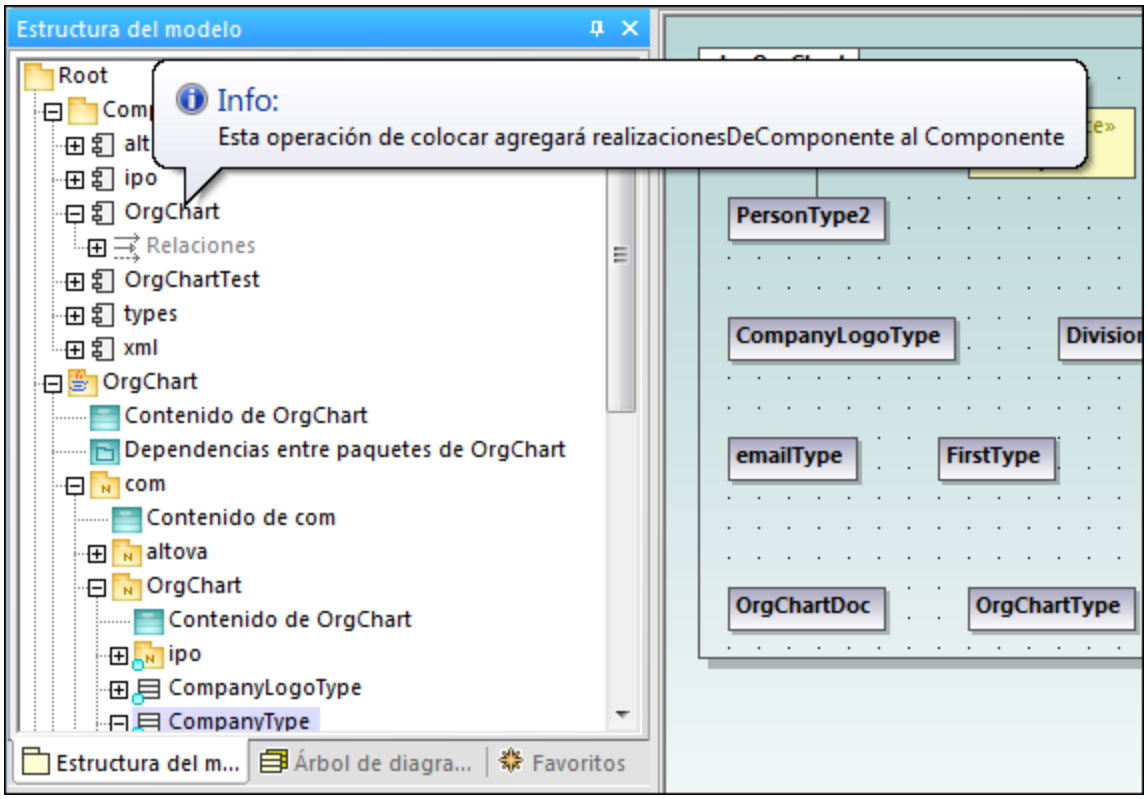
Habilitar la clase nueva para la generación de código

Ahora que el modelo incluye una clase nueva, debemos actualizar el código subyacente para que el modelo y el código estén sincronizados. Sin embargo, si pulsamos F11 para revisar la sintaxis del proyecto en este momento, se emite una advertencia en la ventana Mensajes: *'CompanyType' no tiene una realizaciónDeComponente para un componente. Se generará una RealizaciónDeComponente para el componente OrgChart*. El motivo de esta advertencia es que para poder generar código la nueva clase debe realizar un componente (*véase* *Ingeniería directa (del modelo al código)*). En algunos casos, como en este ejemplo, UModel puede generar la realización necesaria de forma automática. Sin embargo, también puede definir la dependencia de realización manualmente:

1. Seleccione la clase `CompanyType` en el diagrama, busque la propiedad `nombre del archivo de código` en la ventana *Propiedades* e introduzca el valor "CompanyType.java".



2. En la ventana *Estructura del modelo* haga clic en la nueva clase `CompanyType` y arrástrela hacia arriba hasta el componente **OrgChart** situado dentro del paquete "Component View". Si pasa el puntero por encima de un componente aparece una notificación.

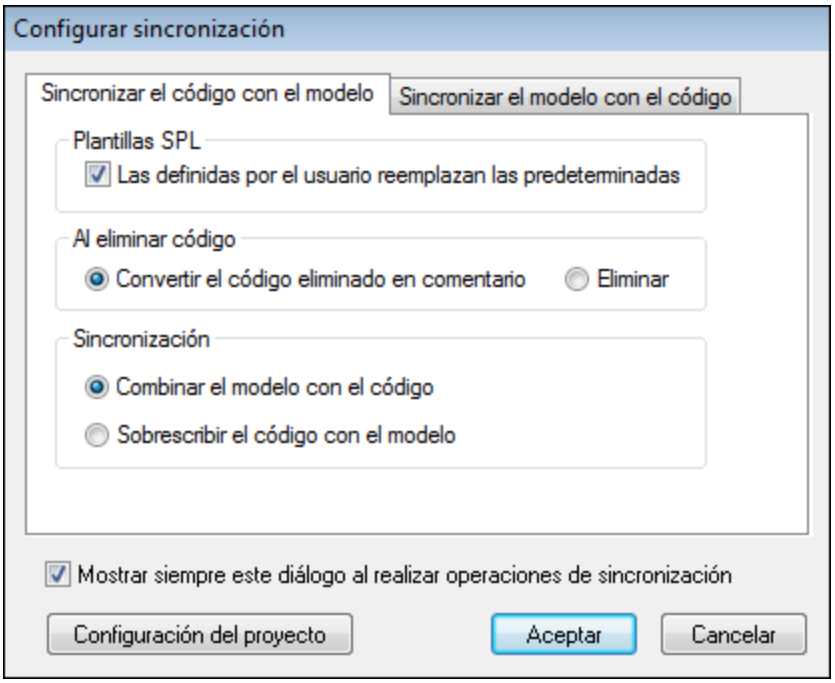


Esto crea una relación de tipo "RealizaciónDeComponente" entre una clase y un componente. También puede crear esta relación dibujando una línea de relación en el diagrama de componentes (*véase* *Diagramas de componentes*) Expanda el elemento `Relaciones` del componente **OrgChart** para ver la relación que acaba de crear.

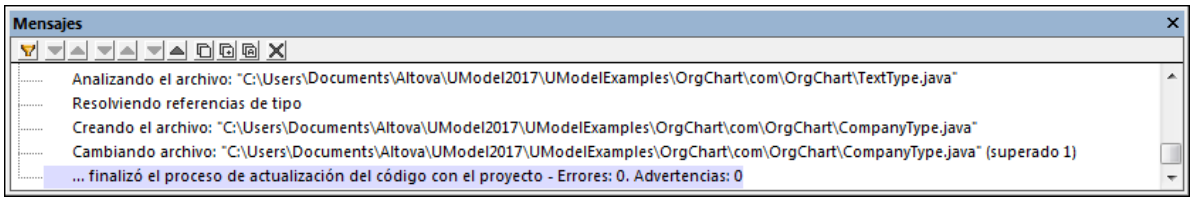
Combinar código de programa con un paquete

En UModel puede generar código a nivel de paquete, de componente o de proyecto (*véase* *Sincronizar el modelo y el código fuente*). En este ejemplo vamos a generar código a nivel de componente:

1. En la ventana *Estructura del modelo* navegue hasta el componente **OrgChart** del paquete "Component View".
2. Haga clic con el botón derecho en el componente **OrgChart** y seleccione el comando **Ingeniería de código | Combinar código de programa con Componente de UModel** del menú contextual.



En la ventana *Mensajes* aparecen los resultados de la revisión sintáctica, así como el estado del proceso de sincronización.



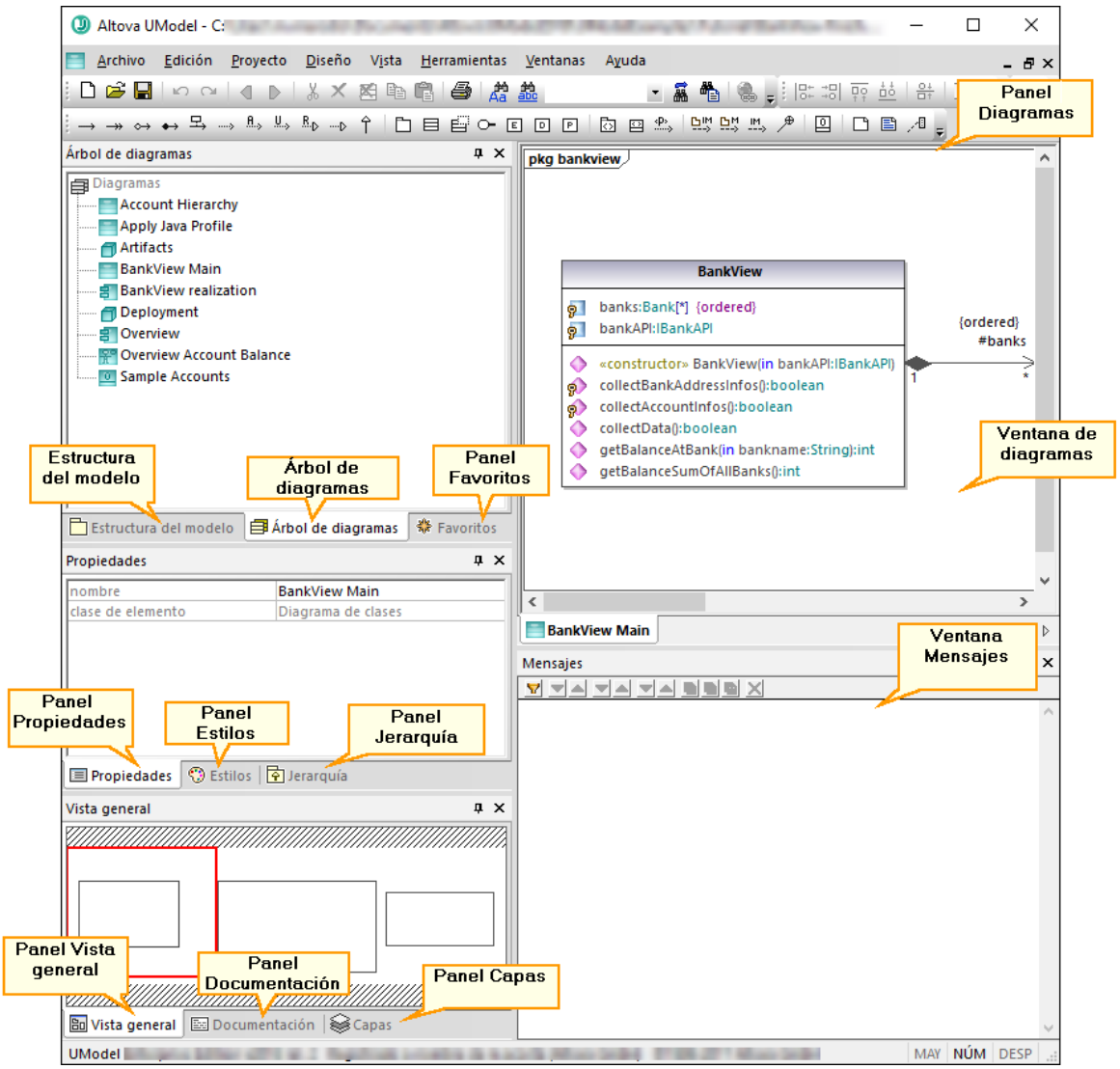
Cuando termine el proceso, la nueva clase **CompanyType.java** estará en la carpeta **... \OrgChart\com\OrgChart\.**

> Todos los métodos y cambios realizados en el código serán eliminados en forma de comentarios o eliminados directamente, dependiendo de la opción seleccionada en el grupo de opciones *Al eliminar código* del cuadro de diálogo "Configurar sincronización".

¡Enhorabuena! Acaba de completar un ciclo de ingeniería de ida y vuelta en UModel.

3 Interfaz gráfica del usuario de UModel

La interfaz gráfica del usuario de UModel consiste en un panel principal de diagramas y varias ventanas auxiliares más pequeñas en las que se puede introducir o visualizar información. El panel *Diagramas* contiene las ventanas de diagrama abiertas. Para recorrerlas use **Ctrl+Tabulador**.



Interfaz gráfica del usuario de UModel

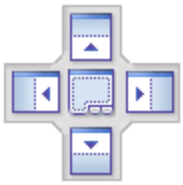
Por defecto, las ventanas auxiliares de la izquierda están acopladas en grupos de tres y la ventana *Mensajes* aparece bajo el *panel Diagramas*. No obstante, puede mover y acoplar o desacoplar cualquier ventana a su gusto. Para realizar búsquedas en todas las ventanas puede usar el cuadro combinado **Buscar** de la barra de herramientas principal o las teclas de acceso rápido **Ctrl+F**. Consulte también Buscar y reemplazar texto.

Para acoplar o desacoplar una ventana:

- Haga clic con el botón derecho en la barra del título y seleccione **Acoplada** (o **Flotante**) en el menú contextual.

Para mover una ventana:

1. Haga clic en la barra del título de la ventana y arrástrela hasta su nueva posición. Verá que aparecen ayudantes de acoplamiento:



2. Arrastre la ventana hasta una de las flechas del ayudante de acoplamiento para colocarla en su nueva posición.

Para restaurar todas las barras de herramientas y ventanas a su estado original:

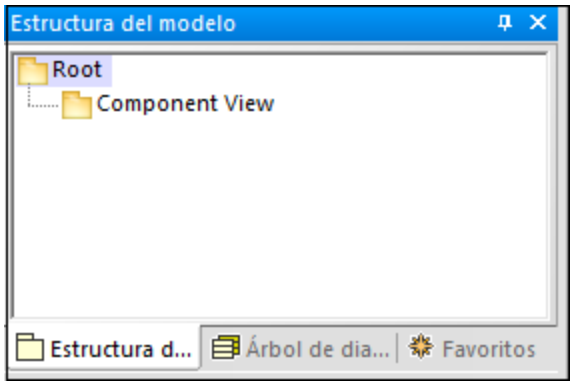
- En el menú **Herramientas** haga clic en **Restaurar barras de herramientas y ventanas**.

En este capítulo explicamos las partes que componen la interfaz gráfica del usuario de UModel, que son las siguientes:

- Ventana *Estructura del modelo*
- Panel *Árbol de diagramas*
- Panel *Favoritos*
- Ventana *Propiedades*
- Panel *Estilos*
- Panel *Jerarquía*
- Ventana *Vista general*
- Panel *Documentación*
- Panel *Capas*
- Ventana *Mensajes*
- Ventana de diagramas
- Panel *Diagramas*

3.1 Ventana Estructura del modelo

La ventana *Estructura del modelo* sirve para visualizar y manipular todos los componentes de los proyectos de UModel.



Panel Estructura del modelo

Al crear un proyecto en UModel hay dos paquetes predeterminados disponibles: "Root" y "Component View". Estos dos paquetes son los únicos que no se pueden renombrar ni eliminar. "Root" sirve de punto de partida para modelar el resto de elementos y "Component View" es necesario para la ingeniería de código.

Puede crear más paquetes, clases, diagramas y jerarquizarlos desde esta misma ventana o directamente desde un diagrama (*véase Crear elementos*). Para ver otras operaciones aplicables a los elementos de la *Estructura del modelo* consulte el apartado Cómo modelar.

Nota: UModel incluye varios proyectos que puede usar para familiarizarse con los conceptos básicos de modelado y con la interfaz gráfica del usuario. Los encontrará aquí: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**.

Mostrar, ocultar y ordenar elementos de la estructura del modelo

Para configurar qué se debe visualizar en la ventana *Estructura del modelo*, así como para ordenar elementos, haga clic con el botón derecho dentro de la ventana y seleccione la opción de menú correspondiente. Para ver todas las acciones que se pueden aplicar a los elementos de la ventana *Estructura del modelo* haga clic con el botón derecho sobre el elemento y observe las opciones del menú contextual.

Contraer y expandir elementos de la estructura del modelo

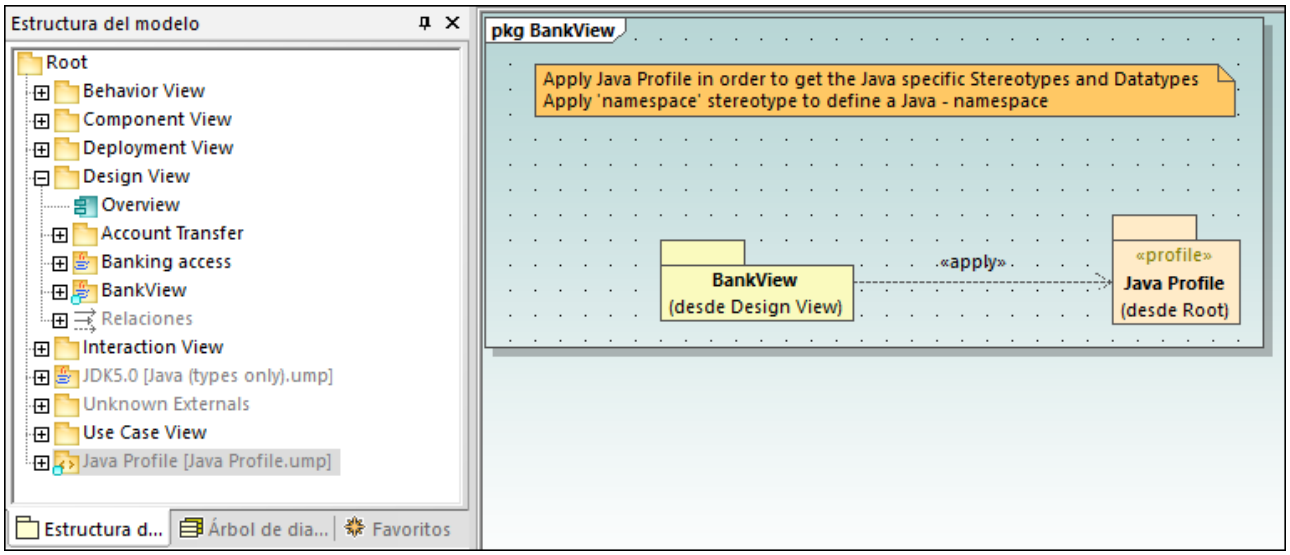
Para expandir elementos (como paquetes) en la ventana *Estructura del modelo*:

- pulse la tecla asterisco (*) para expandir el elemento actual y todos sus elementos secundarios, y
- pulse la tecla más (+) para expandir únicamente el elemento actual.

Para contraer los paquetes, pulse la tecla guion (-) del teclado. Para contraer todos los elementos haga clic en el paquete "Root" y pulse la tecla guion (-). Puede pulsar estas teclas tanto en el teclado estándar como en el numérico.

Identificar elementos de diagrama activos

Cuando un diagrama está abierto en el panel *Diagramas*, la ventana *Estructura del modelo* muestra un punto azul claro junto a la base de los elementos que se muestran en el diagrama activo (*como "Bank View" y "Java Profile" en la imagen siguiente*):



Relación de iconos

En la ventana *Estructura del modelo* puede aparecer un amplio número de iconos que corresponden a los elementos y diagramas del proyecto, a los paquetes de ingeniería de código y a los perfiles importados o subproyectos. En concreto pueden aparecer los siguientes tipos de paquetes:

Icono	Descripción
	Paquete UML estándar
	Paquete raíz del espacio de nombres Java. Se usa para generar o revertir ingeniería de código Java
	Paquete raíz del espacio de nombres C#. Se usa para generar o revertir ingeniería de código C#
	Paquete raíz del espacio de nombres C++. Se usa para revertir ingeniería de código C++
	Paquete raíz del espacio de nombres Visual Basic. Se usa para generar o revertir ingeniería de código VB.NET
	Paquete raíz del espacio de nombres XML Schema. Se usa para generar esquemas XML desde el modelo o para importarlos al modelo (*véase* *Diagramas de esquema XML*)
	Paquete raíz del espacio de nombres de BD. Se usa para importar bases de datos al modelo y cambiar su estructura desde el modelo (*véase* *Trabajar con bases de datos en UModel*)
	Paquete de espacio de nombres (paquete al que se aplica el estereotipo `<<namespace>>`)
	Perfil UML

A continuación mostramos los diagramas que pueden aparecer en la ventana *Estructura del modelo*:

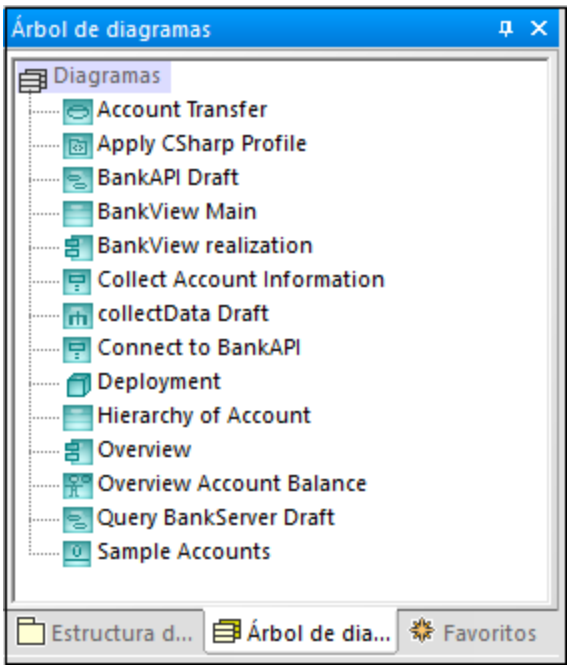
Icono	Descripción
	Diagrama de actividades
	Diagrama de proceso empresarial BPMN 1
	Diagrama de proceso empresarial BPMN 2
	Diagrama de coreografía BPMN 2
	Diagrama de colaboración BPMN 2
	Diagrama de clases
	Diagrama de comunicación
	Diagrama de componentes
	Diagrama de estructura compuesta
	Diagrama de BD
	Diagrama de implementación
	Diagrama global de interacción
	Diagrama de objetos
	Diagrama de paquetes
	Diagrama de perfil
	Diagrama de máquina de estados de protocolos
	Diagrama de secuencia
	Diagrama de máquina de estados
	Diagramas SysML (9 tios de diagramas)
	Diagrama de ciclo de vida
	Diagrama de casos de uso
	Diagrama de esquema XML

A continuación mostramos ejemplos de elementos de modelado UML que pueden aparecer en la ventana *Estructura del modelo*. Para aprender más sobre elementos UML y los tipos de diagramas en los que se dan, consulte el capítulo Diagramas UML.

Icono	Descripción
	Clase
	Propiedad
	Operación
	Parámetro
	Actor
	Caso de uso
	Componente
	Nodo
	Artefacto
	Interfaz
	Instancia de clase (objecto)
	Slot de instancia de clase
	Relaciones
	Restricciones

3.2 Ventana Árbol de diagramas

La ventana *Árbol de diagramas* muestra todos los diagramas que contiene el proyecto de UModel activo.



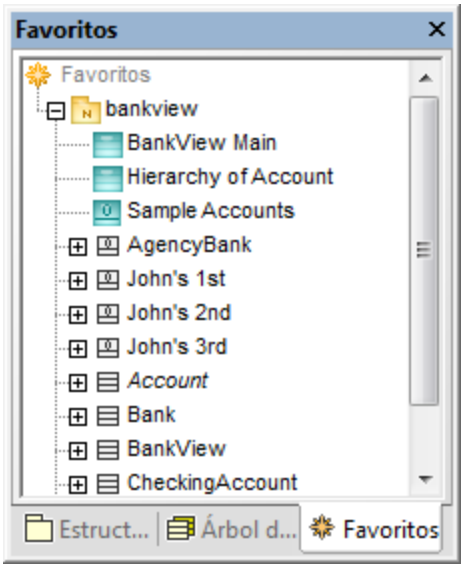
Panel Árbol de diagramas

Los diagramas de este panel se puede mostrar como una lista ordenada por orden alfabético o agrupados por tipo. Para cambiar las opciones de visualización haga clic con el botón derecho en la ventana y marque o desmarque la opción **Agrupar según el tipo de diagrama**.

Para instrucciones sobre crear, abrir y generar diagramas y sobre cómo modelar su contenido, consulte el capítulo Cómo modelar. Para obtener información específica sobre cada tipo de diagrama, consulte el capítulo Diagramas UML.

3.3 Ventana Favoritos

En la ventana *Favoritos* aparecen los elementos de modelado o diagramas que haya agregado a la lista de favoritos. La ventana *Favoritos* contiene una lista personal de elementos o diagramas seleccionados por usted que puede usar como accesos rápidos.



Panel Favoritos

El contenido de la ventana *Favoritos* se guarda automáticamente al guardar el proyecto. Esta opción se puede cambiar en la pestaña *Archivo* del cuadro de diálogo "Opciones" (**Herramientas | Opciones**) activando/desactivando la casilla *Cargar y guardar con el archivo de proyecto*.

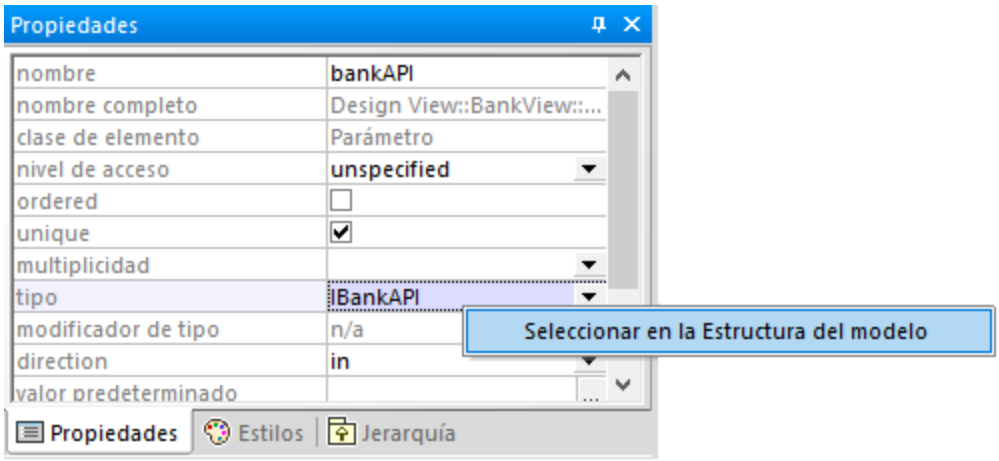
El elemento que aparece en la ventana *Favoritos* es una vista del elemento (es decir, no es ni una copia ni un clon del elemento). Las acciones que aplique en la ventana *Estructura del modelo* también se aplican en la ventana *Favoritos*, lo que incluye añadir o eliminar elementos. Para más información consulte el apartado Cómo modelar.

3.4 Ventana Propiedades

La ventana *Propiedades* muestra información sobre el elemento que esté seleccionado en ese momento, que puede ser un elemento seleccionado en la ventana *Estructura del modelo* (u otros paneles), un elemento seleccionado en el diagrama o incluso un diagrama.

La ventana *Propiedades* permite modificar las propiedades del elemento o de la relación seleccionados en ese momento. Las propiedades disponibles dependen del tipo de elemento que se haya seleccionado. Hay propiedades solo de lectura, que se muestran en gris (*"clase de elemento" en la imagen siguiente*), y propiedades que se pueden modificar (*"nombre" en la imagen siguiente*).

Para visualizar un atributo de una operación o propiedad en la ventana *Estructura del modelo* haga clic con el botón derecho en la entrada tipo en la ventana *Propiedades* y elija **Seleccionar en la *Estructura del modelo*** en el menú contextual. Puede hacer lo mismo con los parámetros de rendimiento.



Ventana Propiedades

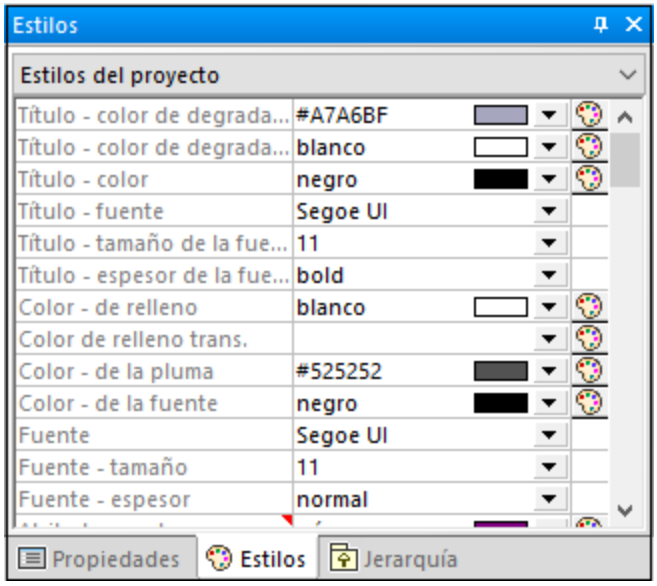
Cualquier cambio en una propiedad de un elemento de la ventana *Propiedades* se refleja inmediatamente en el diagrama. Asimismo, cualquier cambio que se efectúe en el diagrama (por ejemplo, cambiar la visibilidad de una operación de `pública` a `privada`) se verá reflejado en la propiedad correspondiente de la ventana *Propiedades*.

Las propiedades entre comillas angulares representan estereotipos (por ejemplo, `«final»`). Puede añadir estereotipos personalizados al proyecto, en cuyo caso aparecen como propiedades en la ventana *Propiedades*, junto a las propiedades predeterminadas. Para saber más consulte el apartado Ejemplo: crear y aplicar estereotipos.

3.5 Ventana Estilos

La ventana *Estilos* permite visualizar o cambiar el aspecto de los diagramas o elementos que estén seleccionados en ese momento. Los atributos de estilo se dividen en dos grandes grupos:

- ajustes de formato (por ejemplo, tamaño de fuente, peso de fuente, color, etc.) y
- opciones de visualización (por ejemplo, fondo negro, cuadrícula, visibilidad, etc.).



Panel Estilos

Cualquier cambio en una propiedad de la ventana *Estilos* se refleja inmediatamente en la interfaz del usuario. Asimismo, cualquier otro cambio de estilo que se efectúe (por ejemplo, cambiar la visibilidad del diagrama usando el botón **Mostrar cuadrícula** de la barra de herramientas) se verá reflejado en la propiedad correspondiente de la ventana *Estilos*.

En la parte superior de la ventana *Estilos* hay una lista desplegable que permite seleccionar a qué nivel se aplica el cambio de estilo (por ejemplo, a nivel de elemento individual o a nivel de proyecto). Para saber más, consulte:

- Cambiar el estilo de los elementos de un diagrama
- Cambiar el estilo de diagramas
- Cambiar el estilo de las líneas y relaciones

3.6 Ventana Jerarquía

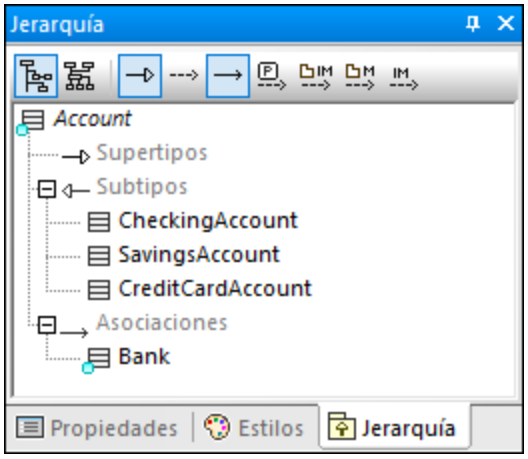
La ventana *Jerarquía* muestra todas las relaciones del elemento de modelado seleccionado en ese momento. El elemento de modelado se puede seleccionar en un diagrama, en la ventana *Estructura del modelo* o en la ventana *Favoritos*.

Los elementos de la ventana *Jerarquía* se pueden mostrar de dos maneras:

- vista en forma de árbol
- vista en forma de diagrama

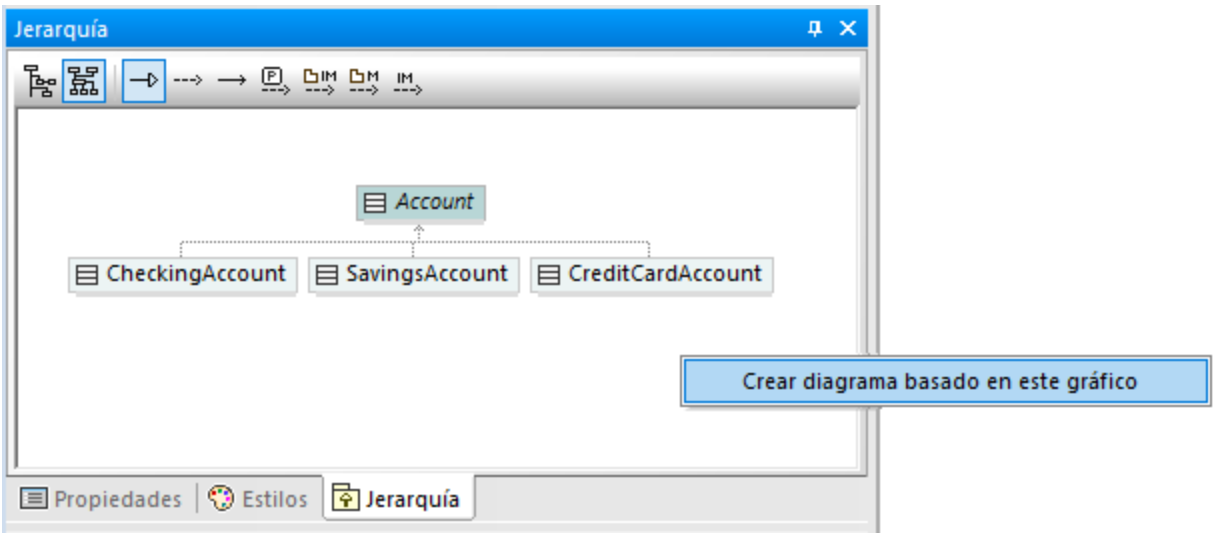
Para cambiar de una vista a otra haga clic en los botones **Mostrar en forma de árbol** o **Mostrar en forma de diagrama** en la esquina superior izquierda de la ventana.

La *vista en forma de árbol* muestra en forma de árbol todas las relaciones del elemento seleccionado en ese momento. Con los botones de la parte superior de la ventana puede seleccionar qué tipos de relaciones quiere que se muestren. En la imagen siguiente solo se han seleccionado las generalizaciones y las asociaciones.



Panel Jerarquía (vista en forma de árbol)

La *vista en forma de diagrama* muestra un solo conjunto de relaciones. En esta vista solo puede haber un icono activo en la barra de herramientas. En la imagen anterior, por ejemplo, está activo el icono **Mostrar generalizaciones**.



Panel Jerarquía (vista en forma de diagrama)

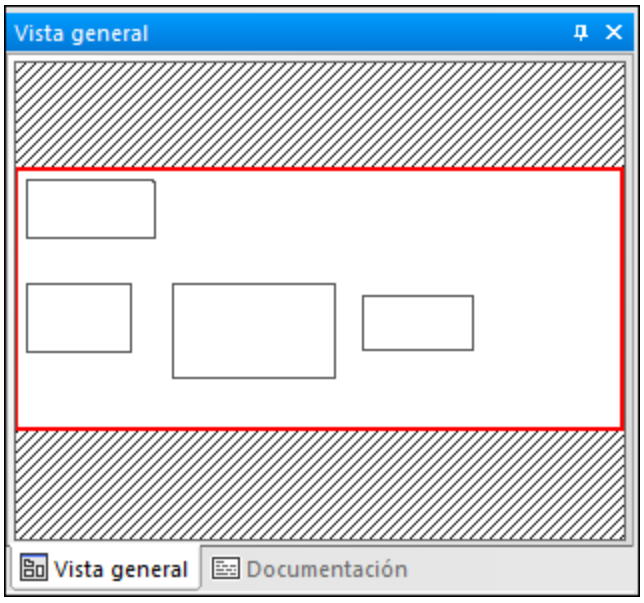
La *vista en forma de árbol* permite generar diagramas que incluyan los elementos visibles en la ventana. Para ello haga clic con el botón derecho dentro de la ventana y seleccione **Crear diagrama basado en este gráfico** en el menú contextual.

Los ajustes de la ventana *Jerarquía* se pueden cambiar usando la opción de menú **Herramientas | Opciones | Vista**, grupo **Jerarquía**, en la parte inferior del cuadro de diálogo.

La ventana *Jerarquía* es navegable: haga doble clic en el icono de uno de los elementos dentro de la ventana para mostrar las relaciones de dicho elemento. Esto se puede hacer tanto en la *vista en forma de árbol* como en la *vista en forma de diagrama*.

3.7 Ventana Vista general

La ventana *Vista general* ofrece una vista resumen del diagrama activo. Esto resulta útil para navegar por diagramas muy grandes. Haga clic en el rectángulo rojo y arrástrelo para desplazarse por la vista del diagrama.

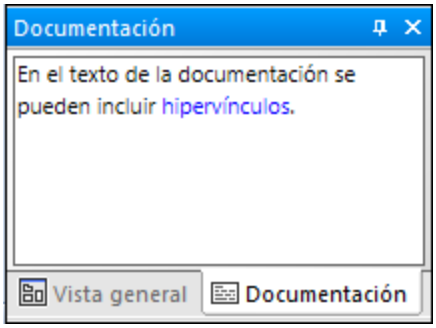


Panel Vista general

Consulte también Acercar y alejar diagramas.

3.8 Ventana Documentación

La ventana *Documentación* sirve para documentar cualquiera de los elementos UML que están disponibles en la ventana *Estructura del modelo*. Haga clic en el elemento que desea documentar y escriba sus comentarios en la ventana *Documentación*. Aquí puede usar las teclas de acceso rápido estándar para **seleccionar todo (Ctrl+A)**, **cortar (Ctrl+X)**, **copiar (Ctrl+C)** y **pegar (Ctrl+V)**.



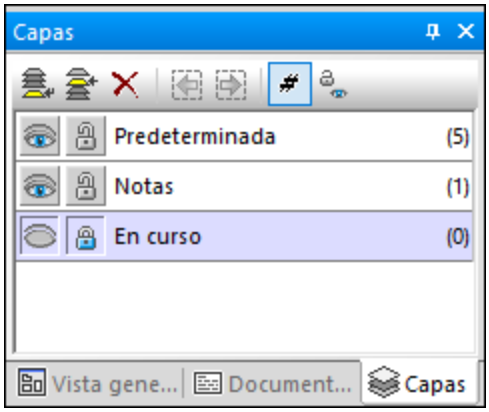
Panel documentación

Se puede pasar el corrector ortográfico al texto de la ventana *Documentación*. Haga clic con el botón derecho dentro de la ventana y seleccione **Ortografía de la documentación** en el menú contextual.

El texto de la documentación también se puede exportar como comentarios dentro del código fuente que se genera o importar desde los comentarios del código fuente con ingeniería inversa. Para saber más consulte Documentar elementos.

3.9 Ventana Capas

La ventana *Capas* sirve para crear distintas capas para cualquier diagrama de UModel. La principal función de las capas es agrupar de forma lógica los elementos de modelado de un diagrama. Por ejemplo, además de la capa predeterminada, también puede crear nuevas capas que contengan notas con información interna o clases inacabadas.

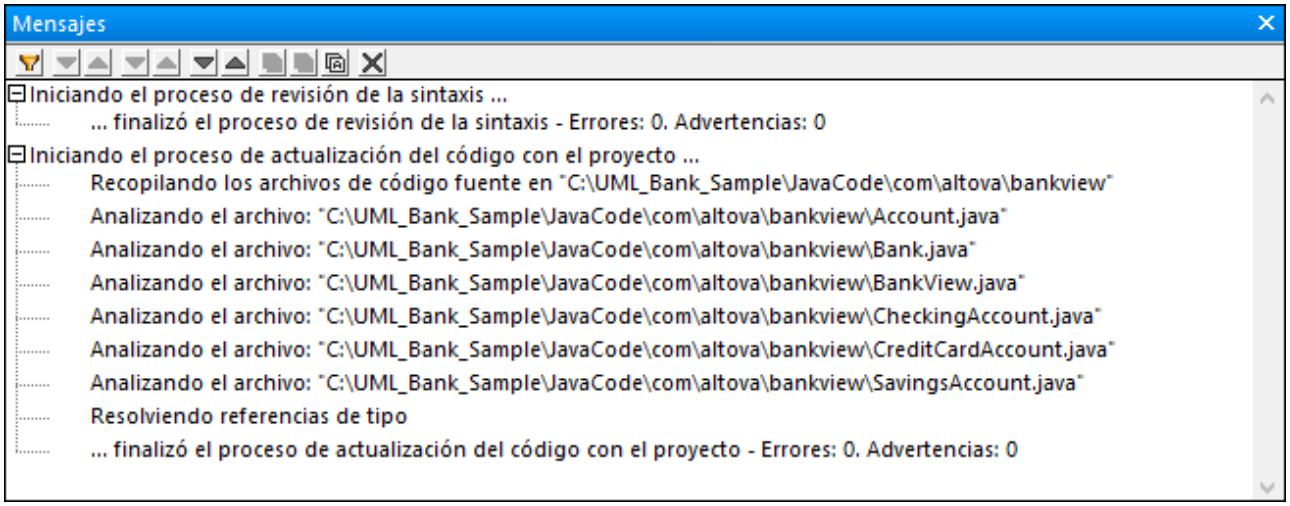


Panel Capas

Para más información consulte Agregar capas a los diagramas.

3.10 Ventana Mensajes

La ventana *Mensajes* muestra mensajes de advertencia, sugerencias y errores que pueden aparecer al revisar la sintaxis del proyecto (*véase* *Revisar la sintaxis del proyecto*) o al realizar tareas de ingeniería de código. Para obtener más información sobre la ingeniería de código, consulte los apartados Generar código de programa e Importar código fuente.



Ventana Mensajes

La siguiente lista muestra los tipos de mensajes que pueden aparecer y los iconos correspondientes.

Icono	Descripción
ningún icono	Mensaje de información.
	Mensaje de advertencia. Las advertencias son menos graves que los errores, pero pueden perjudicar a la importación o generación de código.
	Mensaje de error. Cuando ocurre un error fallan la generación o la importación de código.

Los botones de la parte superior de la ventana *Mensajes* permite realizar las siguientes acciones:

Icono	Descripción
	Filtra los mensajes según el nivel de gravedad: información y advertencias. Seleccione **Activar todos** para incluir todos los niveles de gravedad (opción predeterminada). Seleccione **Desactivar todos** para quitar todos los niveles de gravedad del filtro.
	Ir al siguiente error.
	Ir al error anterior.
	Ir a la siguiente advertencia.

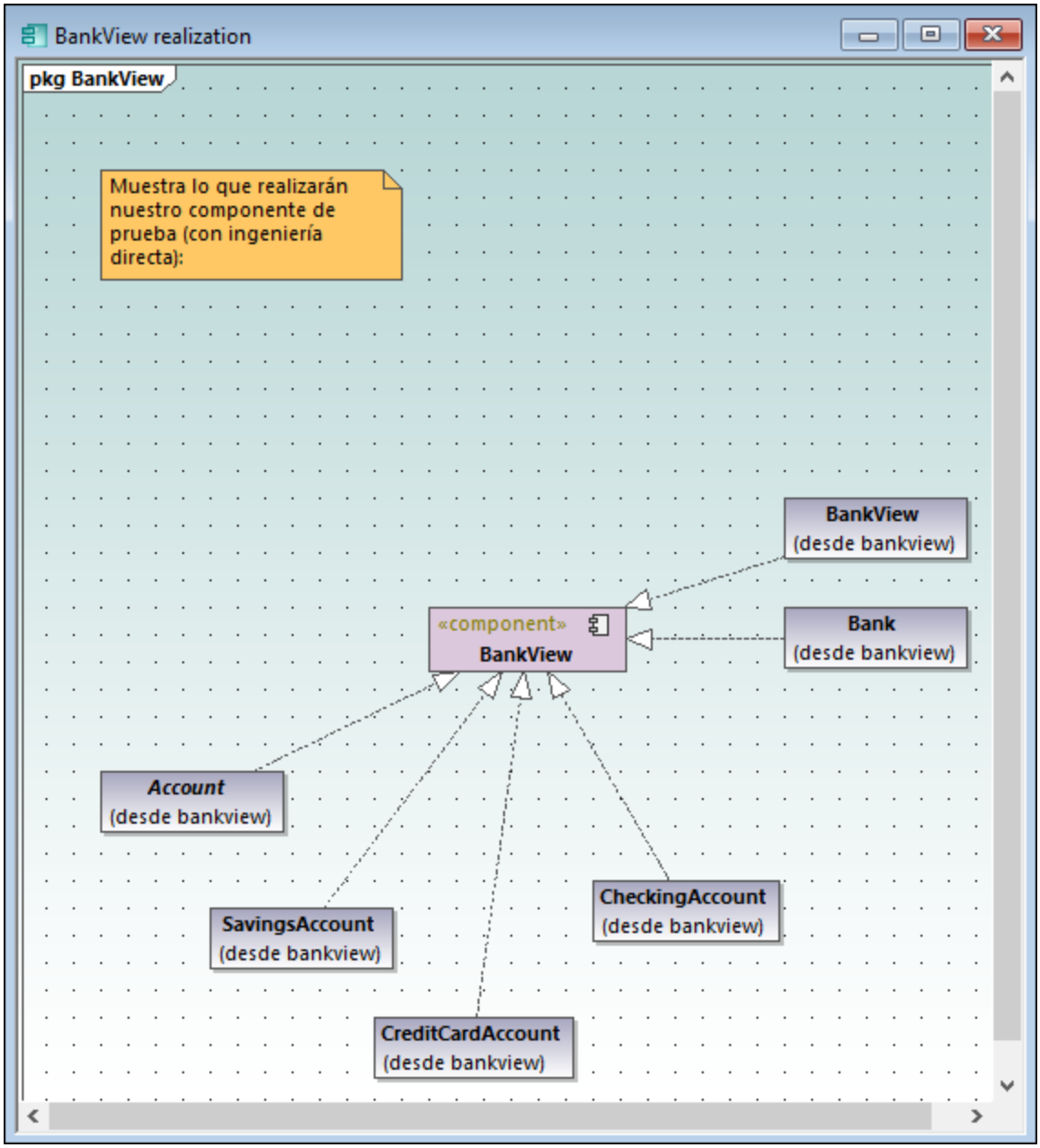
Icono	Descripción
	Ir a la advertencia anterior.
	Ir a la línea siguiente.
	Ir a la línea anterior.
	Copiar la línea seleccionada en el portapapeles.
	Copiar la línea seleccionada en el portapapeles, incluidas todas sus líneas anidadas.
	Copiar todo el contenido de la ventana Mensajes en el portapapeles.
	Borra el contenido de la ventana Mensajes.

Cuando UModel se ejecuta como un complemento de Visual Studio o Eclipse y ocurre un error de análisis, puede acceder al archivo de código fuente donde se originó el error directamente desde la ventana *Mensajes*. Para ello haga clic en la línea de la ventana *Mensajes* que contiene el error de análisis. Para más información, consulte los apartados Complemento de UModel para Visual Studio y Complemento de UModel para Eclipse.

3.11 Ventana de diagramas

Al crear un nuevo diagrama o al abrir uno que ya existe se carga una nueva ventana de diagramas en el panel *Diagramas*. La ventana de diagramas es la superficie en la que se pueden diseñar diagramas UML. Para acceder a los comandos de modelado disponibles haga clic con el botón derecho en la ventana de diagramas o en cualquier elemento que se encuentre en ella.

Es importante resaltar que los botones de la barra de herramientas y los comandos del menú contextual de UModel cambian en función del tipo de diagrama que esté activo en ese momento. Por ejemplo, si hace clic dentro de un diagrama de clases los botones de la barra de herramientas solo incluirán elementos aplicables a diagramas de clases. Para ver de qué tipo es el diagrama que está activo haga clic dentro de un área vacía del diagrama y observe la propiedad "clase de elemento" en el panel *Propiedades*. También puede averiguar el tipo de diagrama por el icono que lo acompaña (*véase Crear diagramas*).



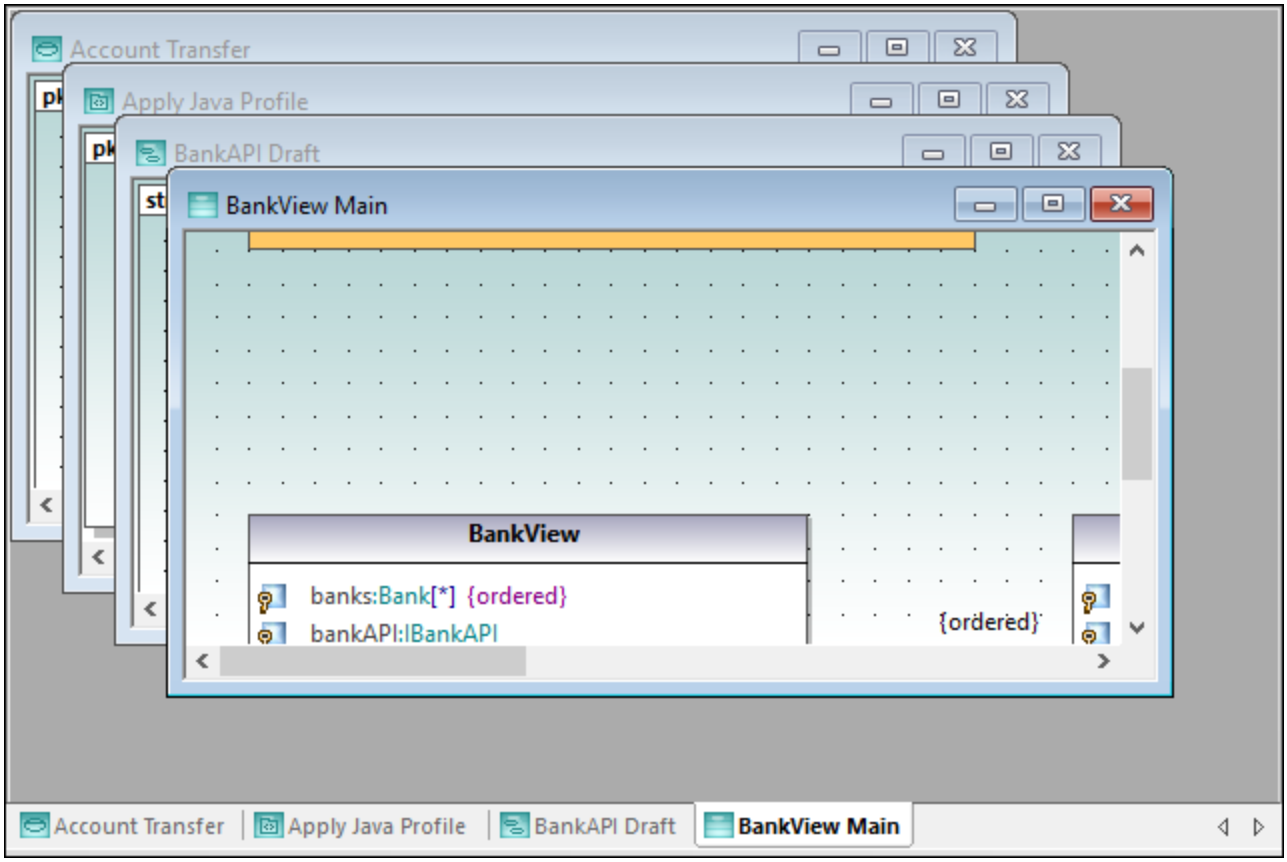
Panel Diagramas

Para más información sobre cómo crear nuevos diagramas, abrir diagramas que ya existen o manipular elementos dentro del diagrama, consulte el apartado Cómo modelar.

3.12 Panel Diagramas

En el panel *Diagramas* se encuentran todas las ventanas de diagramas que estén abiertas. Para más información sobre cómo crear nuevos diagramas, abrir diagramas que ya existen o manipular elementos dentro del diagrama, consulte el apartado How to Model....

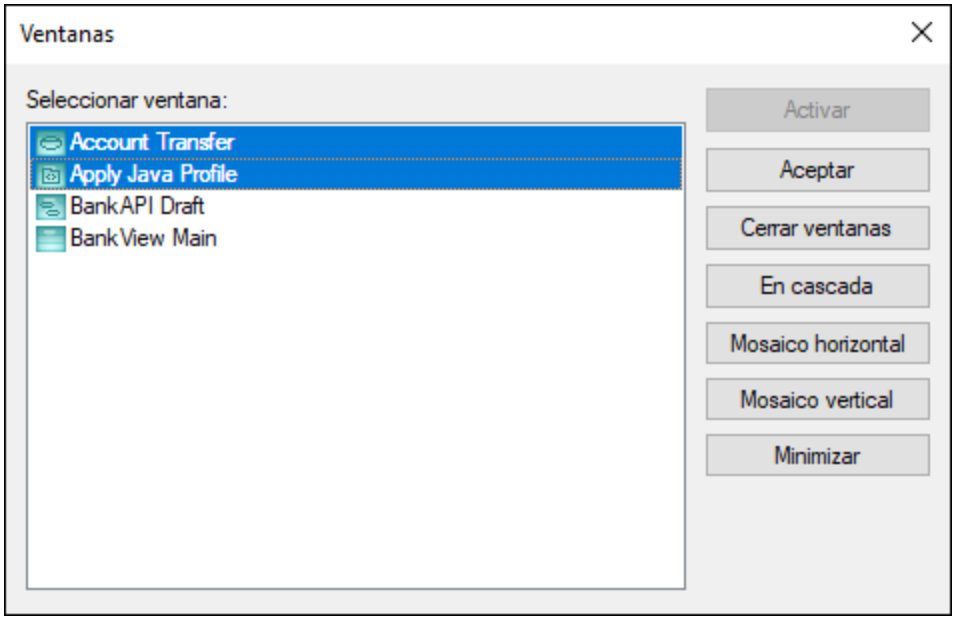
La imagen siguiente muestra el panel *Diagramas* con cuatro ventanas de diagramas abiertas, posicionadas según el comando de menú **Ventanas | En cascada**.



Panel Diagramas

Para acceder a los comandos aplicables a la ventana de diagrama activa en ese momento, haga clic con el botón derecho en la pestaña correspondiente en la parte inferior del panel *Diagramas*.

Para aplicar distintos comandos a las ventanas que estén dentro del panel *Diagramas*, use los comandos disponibles en el menú **Ventanas**. Hay más comandos de este tipo disponibles en el cuadro de diálogo **Ventanas...**, al que puede acceder seleccionando el comando de menú **Ventanas | Ventanas...**



Cuadro de diálogo Ventanas

Para seleccionar varias ventanas en el cuadro de diálogo de la imagen anterior, mantenga pulsada la tecla **Ctrl** y haga clic en las entradas correspondientes.

Para pasar de una ventana de diagramas a otra, pulse **Ctrl+Tabulador**.

4 Interfaz de la línea de comandos

Además de una interfaz gráfica del usuario, UModel ofrece una interfaz de la línea de comandos. Para abrirla debe ejecutar el archivo `UModelBatch.exe`, que está en el directorio `C:\Archivos de programa\Altova\UModel2023`. Si trabaja con la versión de 32 bits de UModel en un sistema operativo de 64 bits, entonces la ruta de acceso del archivo es `UModelBatch.exe` es `C:\Archivos de programa (x86)\Altova\UModel2023`.

En este apartado puede consultar la sintaxis de parámetros de la línea de comandos. Para ver la sintaxis en una ventana del símbolo del sistema, escriba **umodelbatch /?** en la línea de comandos.

Nota: si la ruta de acceso o el nombre de archivo contienen espacios, entonces deben ir entre comillas (p. ej. "C:\Archivos de programa\...\MiProyecto.ump").

```
utilización: UModelBatch.exe [proyecto] [opciones]

/? o /help ... mostrar esta información de ayuda

proyecto          ... archivo de proyecto (*.ump)
/new[=archivo]    ... crear, guardar o guardar como proyecto nuevo (véase Crear, cargar
y guardar proyectos por lotes)
/set              ... establecer operaciones permanentes
/gui              ... mostrar la interfaz de usuario de UModel


comandos (ejecutados en este orden):
/chk                    ... revisar la sintaxis del proyecto
/isd=ruta               ... importar directorio de código fuente
/isp=archivo      ... importar archivo de proyecto de código fuente
                 (*.project,*.xml,*.jpx,*.csproj,*.csdproj,*.vcxproj,*.vbproj,*.vbdproj
,*.sln,*.bdsproj)
/ibt=lista              ... importar tipos binarios (especificar lista de [nombres de
tipos] binarios)
                 (";"=separador, "*"=todos los tipos, "#" antes de los nombres de
ensamblado)
/ixd=ruta               ... importar directorio del esquema XML
/ixs=archivo      ... importar archivo de esquema XML (*.xsd)
/m2c                    ... actualizar código de programa con el modelo
(exportar/ingeniería directa)
/c2m                    ... actualizar modelo con el código de programa
(importar/ingeniería inversa)
/ixf=archivo      ... importar archivo XMI
/exf=archivo      ... exportar a archivo XMI
/inc=archivo      ... incluir archivo
/mrg=archivo      ... combinar archivo
/doc=archivo      ... escribir documentación en el archivo especificado
/lue[=cpri]             ... mostrar una lista de los elementos que no se utilizan en
ningún diagrama (es decir, no utilizados)
/ldg                    ... mostrar una lista de todos los diagramas
/lcl                    ... mostrar una lista de todas las clases
/lsp                    ... mostrar una lista de todos los paquetes compartidos
/lip                    ... mostrar una lista de todos los paquetes incluidos
```

```
opciones para guardar como proyecto nuevo:
/npad=opc   ... ajustar rutas de acceso relativas (Yes | No | MakeAbsolute), es decir
Sí, No o Convertir en absolutas


opciones para comandos de importación:
/iclg=leng  ... lenguaje de código (Java1.4 | Java5.0 | Java6.0 | Java7.0 | Java8.0 |
Java9.0 | Java10.0 | Java11.0 | Java12.0 | Java13.0 | Java14.0 |
                              C#1.2 | C#2.0 | C#3.0 | C#4.0 | C#5.0 | C#6.0 | C#7.0 |
C#7.1 | C#7.2 | C#7.3 | C#8.0 |
                              VB7.1 | VB8.0 | VB9.0 |
                              C++98 | C++11 | C++14 | C++17)
/ipsd[=0|1] ... procesar subdirectorios (recursivo)
/irpf[=0|1] ... importar relativos al archivo de proyecto de UModel
/ijdc[=0|1] ... JavaDocs como comentarios de Java
/icdc[=0|1] ... DocComments como comentarios de C#
/icds[=lst] ... Símbolos definidos de C#
/ivdc[=0|1] ... DocComments como comentarios de VB
/ivds[=lst] ... Símbolos definidos de VB (constantes personalizadas)
/icppdm[=lst] ... macros definidas con C++
/icpphi[=0|1] ... leer solo los archivos de encabezados C++
/icpphc[=0|1] ... tratar archivos .h como archivos .cpp
/icppms[=0|1] ... habilitar la compatibilidad con el compilador C++ de Microsoft
/icppmv[=ver] ... versión de MSVC que usar (1900 | 1800 | 1700 | 1600 | 1500 | 1400 |
1310 | 1300 | 1200)
/icppsy[=0|1] ... detectar automáticamente los archivos C++ system include
/icppid[=lst] ... lista de qué directorios C++ include usar
/icppsd[=lst] ... lista de qué directorios C++ system include usar
/icppag[=arg] ... Otros argumentos C++ para el compilador
/imrg[=0|1] ... sincronizar combinados
/iudf[=0|1] ... utilizar filtro de directorios
/iflt[=lst] ... filtro de directorios (restablece /iudf)


opciones para importar tipos binarios (después de /iclg):
/ibrt=vers  ... versión en tiempo de ejecución
/ibpv=path  ... reemplazo de la variable PATH para buscar bibliotecas de código nativo
/ibro[=0|1] ... usar el contexto de sólo reflexión
/ibua[=0|1] ... usar la opción de agregar tipos referenciados con filtro de paquetes
/ibar[=flt] ... agregar el filtro de paquetes de tipos referenciados
(restablece /ibua)
/ibot[=0|1] ... sólo importar los tipos
/ibuv[=0|1] ... utilizar el filtro de nivel de acceso mínimo
/ibmv[=key] ... palabra clave para el nivel de acceso mínimo necesario
(restablece /ibuv)
/ibsa[=0|1] ... omitir secciones de atributo o modificadores de anotación
/iboa[=0|1] ... crear un solo atributo por sección de atributos
/ibss[=0|1] ... omitir el sufijo "Attribute" en los nombres de tipo de atributo


opciones para la generación de diagramas:
/dgen[=0|1] ... generar diagramas
/dopn[=0|1] ... abrir diagramas generados
/dsac[=0|1] ... mostrar compartimento de atributos
```

```
/dsoc[=0|1] ... mostrar compartimento de operaciones
/dscc[=0|1] ... mostrar compartimento de clasificadores anidados
/dstv[=0|1] ... mostrar valores etiquetados
/dudp[=0|1] ... usar compartimento de la propiedad .NET
/dspd[=0|1] ... mostrar compartimento de la propiedad .NET


opciones par comandos de exportación:
/ejdc[=0|1] ... comentarios de Java como JavaDocs
/ecdc[=0|1] ... comentarios de C# como DocComments
/evdc[=0|1] ... comentarios de VB como DocComments
/espl[=0|1] ... utilizar plantillas SPL definidas por el usuario
/ecod[=0|1] ... convertir código eliminado en comentario
/emrg[=0|1] ... sincronizar combinados
/egfn[=0|1] ... generar los nombres de archivo que falten
/eusc[=0|1] ... usar revisión de sintaxis


opciones para la exportación de XMI:
/exid[=0|1] ... exportar identificadores UUID
/exex[=0|1] ... exportar extensiones específicas de UModel
/exdg[=0|1] ... exportar diagramas (restablece /exex)
/exuv[=ver] ... versión de UML (UML2.0 | UML2.1.2 | UML2.2 | UML2.3 | UML2.4 | UML2.5
| UML2.5.1)


opciones para combinar archivos:
/mcan=archivo  ... archivo antecesor común


opciones para la generación de documentación:
/doof=fmt   ... formato de salida (HTML | RTF | MSWORD | PDF)
/dsps=archivo  ... archivo de diseño SPS
```

Ejemplo nº1: importar código fuente Java y conservar la configuración

Este comando importa código fuente y crea un archivo de proyecto nuevo. Observe que la ruta de acceso del proyecto contiene espacios, por lo que está entre comillas.

```
"C:\Archivos de programa\Altova\UModel2023\UModelBatch.exe" /new="C:\Mis
Proyectos\Fred.ump" /isd="X:TestCases\UModel\Fred" /set /gui /iclg=Java8.0 /ipsd=1 /ijdc=
1 /dgen=1 /dopn=1 /dmax=5 /chk
```

A continuación indicamos el significado de cada opción:

`/new`	Indica que el archivo de proyecto nuevo debe llamarse "Fred.ump" y se debe guardar en `C:\Archivos de programa\Altova\UModel2023\UModelBatchOut\`.
`/isd`	Indica que el directorio fuente debe ser `X:\TestCases\UModel\Fred`.
`/set`	Indica que las opciones utilizadas en la línea de comandos se guardarán en el registro (y estas opciones formarán la configuración predeterminada la próxima vez que se abra UModel).

/gui	Muestra la interfaz gráfica de UModel durante el procesamiento por lotes.
/iclg	UModel importará el código como Java 8.0.
/ipsd=1	Procesa recursivamente todos los subdirectorios del directorio raíz suministrado con el parámetro **/isd**.
/ijdc=1	Crea JavaDocs donde corresponda a partir de los comentarios.
/dgen=1	Genera diagramas.
/dopn=1	Abre los diagramas generados.
/chk	Revisa la sintaxis.

Ejemplo nº2: sincronizar código fuente desde el modelo

Este comando actualiza código desde un archivo de proyecto que ya existe ("C:\UModel\Fred.ump").

```
"C:\Archivos de programa\Altova\UModel2023\UModelBatch.exe" "C:
\UModel\Fred.ump" /m2c /ejdc=1 /ecod=1 /emrg=1 /egfn=1 /eusc=1
```

En este ejemplo existen las mismas opciones que en el anterior, más:

/m2c	Actualiza el código con el modelo.
/ejdc	Genera JavaDoc a partir de los comentarios del modelo del proyecto.
/ecod=1	Convierte el código eliminado en comentarios.
/emrg=1	Sincroniza el código combinado.
/egfn=1	Genera en el proyecto los nombres de archivo que falten.
/eusc=1	Usa la revisión de sintaxis.

Ejemplo nº 3: importar archivos binarios Java en el modelo

Imaginemos que en el directorio **C:\JavaProject\bin** hay unos archivos binarios Java que quiere importar a UModel. Para ello, ejecute el siguiente comando:

```
"<C:\Program Files\Altova\UModel2023\UModelBatch.exe>" /new="C:
\JavaProject\Result.ump" /ibt=*C:
\JavaProject\bin /iclg=Java8.0 /ibrt=JDK1.8.0_144 /dgen=1 /chk
```

A continuación indicamos las opciones que se pueden usar:

/new	Crea un nuevo proyecto de UModel en la ruta indicada.
/ibt	Indica a UModel que importe archivos binarios. El asterisco antes de la ruta indica que se deben importar todos los tipos binarios de esa ruta.
/iclg	Indica el lenguaje en que se debe generar el código ("Java 8.0" en este ejemplo).

`/ibrt`	Indica el entorno en el momento de ejecución ("JDK1.8.0_144" en este ejemplo). Este es el mismo ejemplo que aparece en el cuadro de diálogo "Importar tipos binarios" (véase Importar archivos binarios Java, C# y VB). También puede usar un valor como "jdk-10.0.1" como en la variable de entorno `JAVA_HOME`.
	
	En el caso de C# puede usar el valor `/ibrt:any` o los valores que van apareciendo en la IGU en tiempo de ejecución. No olvide omitir los espacios. Ejemplos:
	
	`/ibrt:any`
	`/ibrt:.NET5`
	`/ibrt:.NETFramework4.8(v4.8.3752)`
	
	La opción "any" es la misma que "any (use disassembler)" de la lista desplegable "Tiempo de ejecución" y es la opción recomendada.

`/dgen=1`	Genera diagramas.
`/chk`	Comprueba la sintaxis después de importar los archivos binarios.

4.1 Crear, cargar y guardar proyectos por lotes

Al ejecutar **UModelBatch.exe** con un comando como `UModelBatch MyProject.ump` puede usar estos parámetros:

`/new`	Este parámetro define la ruta y el nombre del archivo de proyecto de UModel nuevo que se crea. También se puede usar para cargar un proyecto y guardarlo con un nombre distinto, por ejemplo: `UmodelBatch.exe MyFile.ump /new=MyBackupFile.ump`
`/set`	Este parámetro sobrescribe la configuración definida en el registro con las opciones que defina.
`/gui`	Este parámetro muestra la interfaz gráfica del usuario de UModel durante el procesamiento por lotes.

A continuación mostramos cómo crear, cargar o guardar proyectos en modo por lotes integral (es decir, sin usar el parámetro `/gui`).

new

`UModelBatch /new=xxx.ump (opciones)`
crea un proyecto nuevo, ejecuta las opciones y xxx.ump se guarda **siempre** (independientemente de las opciones utilizadas)

auto save

`UModelBatch xxx.ump (opciones)`
carga el proyecto xxx.ump, ejecuta las opciones y xxx.ump **solo** se guarda si hubo cambios en el documento (como **`/ibt`**)

save

`UModelBatch xxx.ump (opciones) /new`
carga el proyecto xxx.ump, ejecuta las opciones y xxx.ump se guarda **siempre** (independientemente de las opciones utilizadas)

save as

`UModelBatch xxx.ump (opciones) /new=yyy.ump`
carga el proyecto xxx.ump, ejecuta las opciones y xxx.ump se guarda siempre como `yyy.ump` (independientemente de las opciones utilizadas)

A continuación mostramos cómo crear, cargar o guardar proyectos en **modo por lotes** con la interfaz de UModel (es decir, usando el parámetro **`/gui`**).

new

`UModelBatch /gui /new (opciones)`
crea un proyecto nuevo, ejecuta las opciones y no se guarda nada; la interfaz gráfica permanece abierta

save new

`UModelBatch /gui /new=xxx.ump (opciones)`

crea un proyecto nuevo, ejecuta las opciones y se guarda el archivo xxx.ump; la interfaz gráfica permanece abierta

user mode

UModelBatch /gui xxx.ump (opciones)
carga el proyecto xxx.ump, ejecuta las opciones y no se guarda nada; la interfaz gráfica permanece abierta

save

UModelBatch /gui xxx.ump (opciones) /new
carga el proyecto xxx.ump, ejecuta las opciones y se guarda el archivo xxx.ump; la interfaz gráfica permanece abierta

save as

UModelBatch /gui xxx.ump (opciones) /new=yyy.ump
carga el proyecto xxx.ump, ejecuta las opciones y el archivo xxx.ump se guarda como yyy.ump; la interfaz gráfica permanece abierta

El proyecto se guardará correctamente siempre y cuando no se produzcan errores graves durante la ejecución de las opciones.

5 Cómo modelar

Sitio web de Altova: Modelado UML

Esta sección es una guía para aprender a modelar con UModel. En ella aprenderemos a crear y manipular elementos, diagramas y relaciones UML desde la interfaz gráfica del usuario de UModel. Las instrucciones son generales para UModel, no específicas para ningún elemento o tipo de diagrama en particular, a no ser que se especifique lo contrario. La información sobre cada tipo de diagrama se encuentra en la sección Diagramas UML.

Esta sección está organizada en las siguientes categorías: elementos, diagramas, relaciones y estereotipos.

Elementos	Diagramas	Relaciones	Estereotipos
Crear elementos	Crear diagramas	Crear relaciones entre elementos	Estereotipos y valores etiquetados
Insertar elementos del modelo en un diagrama	Generar diagramas	Cambiar el estilo de las líneas y relaciones	Valores etiquetados
Renombrar, mover y copiar elementos	Abrir diagramas	Ver las relaciones de los elementos	Aplicar estereotipos
Borrar elementos	Borrar diagramas	Asociaciones	Mostrar u ocultar valores etiquetados
Converting Elements	Cambiar el estilo de los diagramas	Asociación de colecciones	
Buscar y reemplazar texto	Alinear y ajustar el tamaño de elementos de modelado	Contención	
Comprobar si se están usando ciertos elementos y dónde	Finalización automática en clases		
Restricción de elementos	Acercar y alejar diagramas		
Agregar hipervínculos a elementos	Agregar capas a los diagramas		
Documentar elementos			
Cambiar el estilo de los elementos de un diagrama			

Nota: UModel incluye varios proyectos que puede usar para familiarizarse con los conceptos básicos de modelado y con la interfaz gráfica del usuario. Los encontrará aquí: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**.

5.1 Elementos

5.1.1 Crear elementos

En UModel puede crear nuevos elementos como sigue:

- Desde la ventana *Estructura del modelo* los elementos se añaden únicamente al modelo y usted puede insertarlos más tarde en diagramas.
- Desde cualquiera de las ventanas de diagrama. Cualquier elemento que añada a un diagrama se añadirá también automáticamente al modelo. Si necesita eliminar algún elemento posteriormente, puede escoger entre eliminarlo solo del diagrama o eliminarlo también del modelo.

Para añadir elementos desde la ventana *Estructura del modelo*:

- En la ventana *Estructura del modelo* (o en la ventana *Favoritos*) haga clic con el botón derecho en el elemento bajo el cual quiere que aparezca el nuevo elemento (por ejemplo, Root) y seleccione **Elemento nuevo | <Nombre del elemento>** del menú contextual. Por ejemplo, para añadir un nuevo paquete bajo el paquete "Root", seleccione **Elemento nuevo | Paquete**.

Para añadir elementos desde la ventana de diagramas:

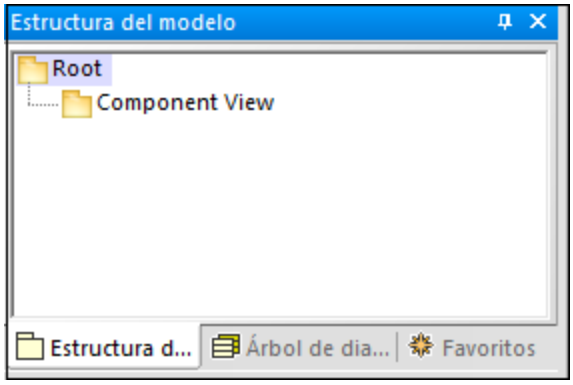
1. Cree un nuevo diagrama (*véase Crear diagramas*) o abra uno que ya existe (*véase Abrir diagramas*).
2. Ahora puede:
   a. Hacer clic con el botón derecho dentro del diagrama y seleccionar **Nuevo/a | <Nombre del elemento>** en el menú contextual.
   b. Haga clic en el botón de la barra de herramientas que quiera añadir dentro del diagrama. Para insertar varios elementos del mismo tipo, mantenga pulsada la tecla Ctrl antes de hacer clic dentro del diagrama.

Paquetes

Al modelar elementos, es probable que trabaje con paquetes más a menudo que con ningún otro elemento. Cada entrada marcada con el icono de carpeta en la Estructura del modelo representa un paquete UML. En UModel los paquetes sirven como contenedores para todos los demás elementos de modelado UML (incluidos diagramas, clases, etc.) y se comportan de la siguiente manera:

- Se pueden crear en cualquier posición de la Estructura del modelo.
- Se pueden mover o copiar a otros paquetes, además de a diagramas de modelo válidos (*véase Renombrar, copiar y mover elementos*).
- Se pueden usar como elementos origen o destino al generar código o sincronizarlo con el modelo. Consulte Ingeniería directa (del modelo al código) y Ingeniería inversa (del código al modelo).

Al crear un proyecto en UModel hay dos paquetes predeterminados disponibles: "Root" y "Component View". Estos dos paquetes son los únicos que no se pueden renombrar ni eliminar. "Root" sirve de punto de partida para modelar el resto de elementos y "Component View" es necesario para la ingeniería de código.



Paquetes predeterminados de UModel

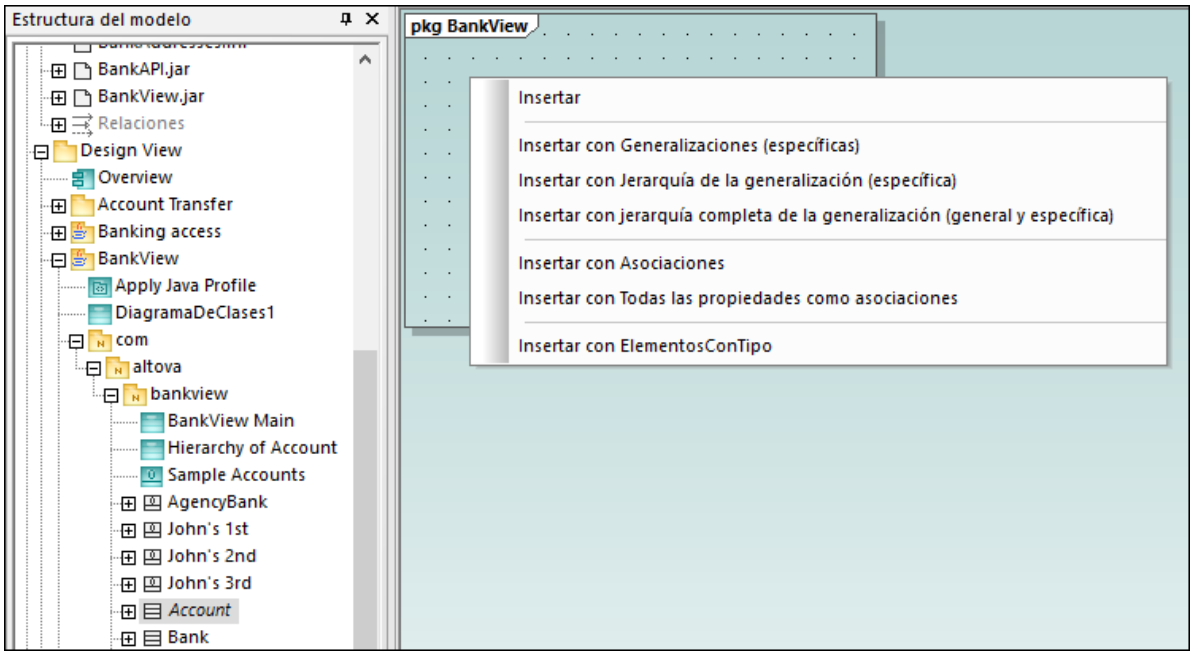
5.1.2 Insertar elementos del modelo en un diagrama

Los elementos del modelo se pueden insertar en un diagrama tanto individualmente como en grupo. Para seleccionar varios elementos de la ventana *Estructura del modelo*, mantenga pulsada la tecla **Ctrl** y vaya haciendo clic sobre los elementos que quiere seleccionar. Hay dos maneras de insertar elementos en un diagrama: arrastrar con botón izquierdo y arrastrar con botón derecho.

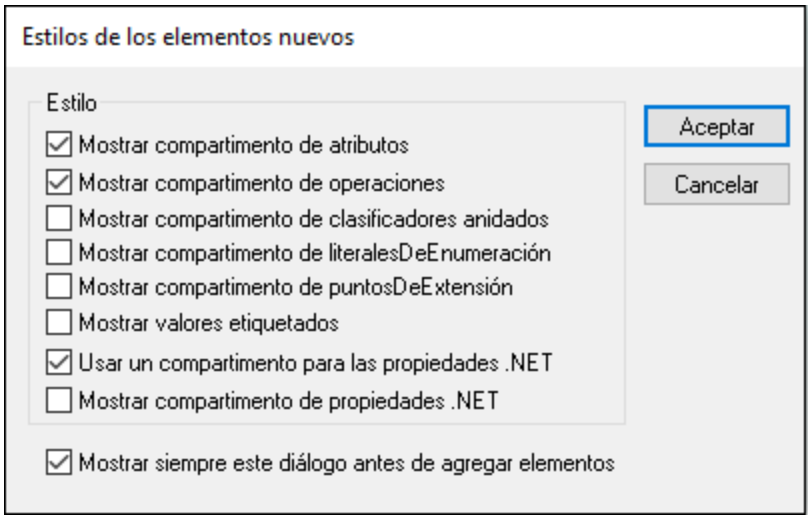
- **Arrastrar con botón izquierdo** (mantenga pulsado el botón izquierdo del ratón mientras arrastra el elemento y suéltelo cuando esté dentro del diagrama) inserta el elemento inmediatamente donde se encuentre el cursor. En este caso se muestra automáticamente cualquier asociación, dependencia, etc. que exista entre los elementos ya insertados y el nuevo.
- **Arrastrar con botón derecho** (mantenga pulsado el botón derecho del ratón mientras arrastra el elemento y suéltelo cuando esté dentro del diagrama) abre un menú contextual del cual p0uede seleccionar las asociaciones y generalizaciones específicas que quiera mostrar.

Por ejemplo, supongamos que quiere crear un nuevo diagrama de clases a partir de una clase que ya existe en el modelo. Para ilustrar este escenario, abra el proyecto de ejemplo **Bank_MultiLanguage.ump**, que encontrará en **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**. Asumiendo que quiera replicar el diagrama "Account Hierarchy" en un nuevo diagrama de clases, haga lo siguiente:

1. Haga clic con el botón derecho en el paquete **bankview** y seleccione **Diagrama nuevo | Diagrama de clases**.
2. Localice la clase abstracta `Account` en la *Estructura del modelo* y **arrastre con el botón derecho** para situarlo en el nuevo diagrama. En este ejemplo queremos mostrar la clase junto con sus clases derivadas. Para ello, seleccione **Insertar con Jerarquía de la generalización (específica)** en el menú contextual.



3. Marque las casillas de los elementos que quiere que aparezcan en el diagrama.



4. Haga clic en **Aceptar**. La clase `Account`, junto con sus tres subclases, se inserta en el diagrama. Las flechas de generalización se muestran automáticamente. Para organizar automáticamente las clases dentro del diagrama, ejecute el comando de menú **Diseño | Aplicar diseño automático a todo | Diseño jerárquico**.

Si hubiera seleccionado el comando **Insertar** en vez de **Insertar con Jerarquía de la generalización (específica)**, se había añadido la clase al diagrama sin ninguna clase derivada. No obstante, sigue pudiendo mostrar la jerarquía de la generalización más adelante de la siguiente manera:

- Haga clic con el botón derecho en la clase Account en el diagrama y seleccione **Mostrar | Jerarquía de la generalización (específica)**, lo que hará que se inserten también las clases derivadas en el diagrama.

5.1.3 Renombrar, mover y copiar elementos

Puede cortar, copiar, renombrar y mover elementos en la ventana *Estructura del modelo* y dentro de diagramas del mismo tipo. En algunos casos, estas acciones también son posibles entre diagramas de distintos tipos. También puede copiar o mover elementos desde la Estructura del modelo a un diagrama, siempre que las especificaciones UML permitan que dicho diagrama contenga el elemento correspondiente.

Para renombrar un elemento:

- Haga doble clic en el nombre del elemento y edítelo.
- También puede hacer clic en el elemento y pulsar la tecla **F2**.

Este procedimiento sirve independientemente de en qué panel se muestre el elemento, incluidos los paneles Estructura del modelo, Propiedades y Diagramas.

> Los paquetes "Root" y "Component View" se muestran siempre en la ventana *Estructura del modelo* y no se pueden renombrar ni eliminar.

Para copiar o mover elementos:

- Use los comandos estándar de Windows **Cortar**, **Copiar** o **Pegar**. Para acceder a ellos puede usar las teclas de acceso rápido **Ctrl+X (cortar)**, **Ctrl+C (copiar)** y **Ctrl+V (pegar)**, los botones correspondientes de la barra de herramientas o el menú **Editar**.
- También puede arrastrar un elemento hasta un paquete (u otro elemento). Al arrastrar un elemento lo mueve. Al mantener pulsada la tecla **Ctrl** y arrastrar un elemento, se crea una copia del mismo.

Por ejemplo, en un diagrama puede mover un miembro de una clase a otra clase arrastrándolo desde la clase de origen hasta la clase de destino. Para copiar el miembro de la clase en lugar de moverlo, selecciónelo y luego arrástrelo a su clase de destino mientras mantiene pulsada la tecla **Ctrl**.

Si pega una clase en el mismo paquete, la nueva clase se crea con un número secuencial anexado al final, por ejemplo "MiClase1". Asimismo, si pega una propiedad en de la misma clase, la nueva propiedad se crea con un número secuencias anexado al final, por ejemplo "MiPropiedad1". Lo mismo ocurre para otros miembros de la clase, como operaciones o enumeraciones. La misma lógica se aplica también al pegar elementos en del mismo diagrama, siempre que el diagrama pertenezca al mismo paquete que los elementos que se pegan.

Si pega una clase en un paquete distinto, la nueva clase tendrá el mismo nombre que la clase original. La misma lógica se aplica si copia miembros de la clase (como propiedades, operaciones, etc.) en clases distintas.

Por defecto, cualquier elemento que se pegue en un diagrama se añade también automáticamente al modelo (por lo que es visible en la ventana *Estructura del modelo*). Sin embargo, también puede copiar y pegar un elemento únicamente en el diagrama actual, sin añadirlo al modelo. Para ello, copie el elemento, haga clic con el botón derecho en el diagrama y seleccione **Pegar solo en el diagrama** en el menú contextual. El comando **Pegar solo en el diagrama** también aparece cuando arrastra un elemento que ya existe al mismo diagrama mientras mantiene pulsada la tecla **Ctrl**.

En el ejemplo anterior, el comando **Pegar** creará una nueva clase en el diagrama y la añadirá también al modelo, mientras que **Pegar solo en el diagrama** únicamente mostrará una segunda vista del mismo en el diagrama. Tenga en cuenta que las copias creadas usando este último método son solamente vistas adicionales del elemento original, al que redirigen, pero no se trata de copias independientes. (Por ejemplo, renombrar una propiedad en el duplicado de una clase aplicará automáticamente ese mismo cambio a la clase original)

5.1.4 Borrar elementos

Se pueden borrar elementos de dos maneras:

- Desde la ventana *Estructura del modelo*. Use este método si quiere borrar de todos los diagramas en los que aparezca y también del proyecto.
- Directamente en los diagramas en los que aparecen. En este caso puede escoger si borrar el elemento solo del diagrama o del modelo (proyecto) también.

Para borrar elementos del proyecto y de todos los diagramas asociados (método 1):

1. En la ventana *Estructura del modelo* haga clic en el elemento que quiere borrar. Mantenga pulsada la tecla **Ctrl** para seleccionar varios elementos.
2. Pulse **Eliminar**.

Para borrar elementos del proyecto y de todos los diagramas asociados (método 2):

1. Abra un diagrama y haga clic en el elemento que quiere borrar. Mantenga pulsada la tecla **Ctrl** para seleccionar varios elementos.
2. Pulse **Eliminar**. Aparecerá un cuadro de diálogo que preguntará si confirma que quiere borrar el elemento del proyecto y del diagrama.
3. Haga clic en **Sí**. El elemento se borra tanto del diagrama como del proyecto.

Para borrar elementos del diagrama pero no del proyecto:

1. Abra un diagrama y haga clic en los elementos que quiere borrar. Mantenga pulsada la tecla **Ctrl** para seleccionar varios elementos.
2. Mantenga pulsada la tecla **Ctrl** y pulse la tecla **Suprimir**. Los elementos se borran del diagrama pero siguen existiendo en el proyecto.

Antes de borrar elementos de un proyecto, recomendamos que compruebe que no se están usando en ningún diagrama.

- Haga clic con el botón derecho en la Estructura del modelo y seleccione **Mostrar el elemento en todos los diagramas** en el menú contextual.

Asimismo, cuando un diagrama está abierto, puede seleccionar rápidamente un elemento en la Estructura del modelo así:

- Haga clic con el botón derecho en un elemento del diagrama y seleccione **Seleccionar en la Estructura del modelo** en el menú contextual.
- También puede hacer clic en el elemento del diagrama y pulsar la tecla **F4**.

5.1.5 Convertir elementos

Algunos de los elementos se pueden convertir rápidamente en otro tipo de elementos. Esta acción puede resultar útil, por ejemplo, si empieza a diseñar una clase pero después prefiere que sea una interfaz o viceversa. Más concretamente, estos son los tipos de elementos que admiten la conversión en cualquiera de los otros elementos de la lista:

- Clase
- Interfaz
- Enumeración
- TipoPrimitivo
- TipoDeDatos

Puede convertir estos tipos de elementos tanto desde la Ventana Árbol de diagramas como desde la Ventana Estructura del modelo.

Para convertir elementos:

1. Abra un diagrama que incluya clases, interfaces, enumeraciones, tipos primitivos o tipos de datos (por ejemplo, un diagrama de clases). También puede buscar estos tipos de elementos en la Estructura del modelo.
2. Haga clic con el botón derecho del botón en el elemento en cuestión (por ejemplo, una clase) y seleccione **Convertir en | <tipo de elemento>** en el menú contextual.

Una vez haya hecho la conversión se conserva el nombre del elemento. Si es posible también se conservan los datos asociados a él. Por ejemplo, convertir una interfaz en una clase o viceversa conserva datos como propiedades u operaciones. Sin embargo, al convertir una clase o interfaz en una enumeración se pierden datos. En estos casos puede restaurar el tipo al estado previo a la conversión con el comando **Deshacer** (**Ctrl+Z**) si lo necesita.

5.1.6 Buscar y reemplazar texto

Puede buscar elementos de modelado, diagramas, texto, etc. dentro de cualquiera de estos lugares:

- panel *Diagramas*

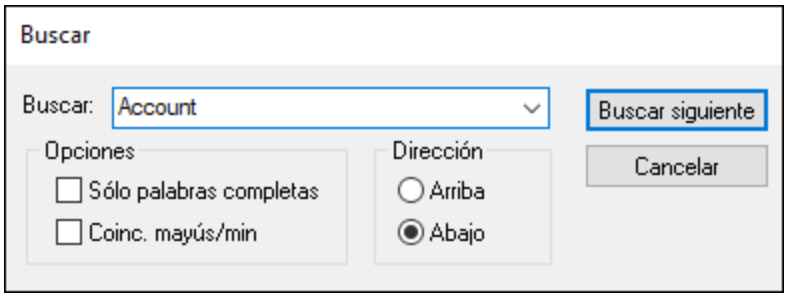
- ventana *Estructura del modelo*
- panel *Árbol de diagramas*
- panel *Favoritos*
- panel *Documentación*
- ventana *Mensajes*

El alcance de la búsqueda lo define el lugar en que esté situado el cursor. Por tanto, si quiere buscar texto dentro de un diagrama debe hacer clic dentro de ese diagrama primero. Asimismo, si quiere buscar un elemento en el proyecto de UModel debe hacer clic dentro de la ventana *Estructura del modelo*.

Para buscar texto o elementos:

1. Haga clic dentro de la ventana en el que quiere buscar el texto.
2. Escoja una de estas dos opciones:
   a. Teclee el texto que quiere buscar en la caja de texto de la barra de herramientas principal y haga clic en **Encontrar siguiente** o pulse la tecla **F3**. Para ir a la aparición anterior del texto de búsqueda, pulse **Mayúsculas+F3**.

   b. En el menú **Edición** haga clic en **Buscar** (o pulse **Ctrl+F**).



Buscar y reemplazar

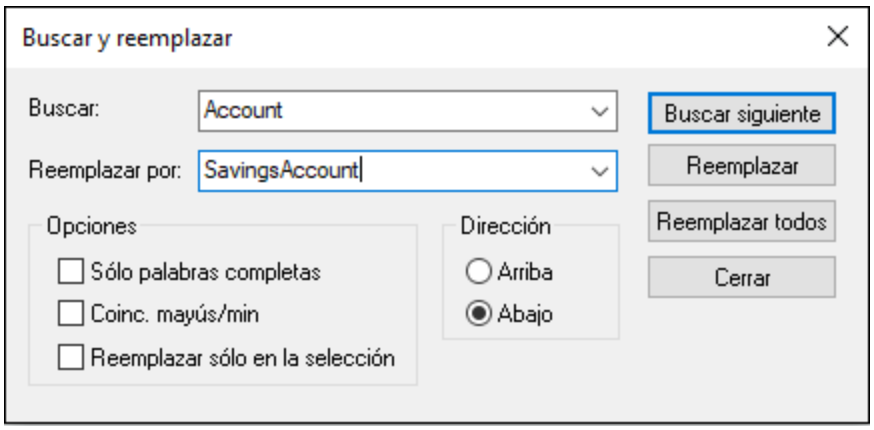
También puede buscar y reemplazar texto (por ejemplo para renombrar rápidamente elementos de modelado). Cuando UModel encuentra el elemento, lo resalta en el diagrama además de en la ventana *Estructura del modelo*. La función **Buscar y reemplazar** funciona en los siguientes paneles:

- ventana de diagramas
- ventana *Estructura del modelo*
- ventana *Árbol de diagramas*
- ventana *Favoritos*
- ventana *Documentación*

Para buscar y reemplazar texto:

1. Haga clic en la ventana en la que quiere buscar y reemplazar texto.
2. Escoja una de las siguientes opciones:

c. Haga clic en el botón **Reemplazar** de la barra de herramientas.
d. En el menú **Edición** haga clic en **Reemplazar** (o pulse **Ctrl+H**).



5.1.7 Comprobar si se están usando ciertos elementos y dónde

Al navegar por los elementos de la *Estructura del modelo* puede que quiera ver si se está usando un elemento o en qué partes de un diagrama del modelo se encuentra. Hay varias opciones para encontrar dónde se está usando un elemento:

- Haga clic con el botón derecho en el elemento en la ventana *Estructura del modelo* y seleccione **Mostrar el elemento en todos los diagramas** (o, si un diagrama ya está abierto, **Mostrar el elemento en el diagrama activo**).

También puede encontrar elementos que no se están usando en ningún diagrama, ni en el proyecto ni en paquetes individuales.

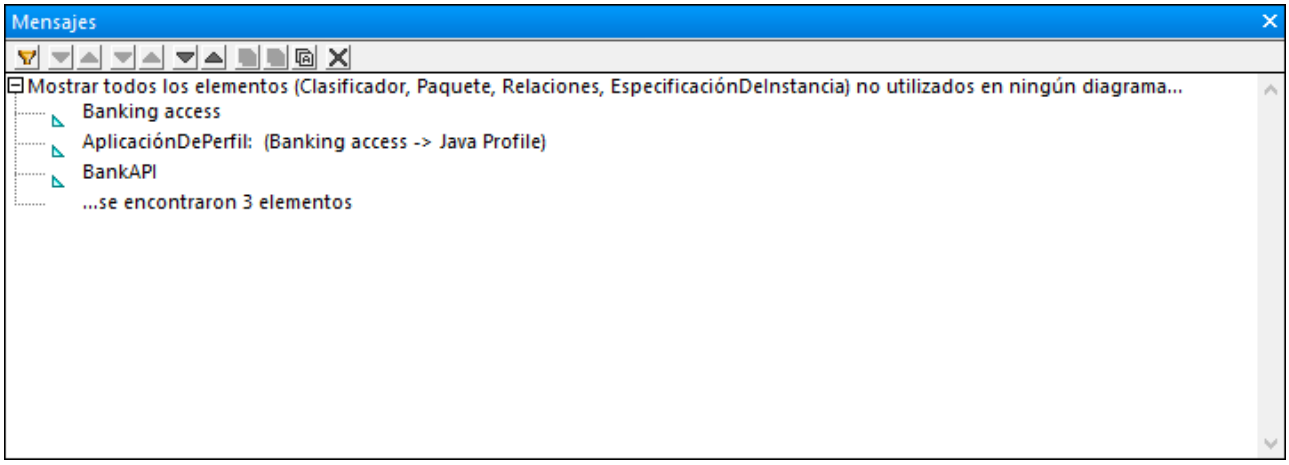
Para encontrar elementos no usados en todo el proyecto:

- En el menú **Proyecto** haga clic en **Mostrar elementos no utilizados en ningún diagrama**.

Para encontrar elementos no usados en un paquete específico:

- Haga clic con el botón derecho en el paquete que quiere inspeccionar y seleccione **Mostrar elementos no utilizados en ningún diagrama**.

Aparecerá una lista de elementos no usados en la ventana *Mensajes*. Tenga en cuenta que se muestran los elementos no usados en ese paquete o sus paquetes subordinados. Los elementos contenidos entre paréntesis han sido configurados para aparecer en la lista de elementos no utilizados desde la pestaña **Herramientas | Opciones | Vista**.



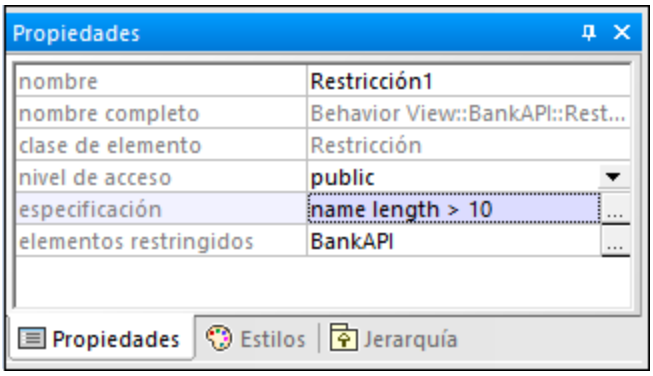
Haga clic en el nombre del elemento dentro de la ventana *Mensajes* para ubicarlo en la *Estructura del modelo*.

5.1.8 Restricción de elementos

En UModel se pueden definir restricciones para la mayoría de elementos de modelado. Tenga en cuenta que el revisor de sintaxis no comprueba las restricciones porque estas no forman parte del proceso de generación de código.

Para restringir un elemento (desde la ventana *Estructura del modelo*):

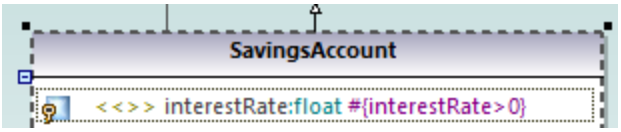
1. Haga clic con el botón derecho en el elemento que quiere restringir y seleccione **Elemento nuevo | Restricciones | Restricción**.
2. Introduzca el nombre de la restricción y pulse **Entrar**.
3. Teclee el texto de la restricción en el campo "especificación" de la ventana Propiedades (por ejemplo, `name length > 10`)



Para restringir un elemento (desde un diagrama):

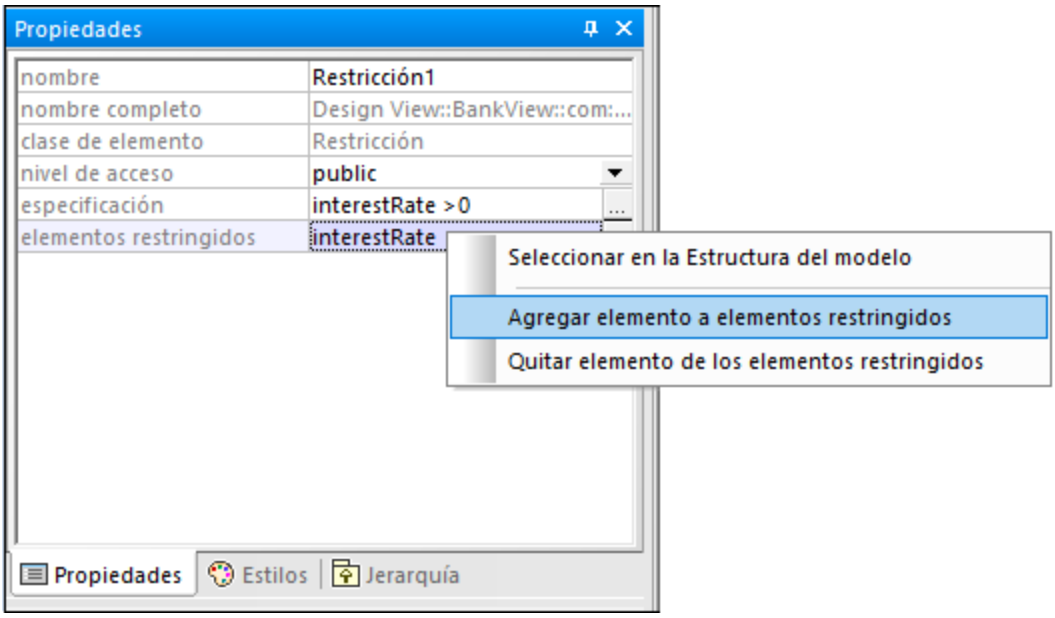
1. Haga doble clic en el elemento especificado para poder editarlo.

2. Teclee "#" e introduzca a continuación el texto de la restricción dentro de los corchetes, por ejemplo `#{interestRate >=0}`.

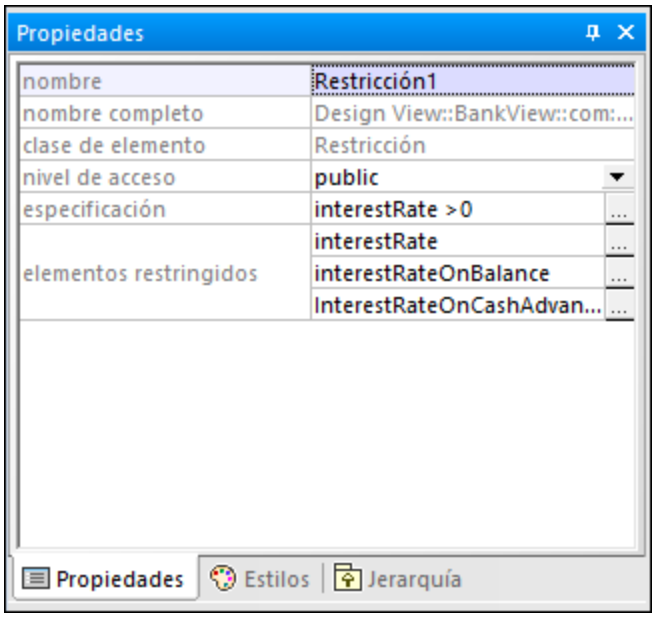


Para asignar restricciones a múltiples elementos de modelado:

1. Seleccione una restricción en la ventana *Estructura del modelo*.
2. Haga clic con el botón derecho en la propiedad "elementos restringidos" de la ventana Propiedades y seleccione **Agregar elemento a elementos restringidos**.



3. Seleccione el elemento al que quiere añadir la restricción actual. Mantenga pulsada la tecla **Ctrl** para seleccionar múltiples elementos.



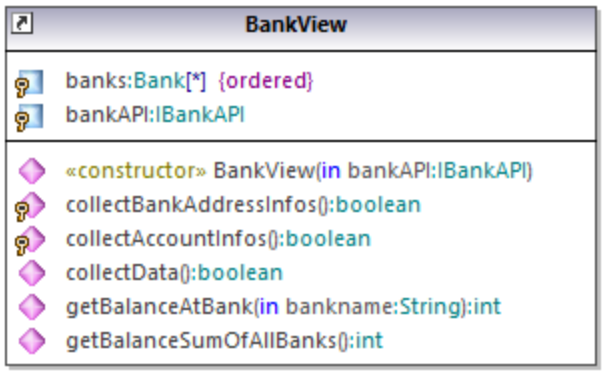
El campo "elementos restringidos" contiene los nombres de los elementos de modelado a los que ha sido asignado. Por ejemplo, en la imagen anterior, `Constraint1` ha sido asignado a las siguientes propiedades: `interestRate`, `interestRateOnBalance`, `interestRateOnCashAdvance`.

5.1.9 Agregar hipervínculos a elementos

Puede crear manualmente hipervínculos entre la mayoría de los elementos de modelado (excepto las líneas) y cualquiera de los siguientes:

- otros elementos (en el diagrama o en la *Estructura del modelo*)
- diagramas
- archivos externos al proyecto (por ejemplo, documentos PDF, Word o Excel, archivos de imagen, etc.)
- páginas web

Un único elemento puede tener uno o más hipervínculos de los mencionados anteriormente. En un diagrama, los elementos que contienen hipervínculos se pueden reconocer fácilmente por el icono de hipervínculo visible junto a ellos (que puede estar en la esquina derecha o izquierda). Para abrir el destino del hipervínculo haga clic con el botón derecho en el icono del elemento y seleccione el destino. Si solo hay un hipervínculo definido, también puede hacer clic en y acceder directamente al destino.



Hipervínculos que contienen clases

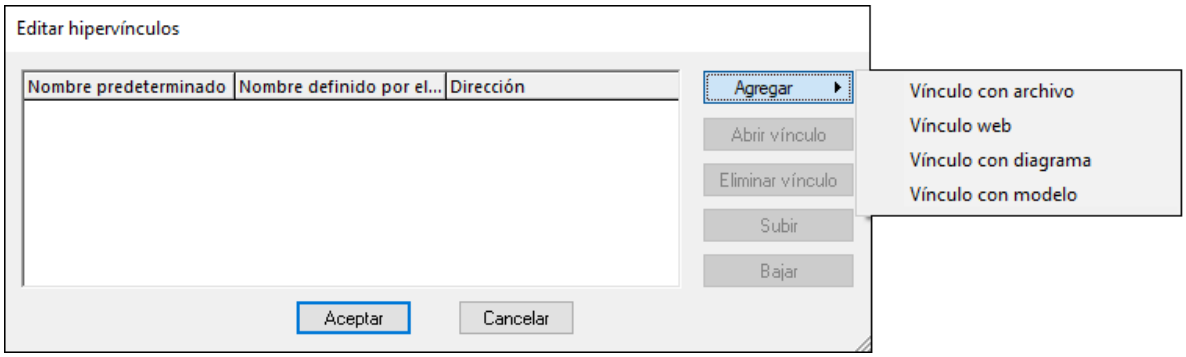
Nota: cuando navegue por la interfaz gráfica del usuario de UModel, con o sin hipervínculos, puede moverse fácilmente haciendo clic en los respectivos botones de la barra de herramientas **Atrás**  o **Adelante**  .

Puede generar automáticamente hipervínculos entre paquetes dependientes y diagramas importando código fuente o archivos binarios a un modelo, siempre que seleccione las opciones correspondientes en el cuadro de diálogo de importación. Para más información, consulte Importar código fuente y Importar archivos binarios Java, C# y VB. Cuando genere documentación UML para su proyecto, puede escoger si incluir hipervínculos en los resultados que se generen o no. Consulte Generar documentación UML.

No solo puede crear hipervínculos desde elementos que aparecen en el diagrama o en la ventana *Estructura del modelo*, sino también desde texto de notas o desde el texto de la ventana *Documentación*, como se muestra más abajo.

Para crear un hipervínculo desde un elemento:

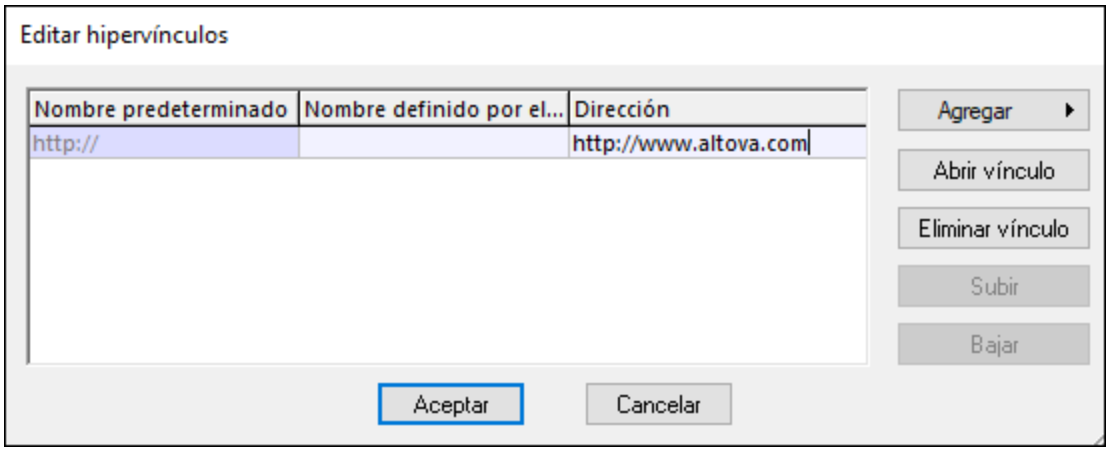
1. Haga clic con el botón derecho en un elemento de un diagrama o en la ventana *Estructura del modelo* y seleccione **Hipervínculos | Insertar o editar hipervínculos** del menú contextual.
2. Haga clic en **Agregar** y seleccione un tipo de hipervínculo (vínculo con archivo, vínculo web, vínculo con diagrama o vínculo con modelo).



3. Escoja una de estas opciones:

- para crear un diagrama o hipervínculo, seleccione el elemento destino o diagrama cuando aparezca el cuadro de diálogo correspondiente.
- para crear un hipervínculo a un archivo haga clic en el botón de puntos y busque el archivo de destino.

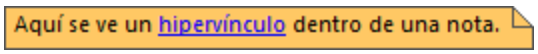
- para crear un hipervínculo web, teclee la dirección de destino en la columna "Dirección" el cuadro de diálogo, por ejemplo:



4. Otra opción es introducir un nombre de enlace personalizado en la columna "Nombre definido por el usuario". Si se define, este nombre se mostrará en la interfaz gráfica del usuario de UModel en lugar de la ruta o dirección de destino.

Para crear un hipervínculo dentro de una nota:

- seleccione texto dentro de la nota haga clic en él con el botón derecho y seleccione **Insertar o editar hipervínculos** en el menú contextual. Siga los mismos pasos que los indicados para la ventana *Documentación*.



Para cambiar o eliminar un hipervínculo:

- haga clic con el botón derecho en el icono del hipervínculo del elemento (o en el texto del hipervínculo) y use el comando apropiado en el cuadro de diálogo "Editar hipervínculos".

5.1.10 Documentar elementos

Puede añadir comentarios de documentación a elementos de modelado de la siguiente manera:

- Haga clic en el elemento (en el diagrama o en la ventana *Estructura del modelo*).
- Introduzca texto en la ventana *Documentación*.

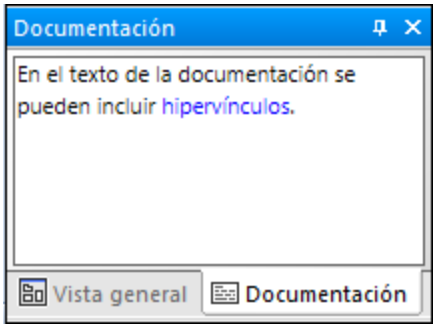
Cualquier texto de documentación se guardará junto con el proyecto.

Cuando se selecciona un elemento, su documentación, si es que existe, siempre está visible en la ventana *Documentación*. También puede mostrar la documentación como un comentario en el diagrama:

- Haga clic con el botón derecho en el elemento del diagrama y seleccione **Mostrar | Comentarios de anotación** en el menú contextual.

Hipervínculos de la documentación

Para crear un hipervínculo dentro de la ventana *Documentación*, seleccione texto en la ventana, haga clic en él con el botón derecho y seleccione **Insertar o editar hipervínculos** en el menú contextual. El destino del hipervínculo puede ser una página web, un diagrama, un archivo u otro elemento (véase Agregar hipervínculos a elementos).



Panel Documentación

Generación de código y comentarios de la documentación

Si genera código desde diagramas de clases, cualquier comentario aplicado a clases y a sus miembros se puede exportar también al código generado. Para ello, seleccione la casilla **Escribir documentación como JavaDocs** (para Java) o Escribir documentación como DocComments (para C# y VB.NET) antes de generar el código de programa. Consulte también Opciones de generación de código.

Asimismo, si aplica ingeniería inversa al código de programa para transformarlo en un modelo, los comentarios del código se pueden importar al modelo. Para ello, seleccione la casilla **JavaDocs como documentación** (para Java) o **DocComments como documentación** (para C# y VB.NET) antes de aplicar la ingeniería inversa al código de programa. Consulte también Opciones de importación de código.

Para más información sobre cómo los comentarios en el código de programa (o esquemas XML) están asignados a los comentarios de UModel, consulte las tablas de asignaciones de cada lenguaje:

- Asignaciones C#
- Asignaciones VB.NET
- Asignaciones Java
- Asignaciones de esquemas XML

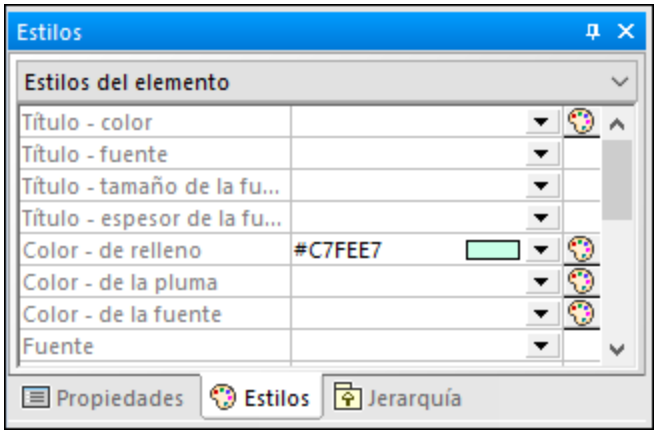
5.1.11 Cambiar el estilo de los elementos de un diagrama

Puede cambiar el aspecto (estilo) de los elementos de modelado, incluidos su color, tamaño de fuente, peso de fuente, color de fondo, grosor de línea, etc. El aspecto de los elementos se puede cambiar a varios niveles: globalmente para todos los elementos del proyecto, de forma selectiva para todos los elementos de la misma familia (por ejemplo, clases) o para cada elemento individual. Para saber más sobre cambiar el estilo del diagrama, consulte Cambiar el estilo de diagramas.

Es posible usar imágenes personalizadas en lugar de las representaciones convencionales de los elementos de diagramas si amplía si proyecto con perfiles y estereotipos personalizados. Para más información consulte Ejemplo: personalizar iconos y estilos.

Cambiar el aspecto de elementos:

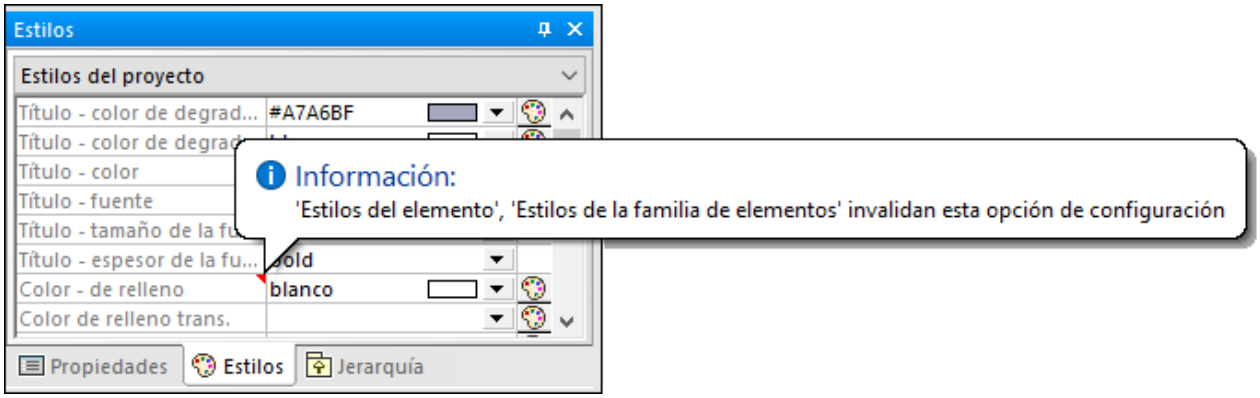
1. Haga clic en un elemento de un diagrama.
2. Aparecerá una lista desplegable en la parte superior de la ventana Estilos:
   a. Para editar solamente las propiedades del elemento actual, seleccione "Estilos del elemento" de la lista.



   b. Para editar las propiedades de todos los elementos del mismo tipo (por ejemplo, clases), seleccione "Estilos de la familia de elementos" de la lista.
   c. Para editar las propiedades de todos los elementos a nivel de proyecto, seleccione "Estilos del proyecto".
   d. Para editar las propiedades de todas las líneas del proyecto, incluidas las líneas de asociación, dependencia y realización, seleccione "Estilos de línea". (Este valor solo es visible si el elemento seleccionado es una línea.)
   e. Para editar las propiedades de todos los elementos que no son líneas (los llamados nodos) en todo el proyecto, seleccione "Estilos de nodos". (Este valor solo es visible si el elemento seleccionado no es una línea.)
3. Cambiar el valor de la propiedad en cuestión (por ejemplo "Color de relleno").

Un estilo genérico se ve invalidado por un estilo más específico. Es decir, los estilos que se apliquen a elementos individuales invalidan a los que se hayan aplicado a nivel de familia de elementos. Asimismo, los estilos aplicados a familias de elementos invalidan a aquellos aplicados a nivel de proyecto.

Cuando un estilo es invalidado aparece un pequeño triángulo rojo en la esquina superior derecha de la propiedad invalidada. Mueva el cursor sobre el triángulo para mostrar la información rápida sobre el estilo invalidado.



Estilo de elemento invalidado

5.2 Diagramas

5.2.1 Crear diagramas

Los diagramas representan visualmente cómo interactúan los elementos de modelado, cuáles son su estructura, sus dependencias, su jerarquía, etc. Los diagramas deben pertenecer a un paquete dentro del proyecto, por lo que deben crearse dentro de un paquete que ya existe dentro de la ventana *Estructura del modelo*. Puede mover diagramas de un paquete a otro en cualquier momento arrastrándolos hasta el paquete de destino.

Para crear un nuevo diagrama:

1. Haga clic con el botón derecho en un paquete de la ventana *Estructura del modelo*.
2. Seleccione **Diagrama nuevo | <Tipo de diagrama>**.

También puede crear un diagrama nuevo desde la ventana *Árbol de diagramas*:

1. Haga clic con el botón derecho en el nodo raíz ("Diagramas") en la ventana *Árbol de diagramas*.
2. Seleccione el paquete al que debe pertenecer el diagrama y haga clic en **Aceptar**.

Cuando una ventana de diagrama está activa, la barra de herramientas solo muestra los elementos de modelado aplicables al tipo de diagrama en cuestión. El tipo de diagrama se muestra en la ventana Propiedades después de que haga clic en un área vacía del diagrama. Los siguientes iconos indican el tipo de diagrama:

Icono	Descripción
	Diagrama de actividades
	Diagrama de proceso empresarial BPMN 1
	Diagrama de proceso empresarial BPMN 2
	Diagrama de coreografía BPMN 2
	Diagrama de colaboración BPMN 2
	Diagrama de clases
	Diagrama de comunicación
	Diagrama de componentes
	Diagrama de estructura compuesta
	Diagrama de BD
	Diagrama de implementación

Icono	Descripción
	Diagrama global de interacción
	Diagrama de objetos
	Diagrama de paquetes
	Diagrama de perfil
	Diagrama de máquina de estados de protocolos
	Diagrama de secuencia
	Diagrama de máquina de estados
	Diagramas SysML (9 tios de diagramas)
	Diagrama de ciclo de vida
	Diagrama de casos de uso
	Diagrama de esquema XML

5.2.2 Generar diagramas

Además de crear diagramas desde cero, también puede generar automáticamente ciertos diagramas a partir de elementos de modelado que ya existen o desde código de programa. En esta sección explicamos cómo generar diagramas a partir de elementos de modelado que ya existen. Para obtener más información sobre cómo generar diagramas a partir de código fuente, consulte las siguientes secciones:

- Generar diagramas de clases
- Generar diagramas de secuencia a partir de código fuente
- Generar diagramas de paquetes al importar código o binarios

Para generar diagramas a partir de elementos que ya existen haga clic con el botón derecho en un elemento (por ejemplo, un paquete) de la Estructura del modelo y seleccione **Mostrar en un diagrama nuevo | <opción>** en el menú contextual. Aquí tiene algunos ejemplos:

Para crear un diagrama que muestra el contenido de un paquete:

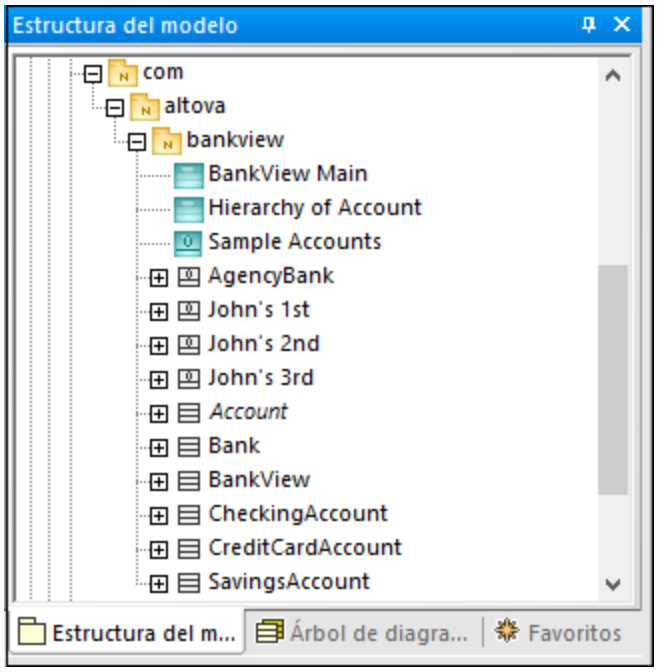
- Haga clic con el botón derecho en un paquete de la Estructura del modelo y seleccione **Mostrar en un diagrama nuevo | Contenido** en el menú contextual.

Para crear un diagrama que muestre las dependencias de un paquete:

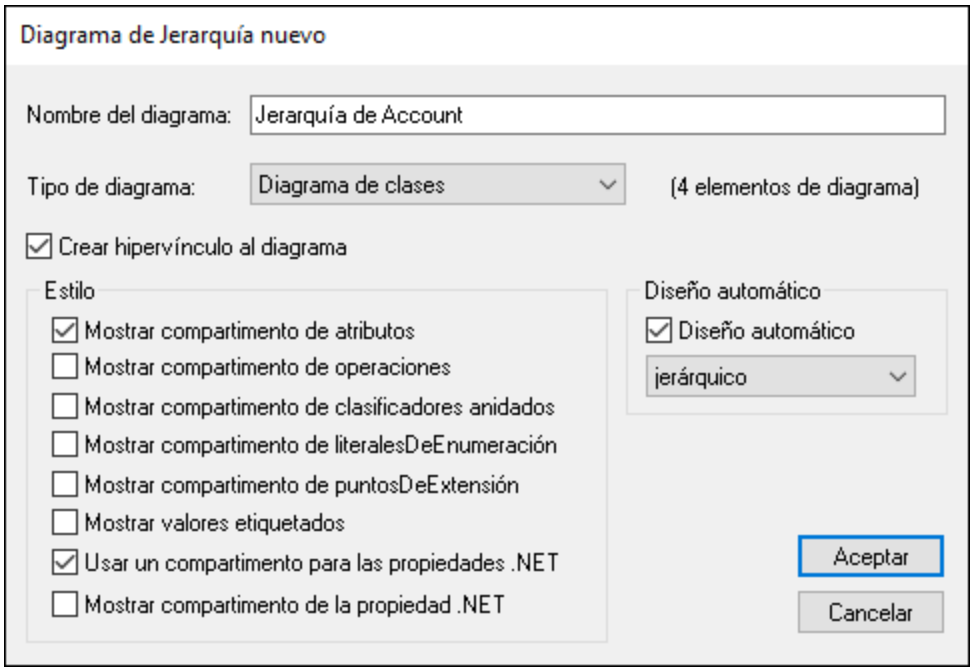
- Haga clic con el botón derecho en un paquete de la ventana *Estructura del modelo y* seleccione **Mostrar en un diagrama nuevo | Dependencias entre paquetes** en el menú contextual.

Para crear un diagrama que muestre la generalización jerárquica de una clase:

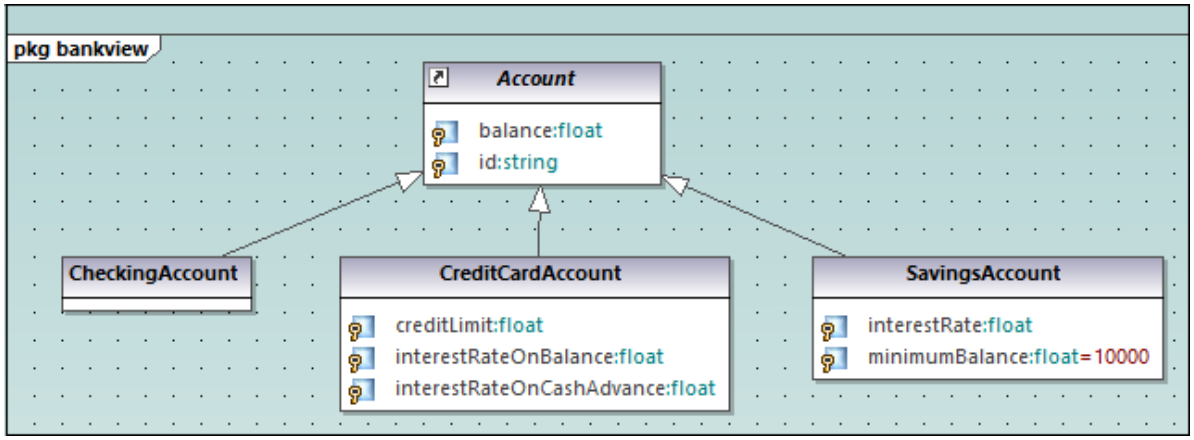
1. En la ventana *Estructura del modelo* haga clic con el botón derecho en una clase que tenga relaciones de generalización hacia o desde otras clases (por ejemplo, la clase `Account` del proyecto de prueba **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Bank_CSharp.ump**).



2. Seleccione **Mostrar en un diagrama nuevo | Jerarquía de la generalización** en el menú contextual. Aparecerá un cuadro de diálogo en el que, entre otras preferencias para el diagrama que va a crear, podrá ajustar el tipo de diagrama. El texto "N elementos de diagrama", que muestra el número de elementos que se añadirán al diagrama. En la imagen siguiente, tipo de diagrama elegido es "Diagrama de clases" y el diagrama tendrá cuatro elementos (clases): la clase `Account` y tres clases derivadas de ella.

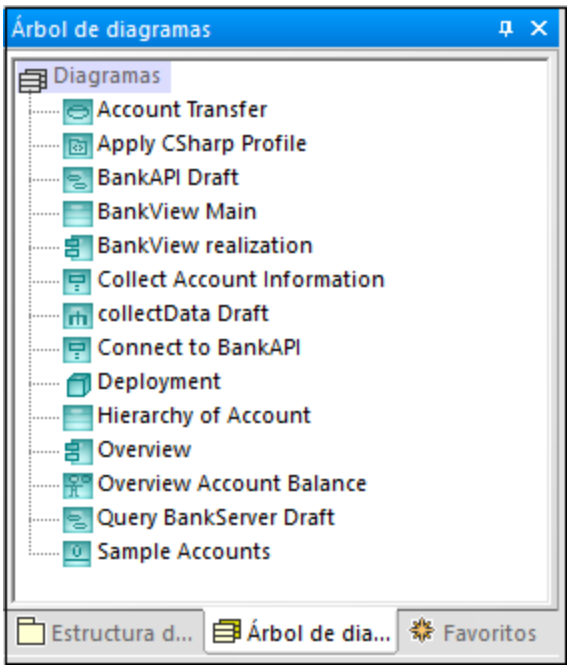


3. Haga clic en **Aceptar**. El diagrama se genera conforme a las opciones seleccionadas y se abre en la Ventana de diagramas, por ejemplo:



5.2.3 Abrir diagramas

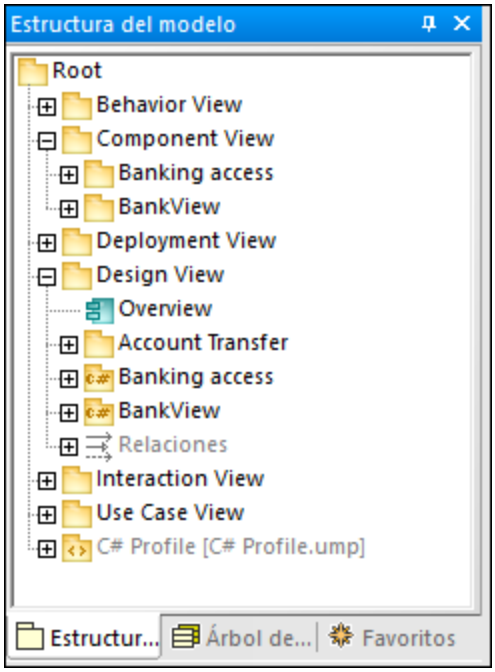
Si el proyecto de UModel contiene diagramas, estos se muestran en el Árbol de diagramas.



Árbol de diagramas

Nota: por defecto, los diagramas están agrupados por tipo en la ventana *Árbol de diagramas*. Para mostrar únicamente diagramas (sin grupos primarios) haga clic con el botón derecho dentro de la ventana y desmarque la opción **Agrupar por tipo de diagrama** en el menú contextual.

Los diagramas también se muestran en la ventana *Estructura del modelo* bajo los paquetes a los que pertenecen, por ejemplo:



Para abrir un diagrama que ya existe:

- Haga doble clic en el icono del diagrama en la ventana *Estructura del modelo* (o en los paneles Árbol de diagramas o Favoritos).
- Haga clic con el botón derecho en el diagrama y seleccione **Abrir diagrama** en el menú contextual.

5.2.4 Borrar diagramas

Los diagramas de UModel se pueden eliminar de las siguientes formas:

- En la ventana *Estructura del modelo* (o en los paneles Árbol de diagramas o Favoritos) haga clic con el botón derecho en el diagrama y seleccione **Eliminar** en el menú contextual.
- Haga clic en el diagrama en cualquiera de los paneles antes mencionados y pulse la tecla **Suprimir**.

Al eliminar un diagrama no se elimina ningún otro elemento del proyecto. Para comprobar si se está usando un elemento en algún diagrama haga clic con el botón derecho en el paquete que quiere inspeccionar y seleccione **Mostrar elementos no utilizados en ningún diagrama**. Consulte también Comprobar si se están usando ciertos elementos y dónde.

Para obtener más información sobre cómo borrar elementos de un diagrama o de un proyecto, consulte Borrar elementos.

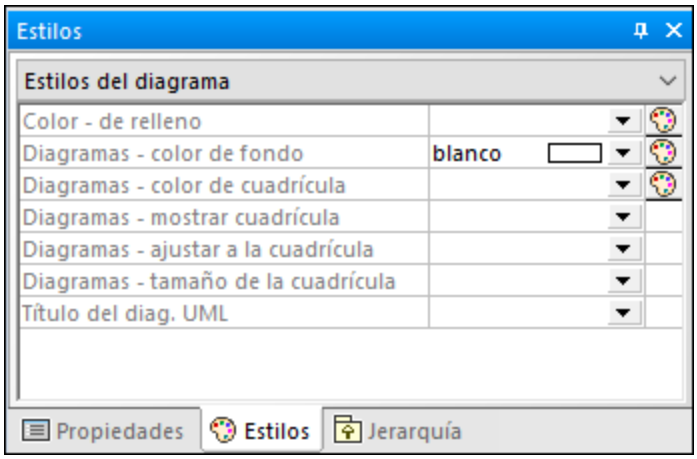
5.2.5 Cambiar el estilo de diagramas

Puede modificar el aspecto (el estilo) de un diagrama, incluidos el color de fondo, la visibilidad, el tamaño o el color de la cuadrícula, así como el aspecto del encabezado del diagrama. Puede cambiar el estilo de diagramas individuales dentro del proyecto o aplicar las mismas propiedades a todos los diagramas de un proyecto. Para obtener más información sobre cómo cambiar el estilo de los elementos de un diagrama, consulte Cambiar el estilo de los elementos de un diagrama.

El tamaño de los diagramas se define en función de sus elementos y la ubicación de estos. Para aumentar el tamaño de un diagrama, arrastre un elemento hasta una de las esquinas del diagrama y el tamaño se ajustará a la nueva posición.

Para cambiar el aspecto de diagramas:

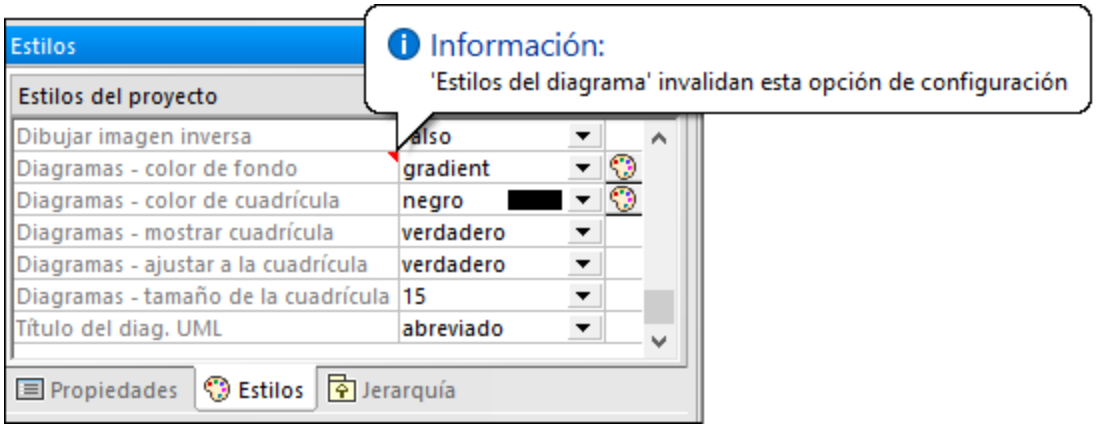
1. Abra un diagrama (*véase Abrir diagramas*).
2. En la parte superior de la ventana *Estilos* hay una lista desplegable que ofrece dos opciones:
   a. Para editar únicamente las propiedades del diagrama actual, seleccione en esa lista "Estilos del diagrama". Este es el valor predeterminado si hace clic en cualquier parte del fondo vacío del diagrama (es decir, sin hacer clic en ningún elemento).



   b. Para aplicar cambios a todos los diagramas del proyecto, seleccione "Estilos del proyecto". Desplácese hasta abajo del todo en la ventana *Estilos* hasta que encuentre los estilos aplicables a diagramas (los que empiezan con "Diagramas - ").
3. Cambie el valor de las propiedades que quera (por ejemplo, "Diagrama - color de fondo").

Los estilos que se apliquen a nivel de diagrama invalidan a aquellos aplicados a nivel de proyecto.

Cuando un estilo es invalidado aparece un pequeño triángulo rojo en la esquina superior derecha de la propiedad invalidada. Mueva el cursor sobre el triángulo para mostrar la información rápida sobre el estilo invalidado.



Estilo de diagrama invalidado

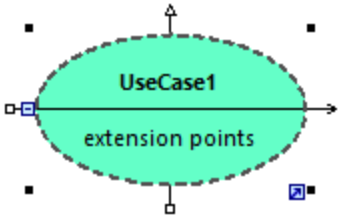
Las siguientes propiedades específicas de diagramas están disponibles como botones en la barra de herramientas. Cambiar la propiedad en la ventana *Estilos* actualizará el estado del botón de la barra de herramientas y viceversa.

	Mostrar cuadrícula	Muestra u oculta la cuadrícula del diagrama.
	Mostrar título del diagrama UML	Muestra u oculta el título del diagrama.
	Ajustar a la cuadrícula	Cuando está activada, esta propiedad hace que todos los elementos se adhieran a la cuadrícula. Cuando está desactivada, los elementos se posicionan independientemente del patrón de malla.

5.2.6 Alinear y ajustar el tamaño de elementos de modelado

Puede cambiar el tamaño de los elementos del diagrama como sigue:

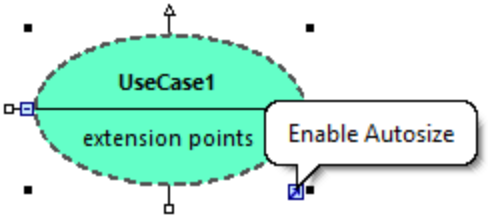
1. Haga clic en un elemento del diagrama. Aparecerán unos puntos negros en los bordes del elemento.



2. Arrastre cualquiera de los puntos negros en la dirección en que quiere agrandar el elemento.

Para devolver el elemento a su tamaño predeterminado, tiene varias opciones:

- Haga clic en el icono **Habilitar ajuste de tamaño automático** que encontrará en la esquina inferior izquierda del elemento.



- Haga clic con el botón derecho en el diagrama y seleccione el comando **Ajustar tamaño automáticamente** en el menú contextual.
- Seleccione uno o más elementos. En el menú **Diseño**, seleccione **Ajustar tamaño automáticamente**.

Cuando se seleccionan al menos dos elementos de modelado en el diagrama, estos se pueden alinear en relación uno con otro (por ejemplo, ambos se pueden alinear para que tengan la misma posición en el eje horizontal o vertical, o el mismo tamaño). Los comandos que alinean o cambian el tamaño de los elementos están disponibles en el menú **Diseño** y en la barra de herramientas de diseño.

Barra de herramientas de diseño

Cuando selecciona varios elementos, el elemento que se seleccionó **en último lugar** sirve como plantilla para los comandos de alineación o cambio de tamaño. Por ejemplo, si selecciona tres elementos de clase y ejecuta el comando **Igualar ancho**, los tres elementos tendrán el tamaño del último que seleccionó. El borde del último elemento seleccionado es siempre una línea discontinua.

A continuación mostramos los comandos de alineación y cambio de tamaño de elementos:

Icono	Comando	Notas
	Alinear a la izquierda	
	Alinear a la derecha	
	Alinear arriba	
	Alinear abajo	
	Centrar verticalmente	
	Centrar horizontalmente	
	Espaciar en horizontal	Este comando está disponible cuando se seleccionan tres o más elementos y distribuye de manera uniforme el espacio horizontal entre los elementos seleccionados.
	Espaciar en vertical	Este comando está disponible cuando se seleccionan tres o más elementos y distribuye de manera uniforme el espacio vertical entre los elementos seleccionados.

Icono	Comando	Notas
	**Poner en fila horizontal**	Este comando vuelve a posicionar en el diagrama todos los elementos seleccionados para que estén ordenados unos después de otros en horizontal.
	**Poner en fila vertical**	Este comando vuelve a posicionar en el diagrama todos los elementos seleccionados para que estén ordenados uno debajo del otro.
	**Igualar ancho**	
	**Igualar alto**	
	**Igualar tamaño**	

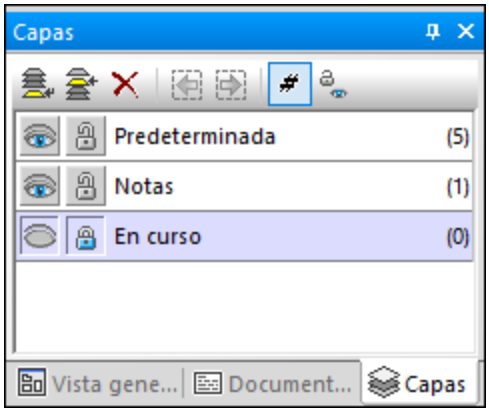
También puede aplicar un diseño automático a todos los elementos del diagrama:

- En el menú **Diseño** haga clic en **Aplicar diseño automático a todo** y escoja una de las siguientes opciones: **Diseño dirigido por fuerzas**, **Diseño jerárquico** o **Diseño por bloques**.

Diseño dirigido por fuerzas	Muestra los elementos de modelado desde un punto de vista céntrico.
Diseño jerárquico	Muestra los elementos en función de sus relaciones jerárquicas. Por ejemplo, una superclase se colocará por encima de cualquiera de sus clases derivadas. Se pueden ajustar las opciones del diseño jerárquico en el menú **Herramientas | Opciones**, pestaña **Vista**, grupo **Autodiseño de la jerarquía**.
Diseño por bloques	Muestra los elementos en un rectángulo y agrupados por tamaño.

5.2.7 Añadir capas a los diagramas

Un diagrama consiste por defecto en una sola capa que almacena todos los elementos visibles en la superficie del diagrama. Sin embargo, también se pueden añadir múltiples capas a un diagrama. Estas capas permiten agrupar los elementos de modelado de forma lógica dentro del mismo diagrama. Por ejemplo, además de la capa predeterminada, también puede crear nuevas capas que contengan notas con información interna o clases inacabadas. Las capas se pueden visualizar y gestionar desde la ventana *Capas*.



Panel Capas

En la imagen anterior se han definido tres capas en un diagrama, de las cuales la capa "Notas" está actualmente seleccionada y la capa "En curso" está actualmente bloqueada. El número que se muestra entre paréntesis a la derecha de cada capa indica cuántos elementos tiene cada capa.

Se puede asignar cualquier elemento UML a cualquiera de las capas. Por defecto, los nuevos elementos se añaden a la capa que esté activa en ese momento (esta está resaltada en la ventana *Capas*). Si todas las capas son visibles, puede crear relaciones de asociación, generalización, etc. entre elementos de distintas capas.

Al imprimir diagramas o al guardarlos como imagen, solo se imprimen los elementos de la capa visible en ese momento. El número máximo de capas por diagrama es 20.

Estos son los botones disponibles en la ventana *Capas*:

Icono	Comando	Notas
	**Anexar capa**	Anexa una nueva capa a la lista de capas y le asigna un nombre predeterminado que usted puede cambiar de inmediato o más tarde con el menú contextual "Renombrar".
	**Insertar capa**	Inserta una nueva capa por encima de la capa activa en ese momento.
	**Eliminar capa**	Borra la capa activa en ese momento. Antes de borrar la capa aparece un cuadro de diálogo que permite indicar a dónde deben moverse (combinarse) los elementos de esa capa (si los hay).
	**Resaltar elemento anterior en la capa activa**	Selecciona el elemento anterior en la capa activa en ese momento. Este comando se habilita solo si la capa contiene elementos.
	**Resaltar siguiente elemento en la capa activa**	Selecciona el elemento siguiente en la capa activa en ese momento. Este comando se habilita solo si la capa contiene elementos.

Icono	Comando	Notas
	**Recuento de elementos de capa**	Muestra u oculta el recuento de elementos de cada capa.
	**Restaurar todos los estados de las capas**	Cambia el estado de todas las capas a visible y sin bloquear.

También puede acceder a algunos de estos comandos como elementos de menú contextual al hacer clic con el botón derecho dentro de la ventana *Capas*.

Para mover elementos de una capa a otra:

- Haga clic con el botón derecho en un elemento del diagrama y seleccione el comando **Capa | <nombre de capa>** en el menú contextual. Este comando también se puede aplicar después de seleccionar varios elementos; en este caso, todos ellos se moverán a la capa de destino.
- Otra opción es seleccionar uno o más elementos en el diagrama y arrastrarlos a la capa de destino en la ventana *Capas*.
- Para mover todos los elementos de una capa a otra distinta haga clic con el botón derecho en la capa y seleccione **Combinar con | <nombre de capa>** en el menú contextual.

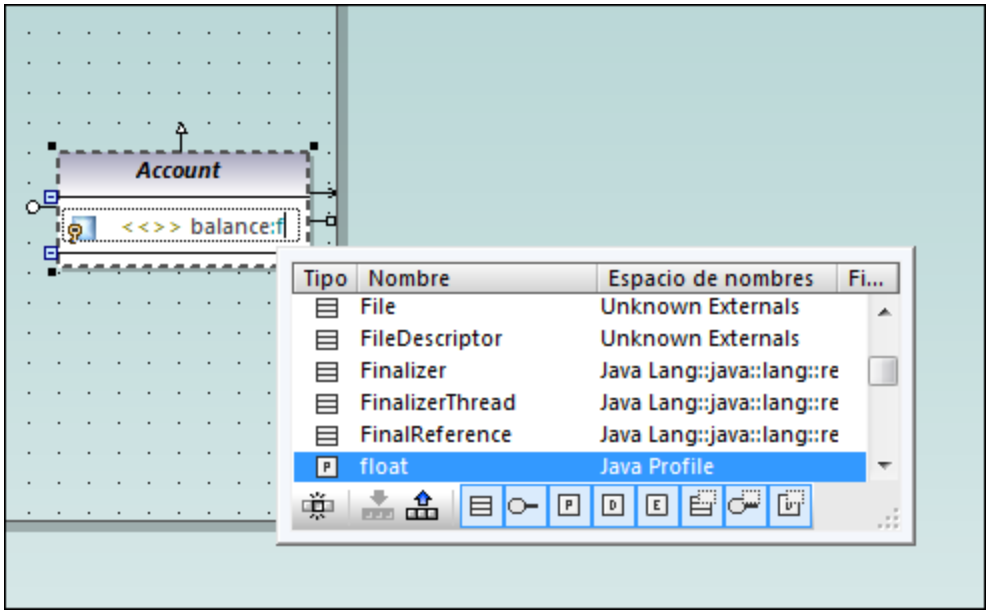
Para mostrar, ocultar o bloquear capas individuales o varias capas de una vez:

- Haga clic con el botón derecho en la ventana *Capas* y seleccione el comando que necesite de entre **Mostrar**, **Ocultar** y **Bloquear**. Los comandos de submenú **la capa seleccionada** y **las otras capas** permiten usar estos comandos para la capa seleccionada o para todas las demás.
- Otra opción es hacer clic con el botón derecho en la capa y usar **Alternar visibilidad** o **Alternar bloqueo**. Esto ocultará las capas que fueran visibles hasta entonces o las bloqueará si previamente no lo estaban (y viceversa).

5.2.8 Finalización automática en clases

Al añadir operaciones y atributos a una clase, la finalización automática del tipo de datos está habilitada por defecto en UModel. Esto permite especificar el tipo de datos de la operación o propiedad directamente en el diagrama, por ejemplo:

1. Haga clic con el botón derecho en una clase y seleccione **Nuevo/a | Operación** del menú contextual.
2. Teclee el nombre de la operación después de los paréntesis angulares <<>> y después dos puntos (:).
3. Al teclear los dos puntos se abrirá automáticamente una ventana de finalización automática.



Ventana de finalización automática

La ventana de finalización automática tiene las siguientes características:

- Al hacer clic en el nombre de una de las columnas, los contenidos de esa columna se colocan en orden ascendiente o descendiente conforme a ese atributo.
- Se puede cambiar el tamaño de la ventana haciendo clic en la esquina inferior derecha de la ventana y arrastrando hacia afuera.
- El contenido de la ventana se puede filtrar haciendo clic en los filtros correspondientes (categorías), en la parte inferior de la ventana: Clase, Interfaz, TipoPrimitivo, TipoDeDatos, Enumeración, Plantilla de clase, Plantilla de interfaz, Plantilla de tipo de datos.

Para habilitar solo un filtro:

- Haga clic en el botón Modo único . La imagen anterior muestra la ventana de finalización automática en "modo múltiple", es decir, que todos los filtros están habilitados. El botón de modo único no está habilitado.

Para seleccionar o deshabilitar todos los filtros al mismo tiempo:

- Haga clic en los botones **Activar todas las categorías** o **Desactivar todas las categorías** .

Para deshabilitar la finalización automática:

1. En el menú **Herramientas** haga clic en **Opciones** y luego en la pestaña **Edición de diagramas**.
2. Desactive la casilla **Habilitar ayudante de entrada automática**.

Para activar la finalización automática a voluntad (cuando se encuentra deshabilitada):

1. Asegúrese de que el cursor está dentro de un atributo o de una operación de una clase, después de los dos puntos (:).
2. Pulse **Ctrl+Barra espaciadora**.

5.2.9 Acercar y alejar diagramas

Para acercar la vista del diagrama de una de las siguientes maneras:

- Ejecute el comando de menú **Vista | Acercarse** (**Ctrl+Mayús+I**) o **Vista | Alejarse** (**Ctrl+Mayús+O**).
- Seleccione un valor porcentual predefinido en la barra de herramientas del zoom.

- Mantenga pulsada la tecla Ctrl mientras gira la rueda del ratón.

Para ajustar el área del diagrama a la ventana visible:

- Ejecute el comando de menú **Vista | Ajustar a la ventana** (o haga clic en el botón de la barra de herramientas **Ajustar a la ventana**).

5.3 Relaciones

5.3.1 Crear relaciones entre elementos

Una relación necesita dos elementos, por lo que su diagrama deberá contener elementos a los que añadir relaciones. Puede crear relaciones como sigue:

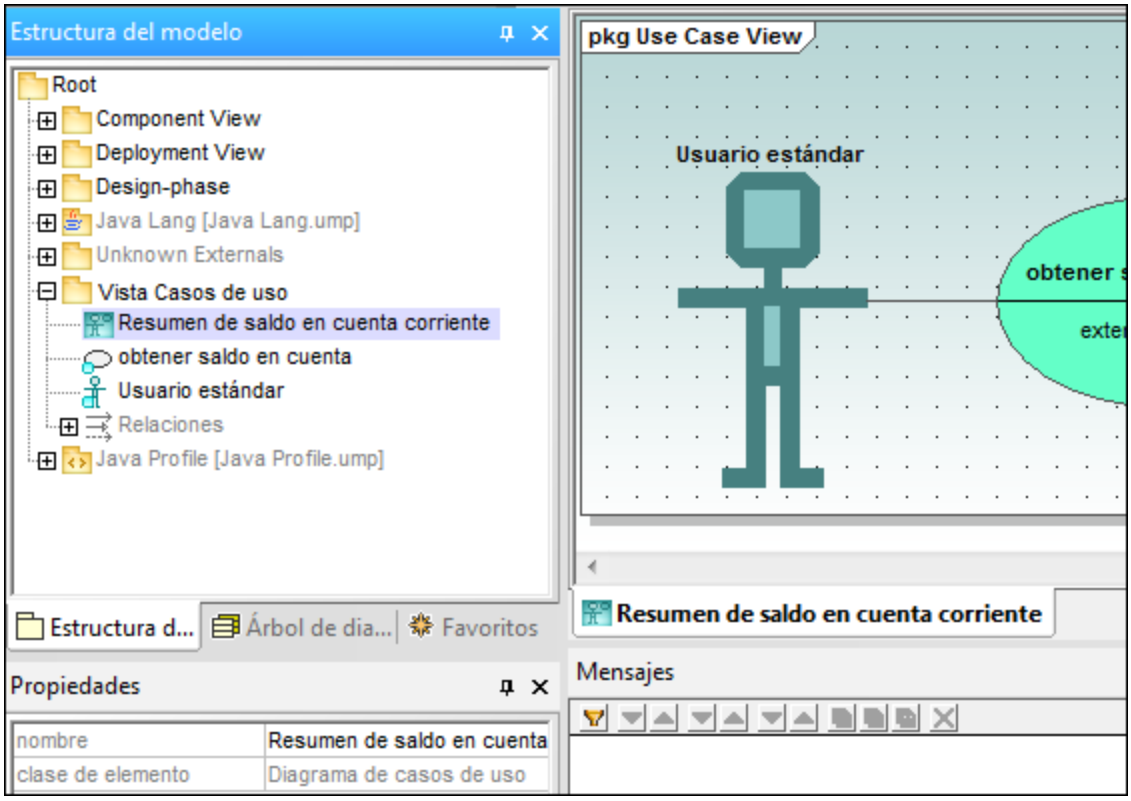
1. Usando el botón de la barra de herramientas que representa la relación que necesita (por ejemplo, asociación).
2. Usando los controladores que aparecen al hacer clic en cualquiera de los elementos del diagrama.

Crear relaciones usando los botones de la barra de herramientas

Cuando una ventana de diagrama está activa en el panel principal de UModel, la barra de herramientas muestra todos los elementos y las relaciones disponibles para ese diagrama. Por ejemplo, la barra de herramientas de un diagrama de clases dispone de botones para todas las relaciones aplicables, como asociación , asociación de colecciones , agregación , composición , realización , generalización , etc. Asimismo, la barra de herramientas de un diagrama de casos de uso dispone de botones de asociación , generalización , pero también de relaciones de Inclusión y Extensión .

A continuación explicamos cómo crear una relación de asociación entre un personaje y un caso de uso. Use el mismo enfoque para cualquier otra relación que quiera establecer.

1. Haga clic en un elemento del diagrama (personaje "Usuario estándar", en la imagen siguiente).
2. Haga clic en el botón de la barra de herramientas que corresponda a la relación que necesite (asociación , en este ejemplo).
3. Mueva el cursor sobre el "Usuario estándar" y arrástrelo hasta el elemento de destino ("obtener saldo de cuenta" en este caso). El elemento de destino está resaltado en color verde y solo acepta la relación si es significativa conforme a las especificaciones UML.



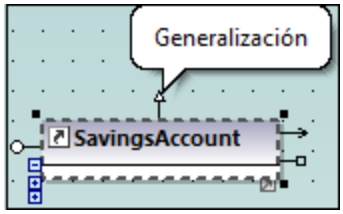
Asociación en un diagrama de casos de uso

Crear relaciones usando controles

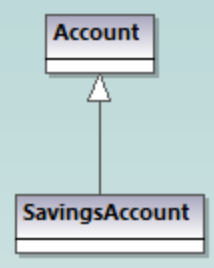
Al hacer clic en un elemento de un diagrama aparecen distintos controles encima, debajo, a la izquierda o a la derecha de ese elemento. Los controles solo aparecen para elementos que permiten relaciones. Cada control corresponde a un tipo de relación. Por ejemplo, los elementos de clase tienen los siguientes controles:

- RealizaciónDeInterfaz
- Generalización
- Asociación
- Asociación de colección

Para ver el tipo de relación que crea cada controlador, mueva el cursor sobre el control. Por ejemplo, en la imagen siguiente el controlador de encima del elemento se puede usar para crear una relación de generalización.



Para crear la relación haga clic en el controlador y arrastre el cursor hasta el elemento de destino, lo que crea la relación correspondiente (generalización, en este caso).



Relación de generalización entre dos clases

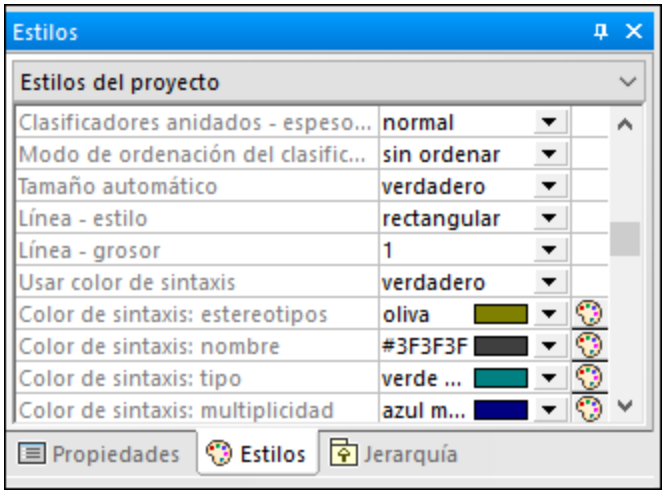
5.3.2 Cambiar el estilo de las líneas y relaciones

Puede modificar el grosor, color y tipo de curva de las líneas en la ventana *Estilos*. También puede añadir texto (etiquetas) a las relaciones, recolocar las etiquetas y mostrarlas en el diagrama u ocultarlas tanto individualmente para cada relación como en bloque.

Nota: es importante distinguir las "líneas" (cualquier línea de un diagrama) y "relaciones", como la de asociación, generalización, composición, etc. Todas las relaciones son líneas, pero no todas las líneas son relaciones. Por ejemplo, el enlace a un comentario o a una nota es simplemente una línea y no una relación.

Para cambiar las propiedades de una línea:

1. Haga clic en una línea en el diagrama.
2. En la ventana *Estilos*, indique la propiedad correspondiente (por ejemplo, "Línea - grosor").



Los valores disponibles para la propiedad "Línea - estilo" también están disponibles como comandos en el menú **Diseño | Estilo de la línea** y como botones de la barra de herramientas. Si modifica esta propiedad, se activará/desactivará el botón correspondiente de la barra de herramientas.

	**Línea ortogonal**	Una línea de este estilo solo se dobla en ángulos rectos.
	**Línea directa**	Una línea de este estilo crea una conexión directa entre dos elementos, sin ningún punto de paso.
	**Línea personalizada**	Una línea de este estilo se puede doblar en cualquier ángulo. Para mover la línea, arrastre uno de sus puntos de paso (pequeño cuadrado negro) en la dirección deseada. Para crear nuevos puntos de paso haga clic entre dos puntos de paso que ya existen y arrastre la línea hacia donde necesite. Para borrar un punto de paso, arrástrelo hasta situarlo justo encima de otro dentro de la misma línea.

Los estilos de línea, como cualquier otro estilo de elemento, se pueden aplicar a una sola línea o a un nivel más genérico (por ejemplo a nivel de proyecto). Uno estilo más específico invalida uno más genérico. Cuando un estilo es invalidado aparece un pequeño triángulo rojo en la esquina superior derecha de la propiedad invalidada, en la ventana *Estilos*. Consulte también la sección Cambiar el estilo de los elementos de un diagrama.

Para añadir etiquetas de texto a una relación:

- Haga clic en una relación del diagrama y comience a escribir.

Para mover la etiqueta de texto:

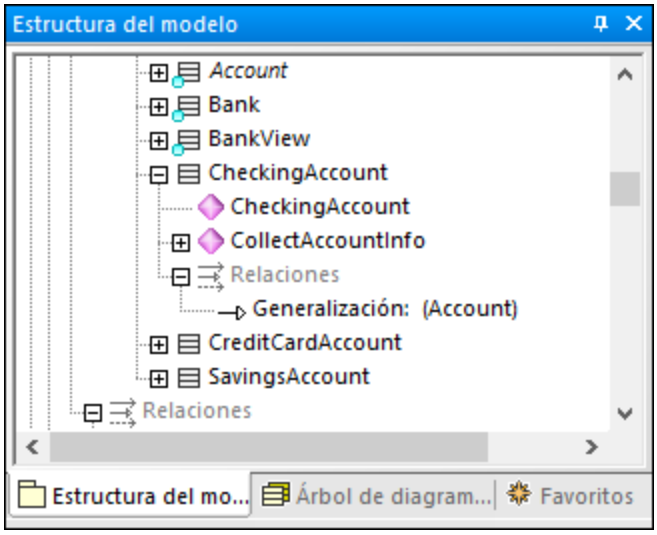
- Haga clic en la etiqueta y arrástrela a su nueva posición en el diagrama.
- Para devolver la etiqueta a su posición original haga clic con el botón derecho en la relación y seleccione **Etiquetas de texto | Ajustar la posición de las etiquetas de texto** en el menú contextual.
- Para recolocar varias etiquetas al mismo tiempo, seleccione una o más relaciones en el diagrama y ejecute el comando de menú **Diseño | Ajustar la posición de las etiquetas de texto**.

Para mostrar u ocultar etiquetas de texto:

- Haga clic con el botón derecho en la relación y seleccione **Etiquetas de texto | Mostrar todas las etiquetas de texto** o **Etiquetas de texto | Ocultar todas las etiquetas de texto**.

5.3.3 Ver las relaciones de los elementos

Las relaciones de un elemento se pueden ver en la ventana *Estructura del modelo* bajo ese elemento en concreto. Por ejemplo, en la imagen siguiente la clase `CheckingAccount` tiene una relación de generalización con la clase `Account`:



Relación vista en la ventana Estructura del modelo

Nota: para ocultar relaciones en la ventana *Estructura del modelo* haga clic con el botón derecho dentro de la ventana y desmarque la opción **Mostrar relaciones**.
Para mostrar las relaciones de un elemento del diagrama haga clic con el botón derecho en el elemento del diagrama y seleccione **Mostrar | <tipo de relación>** en el menú contextual.

5.3.4 Associations

Una asociación es una conexión conceptual entre dos elementos. Puede crear relaciones de asociación de la misma forma en que crearía cualquier otro tipo de relación en UModel (*véase* *Crear relaciones entre elementos*)

Cuando crea una asociación entre dos clases se inserta automáticamente un nuevo atributo en la clase de origen. Por ejemplo, al crear una asociación entre las clases `Coche` y `Motor` se añade una propiedad de tipo `Motor` a la clase `Coche`.

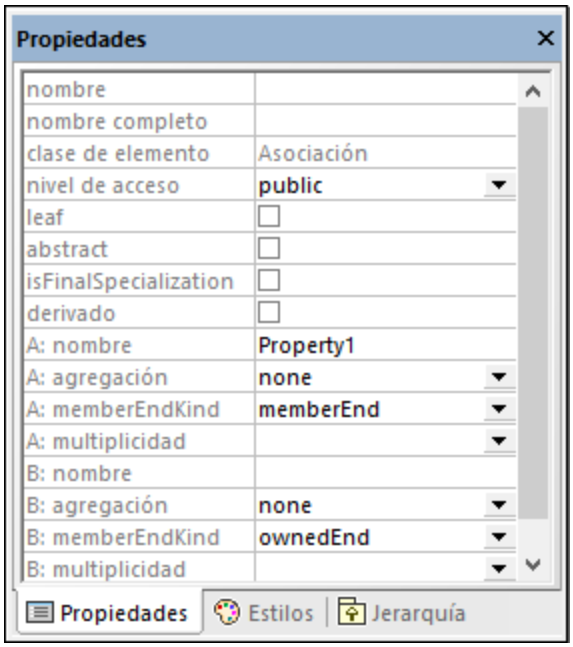
Cuando se añade una clase a un diagrama, sus asociaciones se muestran automáticamente en el diagrama, siempre que se cumplan las siguientes condiciones:

- La opción Crear asociaciones automáticamente se habilita desde **Herramientas | Opciones | Edición de diagramas**.
- Se determina el tipo de atributo (en la imagen anterior, `Propiedad1` es de tipo `Motor`)
- La clase del "tipo" referenciado también está presente en el diagrama actual (en la imagen anterior, la clase `Motor`).

También puede mostrar explícitamente las propiedades de cualquier clase como asociaciones en el diagrama. Para ello haga clic con el botón derecho en una propiedad de clase y seleccione uno de los siguientes comandos:

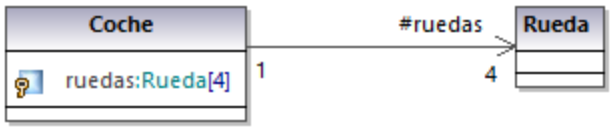
- **Mostrar | <Propiedad> como asociación**
- **Mostrar | Todas las propiedades como asociaciones**

Al hacer clic en una asociación del diagrama, sus propiedades se pueden modificar, si es necesario, desde la ventana Propiedades.



Preste atención a las siguientes propiedades. Al modificarlas, la asociación en el diagrama cambia el aspecto o añade varias etiquetas de texto informativas. Para obtener más información sobre mostrar o esconder etiquetas de texto, o sobre cambiar el aspecto de una relación (como el color o el grosor de la línea), véase Cambiar el estilo de las líneas y relaciones.

Propiedad	Finalidad
A: nombre	El nombre del miembro en el extremo A de la relación. En el ejemplo anterior es `Propiedad1`.
A: agregación	Permite cambiar el tipo de asociación en el extremo A. Al cambiar esta propiedad también cambia la representación de la relación en el diagrama. Los valores válidos son: **none** Indica una asociación normal **shared** Transforma la asociación en una agregación **composite** Transforma la asociación en una composición

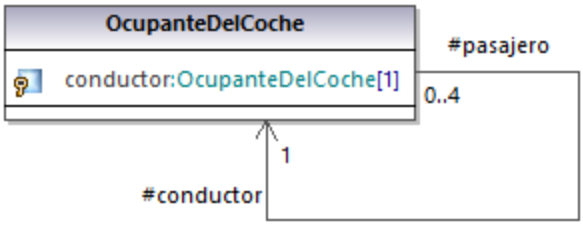
Propiedad	Finalidad
A: memberEndKind	Los atributos que participan en una relación pueden pertenecer a una clase o a la asociación. Esta propiedad especifica a quién pertenece este extremo de la relación y si es navegable (es decir, que la línea termina en una flecha). Los valores válidos son: **memberEnd** — El miembro en este extremo pertenece a la clase. **ownedEnd** — El miembro en este extremo pertenece a la asociación **navigableOwnedEnd** — El miembro en este extremo pertenece a la asociación y es navegable. Si establece ambos extremos como **ownedEnd**, la asociación se convierte en bidireccional.
A: multiplicidad	La multiplicidad indica el número de objetos en este extremo de la relación. Por ejemplo, si un coche tiene cuatro ruedas, la multiplicidad se indicaría con un 1 en un extremo de la relación y un 4 en el otro. 

Existen los mismos atributos disponibles para el extremo B de la relación.

Al activar la propiedad **Mostrar pto. de propiedad de la asoc.** en la ventana *Estilos*, esta muestra con puntos la propiedad de la relación seleccionada. El valor predeterminado de esta propiedad es **False**. En el ejemplo de la imagen siguiente la propiedad **Mostrar pto. de propiedad de la asoc.** de la clase se ha cambiado a **True**:

Crear asociaciones reflexivas

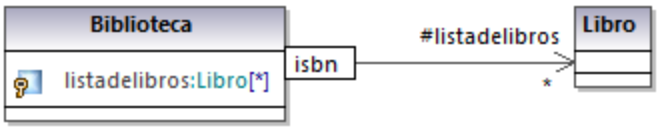
Las asociaciones se pueden crear usando la misma clase para origen y destino. A esto se le llama asociación reflexiva o recursiva. Este tipo de asociación describe, por ejemplo, la habilidad de un objeto para enviarse un mensaje a sí mismo, es decir, para hacer llamadas recursivas. Para crear este tipo de enlace haga clic en el botón de asociación de la barra de herramientas y arrastre la línea de vuelta al mismo elemento.



Crear calificadores de asociaciones

Las asociaciones se pueden completar con calificadores de asociaciones. Los calificadores son los atributos de una asociación. En el ejemplo siguiente, el calificador `isbn` indica que se puede recuperar un libro de la lista de libros con este atributo. Para añadir un calificador:

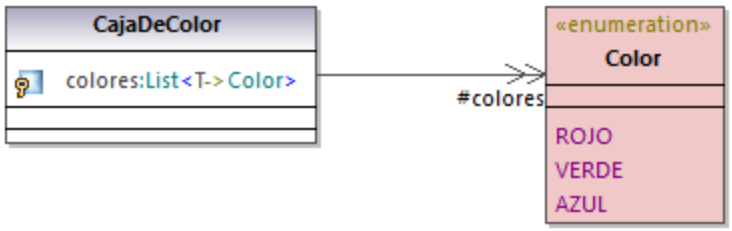
1. Cree una asociación entre dos clases.
2. Haga clic con el botón derecho en la asociación y seleccione **Nuevo/a | Calificador**.



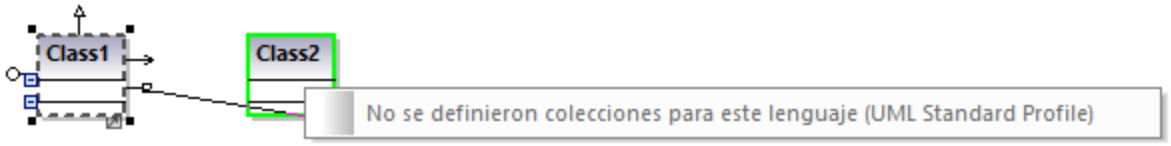
Para renombrar o eliminar los calificadores de una asociación, siga los mismos pasos que para el resto de elementos (véanse Renombrar, mover y copiar elementos and Borrar elementos).

5.3.5 Asociación de colecciones

Una asociación de colecciones sirve para indicar que una propiedad de una clase es una colección de algún tipo. Por ejemplo, en el siguiente diagrama la propiedad `colores` de la clase `CajaDeColor` es una lista de colores. En este caso esa propiedad se expresa como una enumeración, pero en otros casos podría ser otra clase o incluso una interfaz.



Para que pueda crear asociaciones de colecciones, el proyecto de UModel primero tiene que contener las plantillas de colección para el lenguaje de proyecto que quiera usar (como Java, C#, o VB.NET). De lo contrario aparecerá el texto "No se definieron colecciones para este lenguaje" cuando intente crear este tipo de asociación.

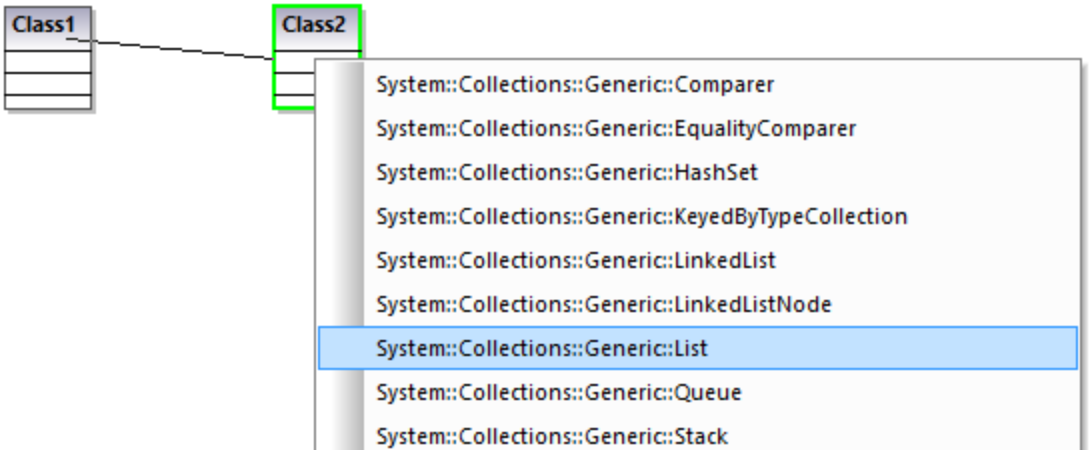


Si su proyecto utiliza únicamente UML (sin ningún otro lenguaje de ingeniería de código), puede definir las plantillas de colección en **Herramientas | Opciones | Edición de diagramas | Plantillas de colecciones | UML**.

Si su proyecto ya contiene un espacio de nombres de un lenguaje (como Java, C#, o VB.NET), las plantillas de colecciones estarán predefinidas desde el perfil de ese lenguaje. Se pueden añadir más plantillas desde **Herramientas | Opciones | Edición de diagramas | Plantillas de colecciones**.

Para crear una asociación de colecciones (por ejemplo entre dos clase):

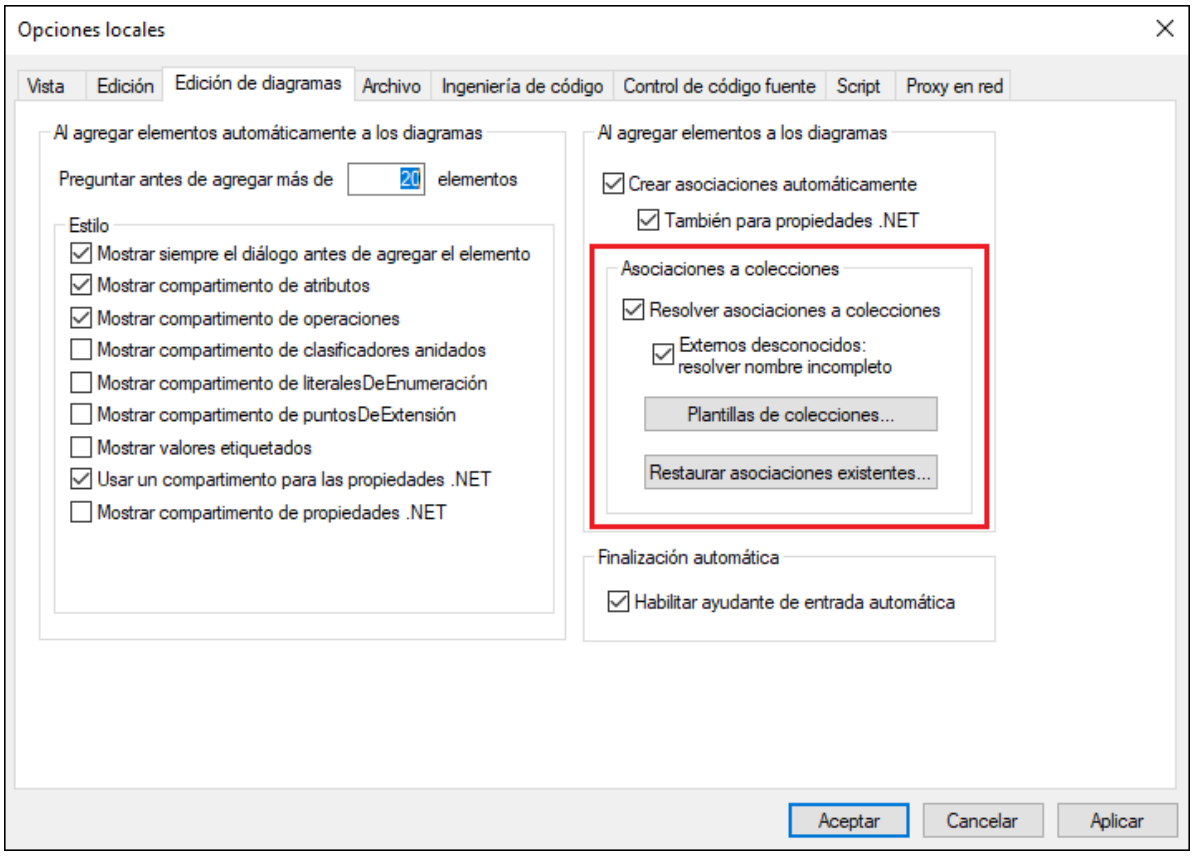
1. Añada dos clases al diagrama.
2. Haga clic en el botón de la barra de herramientas **Asociación de colección** .
3. Arrastre la primera clase hasta la segunda. Aparecerá un menú contextual con las plantillas de colecciones definidas para el proyecto. Seleccione la que necesite.



Asociación de colecciones e ingeniería de código

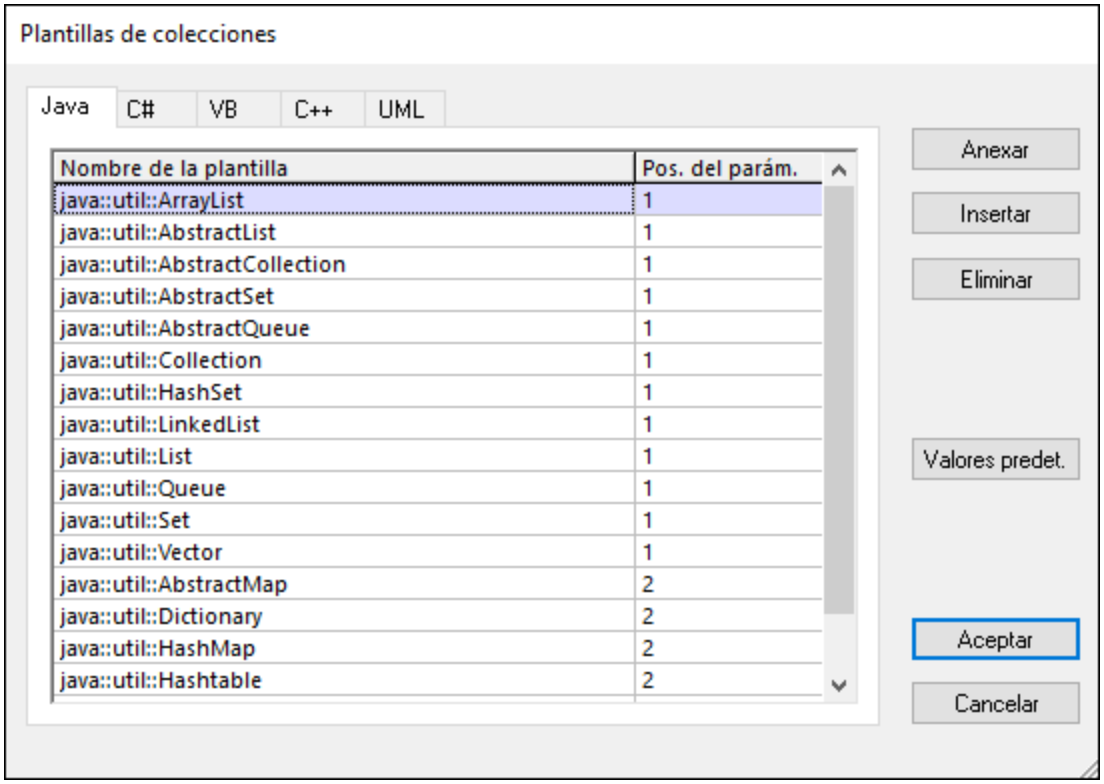
Si importa código de programa en el modelo, las asociaciones de colecciones se crearán automáticamente por defecto y se basarán en plantillas de colecciones predefinidas. Para activar o desactivar esta opción:

1. En el menú **Herramientas** haga clic en **Opciones**.
2. Haga clic en la pestaña **Edición de diagramas**.
3. Marque o desmarque, según lo que necesite, la casilla **Resolver asociaciones a colecciones**.



Por defecto, las asociaciones de colecciones se resuelven basándose en una lista de plantillas de colecciones integradas. Para ver o modificar las plantillas de colecciones integradas haga clic en **Plantillas de colecciones**.

Para insertar tipos de colecciones personalizados, use los botones **Anexar**, **Insertar** o **Eliminar** que encontrará en el siguiente cuadro de diálogo. La columna **Pos. del parám.** indica la posición del parámetro que contiene el valor tipo de la colección.



Cuadro de diálogo de las plantillas de colecciones

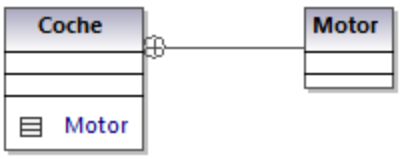
Para restaurar las plantillas de colecciones a sus valores originales haga clic en **Valores predet**.

5.3.6 Contención

Las líneas de contención se usan, por ejemplo, para mostrar relaciones primario-secundario entre dos clases o dos paquetes.

Para ilustrar contención entre dos clases:

1. Haga clic en el botón de la barra de herramientas **Contención** (en un diagrama de clases o de paquetes).
2. Arrastre la línea desde la clase que debe "ser contenida" hasta la clase que la contendrá.



La clase contenida, `Motor` en este caso, ahora es visible en el compartimento `Coche`. Esto también sitúa la clase contenida en el mismo espacio de nombres que la clase contenedora.

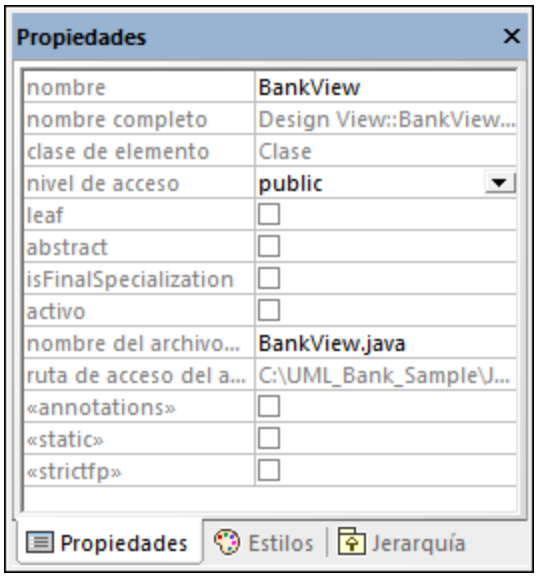
5.4 Estereotipos y valores etiquetados

Un estereotipo es un mecanismo de extensión que permite expandir de forma flexible un elemento UML que ya existe y capturar algún aspecto del mismo que al UML estándar se le escapa. El que se hayan aplicado estereotipos a un elemento implica que ese elemento tiene algún uso especial. Los perfiles integrados de UModel (C#, Java, VB.NET, etc.) contienen todos los estereotipos necesarios para modelar proyectos en los lenguajes correspondientes. Sin embargo, también puede crear sus propios perfiles y sus respectivos estereotipos (véase Crear y aplicar perfiles personalizados).

Al importar código fuente o archivos binarios en un modelo, UModel aplica estereotipos a elementos de forma automática basándose en la estructura del código original. Por ejemplo, si existen modificadores de anotaciones en el código fuente Java importado, los elementos correspondientes reciben el estereotipo «annotations». Para más información acerca de cómo las distintas construcciones de los lenguajes se corresponden con los elementos de UModel y se convierten en estereotipos en el modelo (véase Correspondencias con elementos de UModel).

También puede aplicar estereotipos a los elementos de forma manual mientras los modela. Por ejemplo, puede aplicar el estereotipo «attributes» a una clase C#, lo que indicaría que esa clase debe llevar atributos en el código generado. Para especificar los valores de los atributos en el código generado puede añadir los llamados "valores etiquetados" en UModel, como se muestra en Aplicar estereotipos. Los estereotipos también se usan ampliamente en el modelado de esquemas XML para modelar elementos como tipos simples, complejos, facetas, etc. Asimismo, los estereotipos se usan en el modelado de bases de datos para modelar elementos como tablas, indices, etc. (*Véase Diseñar objetos de base de datos*.)

En la interfaz gráfica de UModel los estereotipos se muestran entre comillas angulares (por ejemplo, «static»). Todos los estereotipos incluidos en los perfiles integrados de UModel aparecen en la ventana *Propiedades* al hacer clic en un elemento. Por ejemplo, si hace clic en una clase Java de la Estructura del modelo, en la ventana *Propiedades* aparecen únicamente los estereotipos de clase que se pueden aplicar al perfil Java (en este ejemplo serían «annotations», «static», «strictfp»).

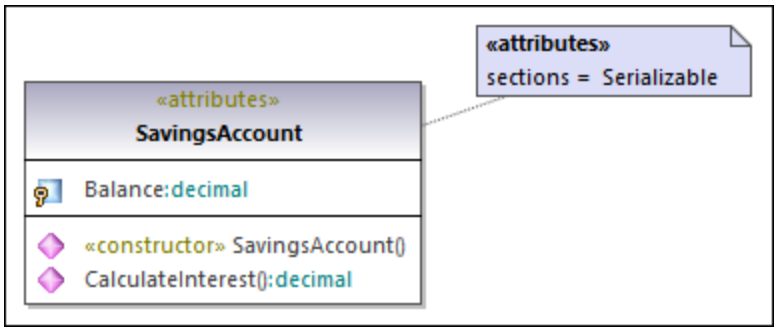


En los diagramas de clases, los estereotipos aparecen encima del nombre de la clase. Por ejemplo, en la imagen siguiente a la clase se le ha aplicado el estereotipo «attributes».

En el caso de los métodos o las propiedades, los estereotipos se muestran en línea (inline), como el estereotipo que se ha aplicado al método **Account()** de la clase de la imagen anterior.

5.4.1 Valores etiquetados

Los estereotipos pueden tener atributos (valores etiquetados) asociados a ellos. Los valores etiquetados son pares nombre-valor que proporcionan información adicional relacionada con el estereotipo al que pertenecen. Por ejemplo, a la clase de la imagen siguiente se le ha aplicado el estereotipo `«attributes»`. Observer que el estereotipo `«attributes»` tiene valores etiquetados asociados: una clave (nombre) llamada "sections" y un valor llamado "Serializable".



Valores etiquetados

Un estereotipo puede tener varios pares de valores etiquetados. Además, un valor se puede seleccionar de un conjunto de valores de enumeración.

Puede cambiar la forma en que los valores etiquetados se muestran en el diagrama u ocultarlos directamente (*véase* *Mostrar u ocultar valores etiquetados*). Para más información sobre cómo cambiar los valores etiquetados de un estereotipo consulte el apartado Aplicar estereotipos. Para ver un ejemplo que ilustre cómo crear estereotipos con valores etiquetados consulte el apartado Ejemplo: crear y aplicar estereotipos.

5.4.2 Aplicar estereotipos

Al aplicar un estereotipo a un elemento está indicando que el elemento tiene un uso específico. En el caso de los lenguajes de programación compatibles con UModel (com C#, VB.NET o Java) se suelen aplicar estereotipos para adecuarse a la gramática de ese lenguaje en cuestión. Por ejemplo, se puede aplicar el estereotipo «static» a una clase Java.

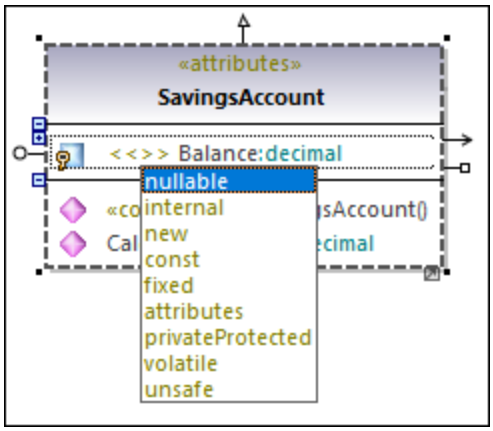
Antes de poder aplicar estereotipos hay que aplicar a los paquetes el perfil correspondiente. Esto lo hace automáticamente UModel si hace clic con el botón derecho en un paquete y selecciona el comando **Ingeniería de código | Establecer como raíz de espacio de nombres de {lenguaje}**. Para más información consulte Aplicar perfiles de UModel.

Si creó perfiles personalizados debe aplicarlos al paquete de forma manual (*véase Crear y aplicar perfiles personalizados*).

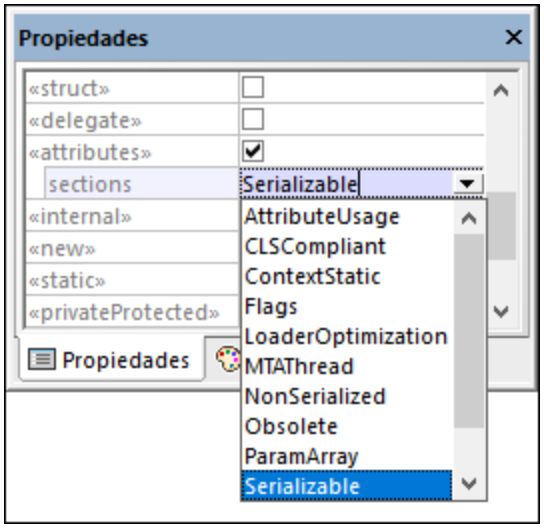
Para aplicar un estereotipo a un elemento:

1. Haga clic en el elemento en la ventana *Estructura del modelo*. Si el elemento se puede ampliar con estereotipos, estos aparecen como propiedades en la ventana *Propiedades* entre comillas angulares ("«" y "»").
2. Marque la casilla del estereotipo en la ventana *Propiedades* (por ejemplo, «static»).

También puede aplicar estereotipos mientras diseña elementos dentro de un diagrama de clases. Para ello, haga clic en una propiedad de una clase y comience a escribir el texto dentro de los caracteres "<< y ">>".

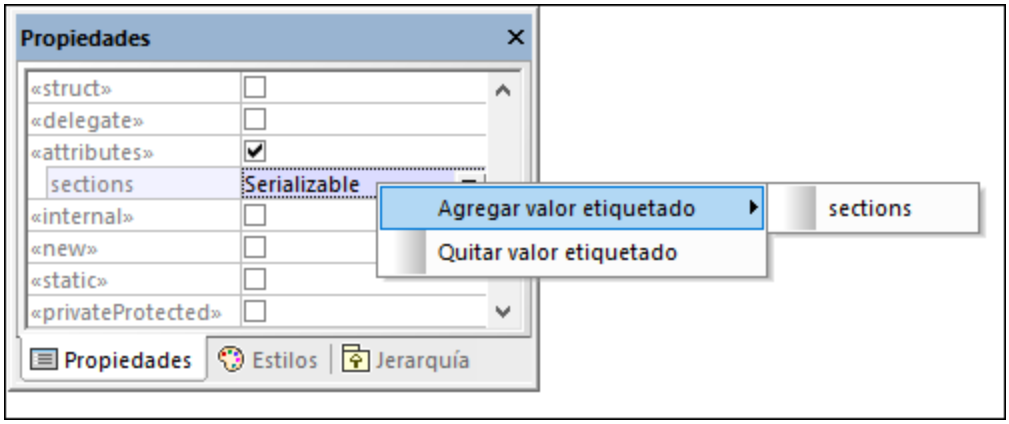


Algunos estereotipos están asociados a una lista de pares nombre-valor a los que en lenguaje UML se llama "valores etiquetados". Para aplicar un estereotipo con valores etiquetados a un elemento, marque la casilla del estereotipo en la ventana *Propiedades* (en este ejemplo, «attributes»), lo que añade una entrada indentada en la que puede seleccionar el valor deseado de un lista predefinido.

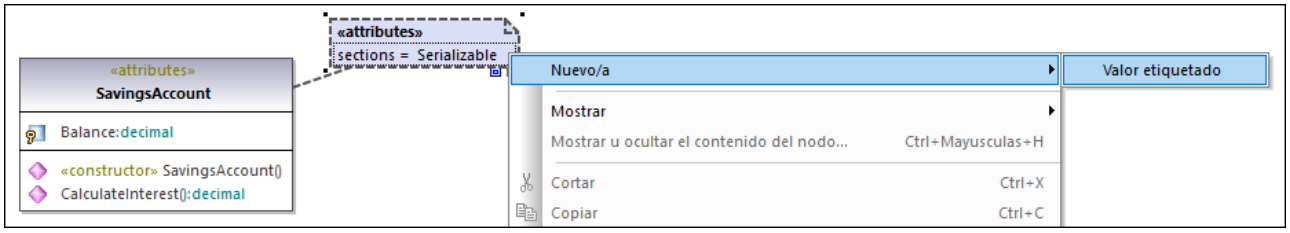


Valores etiquetados

También puede añadir varios valores a la misma clave. Para ello, haga clic con el botón derecho en esta entrada indentada y seleccione **Agregar valor etiquetado | <nombre>** del menú contextual.

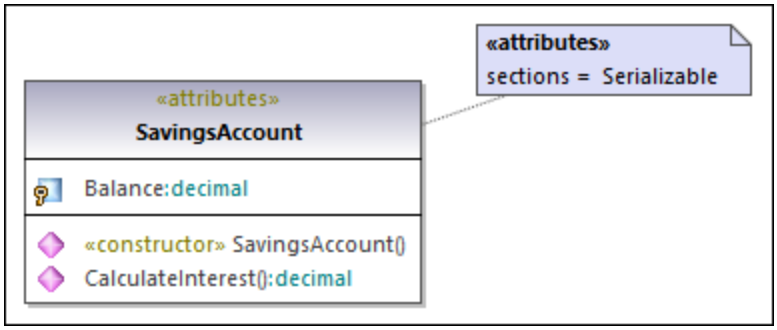


Otra alternativa es añadir valores etiquetados directamente desde el diagrama haciendo clic con el botón derecho en un valor y seleccionando **Nuevo | Valor etiquetado** del menú contextual.



5.4.3 Mostrar u ocultar valores etiquetados

Cuando un elemento tiene valores etiquetados puede verlos todos bien en un cuadro aparte o en un compartimento dentro del mismo elemento. También puede ocultar por completo los valores etiquetados. Para elegir cómo quiere que se muestren los valores etiquetados haga clic con el botón derecho en el elemento relevante del diagrama y seleccione **Valores etiquetados | <display option>**. Por ejemplo, para mostrar todos los valores etiquetados fuera de la clase haga clic con el botón derecho en la clase del diagrama y seleccione **Valores etiquetados | todos**. Para ocultar todos los valores etiquetados de una clase haga clic con el botón derecho en la clase del diagrama y seleccione **Valores etiquetados | ninguno**.

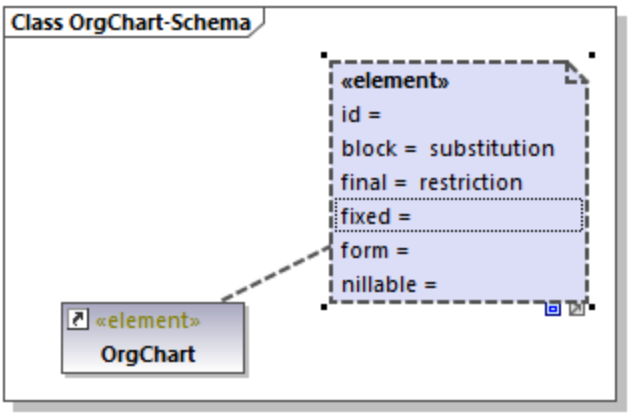


Valores etiquetados fuera de una clase

Activar el modo compacto

Si en un cuadro de valores etiquetados hay valores vacíos puede optar por ocultarlos:

1. Seleccione un cuadro de valores etiquetados en el diagrama (uno que tenga valores pero también valores vacíos).



2. Haga clic en Activar/Desactivar el modo compacto, en la esquina inferior derecha del cuadro.

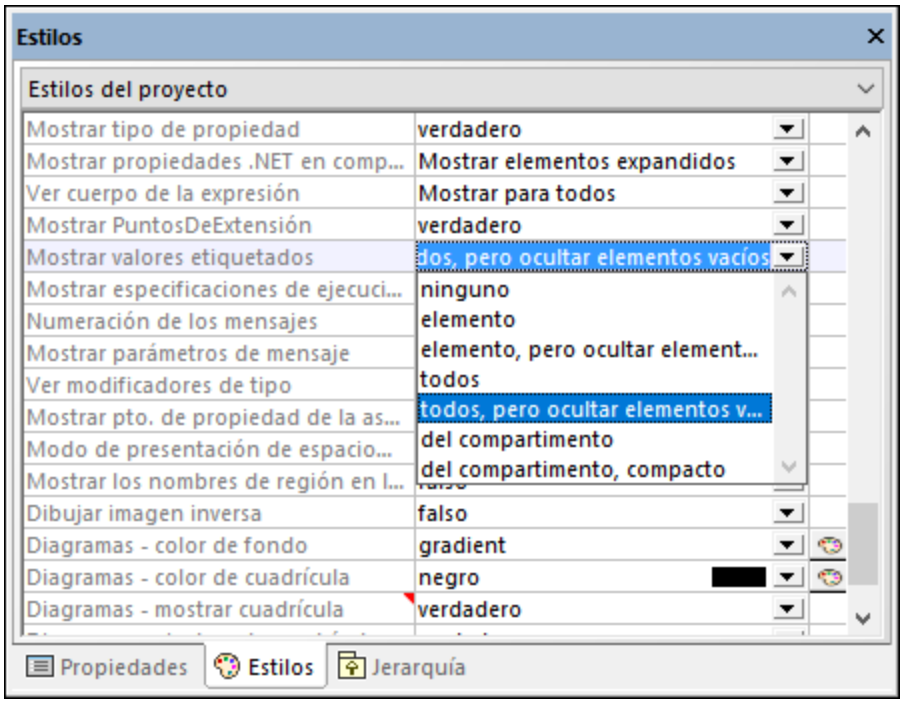
Cuando en el icono el cuadrito aparece expandido, se muestran los valores vacíos. Si está en modo compacto, los valores vacíos permanecen ocultos.

Cambiar la visualización de los valores etiquetados a nivel global

Puede decidir qué valores etiquetados se muestran de forma individual para cada elemento, como acabamos de explicar, o a nivel global para todo el proyecto.

Para cambiar los valores etiquetados a nivel del proyecto:

1. Seleccione **Estilos del proyecto** de la lista que hay en la parte superior de la ventana Estilos.
2. Baje hasta la propiedad `Mostrar valores etiquetados` y seleccione la opción deseada de la lista (por ejemplo **todos, pero ocultar elementos vacíos**).

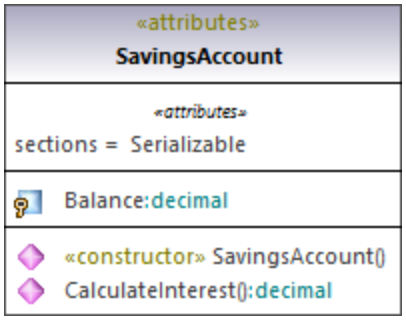


Para obtener más información sobre cómo cambiar los estilos a distintos niveles consulte Cambiar el estilo de los elementos de un diagrama.

Opciones de visualización

En la tabla siguiente puede ver las opciones de visualización de los valores etiquetados. Estas opciones son parecidas tanto si los cambia a nivel global como si lo hace para elementos individuales.

Ninguno	Oculta todos los valores etiquetados.
Todos	Muestra los valores etiquetados de un elemento (por ejemplo, de una clase), además de los elementos que pertenecen a esa clase, como atributos y operaciones.
Todos, pero ocultar elementos vacíos	Muestra solamente los valores etiquetados que tienen un valor.

Elemento	Muestra los valores etiquetados de un elemento (por ejemplo, de una clase) pero no los de atributos, operaciones, etc.
Elemento, pero ocultar elementos vacíos	Muestra los valores etiquetados de un solo elemento que contengan un valor.
En compartimento	Muestra todos los valores etiquetados en un compartimento separado. Por ejemplo, la clase de la imagen siguiente tiene el compartimento *«attributes»*, que contiene valores etiquetados. 
En compartimento, pero ocultar elementos vacíos	Muestra en un compartimento solamente los valores etiquetados que tengan un valor.
Del compartimento, compacto	Igual que el punto anterior.

6 Proyectos e ingeniería de código

Esta sección explica cómo crear proyectos de modelado en UModel desde cero o importando datos desde código fuente o archivos binarios. También describe varias operaciones relacionadas con la ingeniería de código:

- ingeniería directa (generar código a partir de un proyecto de UModel),
- ingeniería inversa (importar código fuente a un proyecto de UModel) e
- ingeniería de ida y vuelta (es decir, sincronizar el modelo y el código en ambos sentidos cuando sea necesario).

Los comandos de menú relacionados con la función de ingeniería de código están en el menú **Proyecto**. Por ejemplo, el comando **Proyecto | Importar proyecto de código fuente** permite importar soluciones C#, C++, VB.NET Visual Studio o código Java y generar diagramas de UModel a partir de ellos. Si no dispone de ninguna solución de proyecto, use el comando **Proyecto | Importar directorio de código fuente** (*véase Importar código fuente*). También pueden importarse binarios Java, C# y VB.NET siempre y cuando se cumplan ciertos requisitos básicos (*véase Importar archivos binarios Java, C# y VB*).

Las operaciones de ingeniería de código recién mencionadas no solamente pueden llevarse a cabo con lenguajes de programación, sino también con bases de datos y documentos XML Schema. Por ejemplo, con el comando **Proyecto | Importar archivo de esquema XML** puede aplicar ingeniería inversa a un esquema XML y generar automáticamente un diagrama de clases basado en dicho esquema.

Para ver las correspondencias entre elementos de UModel y elementos de cada perfil de lenguaje compatible (bases de datos y XML Schema incluido consulte Correspondencias con elementos de UModel. Si necesita información sobre bases de datos y conectividad consulte Trabajar con bases de datos en UModel.

6.1 Administrar proyectos de UModel

Los proyectos de UModel hacen las veces de contenedores para los elementos y diagramas UML y para las opciones de configuración. Los proyectos de UModel tienen la extensión de archivo .ump (archivo de proyecto de UModel).

En UModel no es necesario seguir ninguna secuencia de modelado concreta. Puede agregar cualquier tipo de elemento de modelado al proyecto (diagramas, paquetes, actores, etc.) en cualquier orden y en cualquier posición. Todos los elementos de modelado se pueden insertar, renombrar y eliminar en la ventana *Estructura del modelo*.

6.1.1 Crear, abrir y guardar proyectos

Cuando se inicia UModel por primera vez, se abre un proyecto nuevo automáticamente. A partir de ese momento, cada vez que inicie UModel, se abrirá el último proyecto en el que haya trabajado.

Nota: UModel incluye varios proyectos que puede usar para familiarizarse con los conceptos básicos de modelado y con la interfaz gráfica del usuario. Los encontrará aquí: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**.

Para crear un proyecto nuevo:

- En el menú **Archivo** haga clic en **Nuevo** (o en el botón **Nuevo** de la barra de herramientas).

Esto crea un proyecto nuevo con el nombre predeterminado ProyectoNuevo1. A ese proyecto se le añaden automáticamente dos paquetes (visibles en la ventana *Estructura del modelo*):

- `Root`
- `Component View`

Estos dos paquetes son especiales y son los únicos que no se pueden renombrar ni eliminar, tal y como se explica en el apartado del tutorial *Ingeniería directa (del modelo al código)*.

Una vez creado el proyecto podrá añadirle elementos de modelado, como paquetes y diagramas UML. En la ventana *Estructura del modelo*, las entradas con el símbolo de carpeta  representan paquetes UML. Los paquetes son contenedores donde se almacenan los demás elementos de modelado UML, como diagramas de casos de uso, clases, instancias, etc. Los paquetes funcionan de la siguiente manera:

- puede crear paquetes en cualquier posición de la *Estructura del modelo*.
- puede mover y copiar paquetes y su contenido a otros paquetes de la *Estructura del modelo* (y también a otros diagramas de modelado de la ventana de diagramas.
- puede ordenar los paquetes y su contenido en la ventana *Estructura del modelo*, siguiendo diferentes criterios de ordenación.
- puede colocar paquetes dentro de otros paquetes.
- puede usar paquetes como elementos de origen o destino cuando combine o sincronice código.

Para agregar un paquete nuevo:

1. Haga clic con el botón derecho en el paquete donde desea insertar un paquete nuevo (en proyectos nuevos será el paquete `Root` o `Component View`).
2. Seleccione **Elemento nuevo | Paquete** en el menú contextual.

Debe tener en cuenta que los paquetes, al igual que otros elementos de modelado, también se pueden añadir desde diagramas UML. Si se añaden desde diagramas UML, los paquetes aparecerán automáticamente en la ventana *Estructura del modelo*.

Para agregar un diagrama nuevo:

- Haga clic con el botón derecho en la *Estructura del modelo* y seleccione **Diagrama nuevo**.

Para agregar elementos a un diagrama:

- Hay dos maneras de agregar elementos a un diagrama:
  - Haciendo clic con el botón derecho en el diagrama y seleccionando **Nuevo/a | <Tipo de elemento>** en el menú contextual.
  - Seleccionando el elemento correspondiente en la barra de herramientas y haciendo clic dentro del diagrama.

Para ver un ejemplo consulte el apartado *Ingeniería directa (del modelo al código)*, que explica cómo crear un proyecto nuevo y generar código de programa a partir de él.

Para abrir un proyecto que ya existe:

- En el menú **Archivo** haga clic en **Abrir** y navegue hasta el archivo de proyecto .ump.

Nota: UModel registra por defecto todos los cambios realizados de forma externa en el archivo de proyecto .ump o en los archivos que incluye y emite un aviso pidiendo que recargue del proyecto. Este comportamiento puede deshabilitarse en **Herramientas | Opciones | Archivo**.

Para guardar un proyecto:

- En el menú **Archivo** haga clic en **Guardar** (o en **Guardar como**).

Todos los datos relacionados con el proyecto se almacenan en el archivo de proyecto de UModel, que tiene la extensión de archivo *.**ump** (archivo de proyecto de UModel).

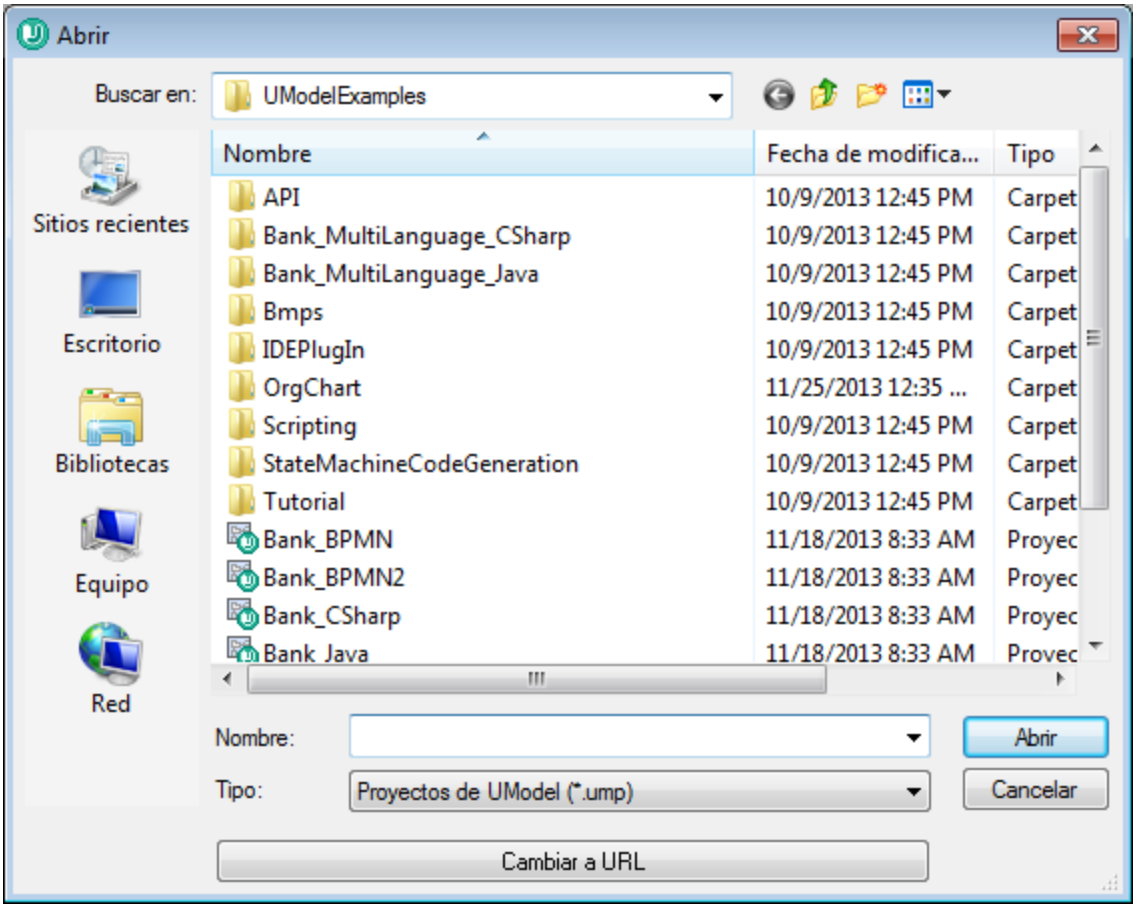
Nota: el formato de archivo ***.ump** es un formato de archivo XML al que se le puede aplicar la opción de formato mejorado *pretty-print* cuando se guarda (para habilitarla vaya a **Herramientas | Opciones | Archivo**).

6.1.2 Abrir proyectos desde una URL

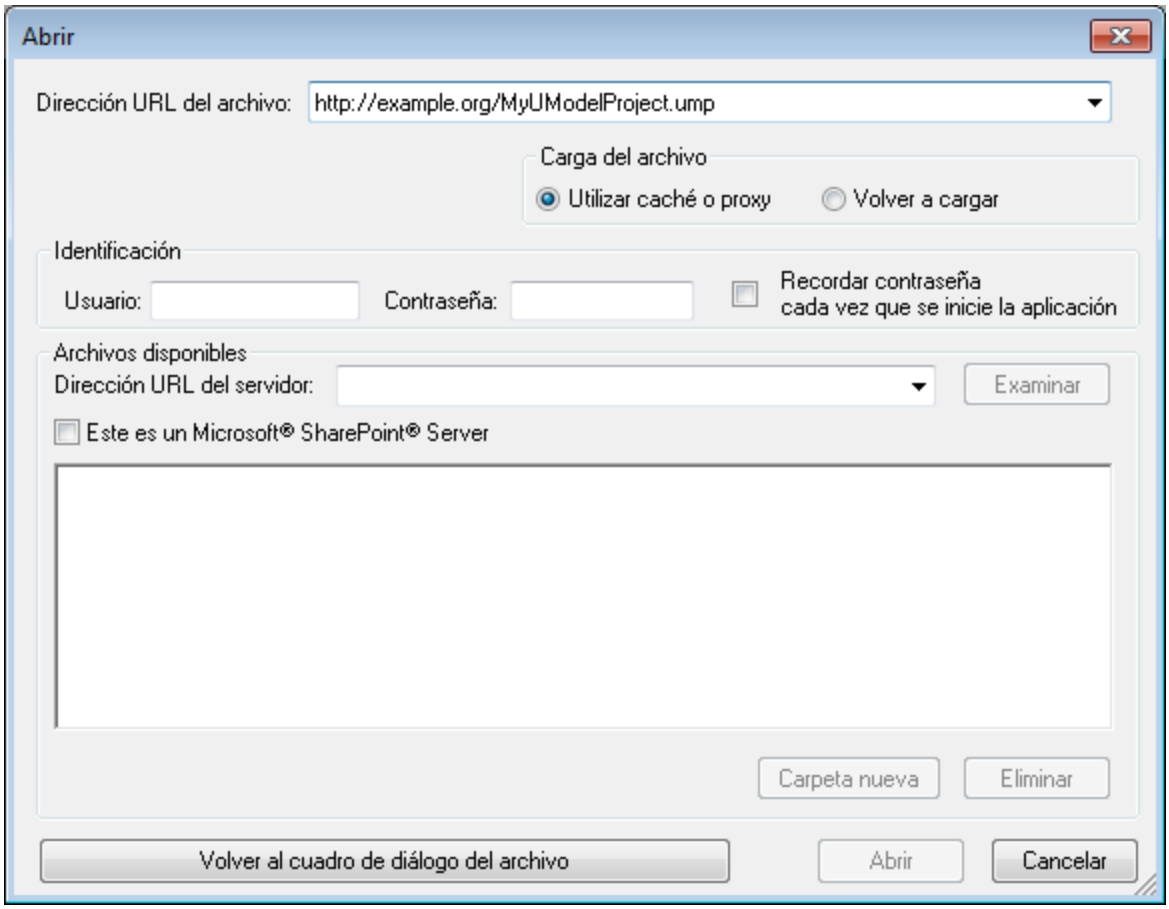
Además de abrir archivos de proyecto locales, también puede abrir archivos desde direcciones URL. Los protocolos compatibles son HTTP, HTTPS y FTP. No obstante, recuerde que los archivos que se cargaran desde direcciones URL no se pueden volver a guardar en su ubicación original (en otras palabras, el acceso al archivo es de solo lectura) a no ser que se desprotegieran desde un servidor Microsoft® SharePoint® Server (ver ejemplos más abajo).

Para abrir un archivo desde una dirección URL:

1. En el cuadro de diálogo "Abrir" haga clic en el botón **Cambiar a URL**.



2. Introduzca la URL del archivo en el cuadro de texto *URL del archivo* y después haga clic en **Abrir**.



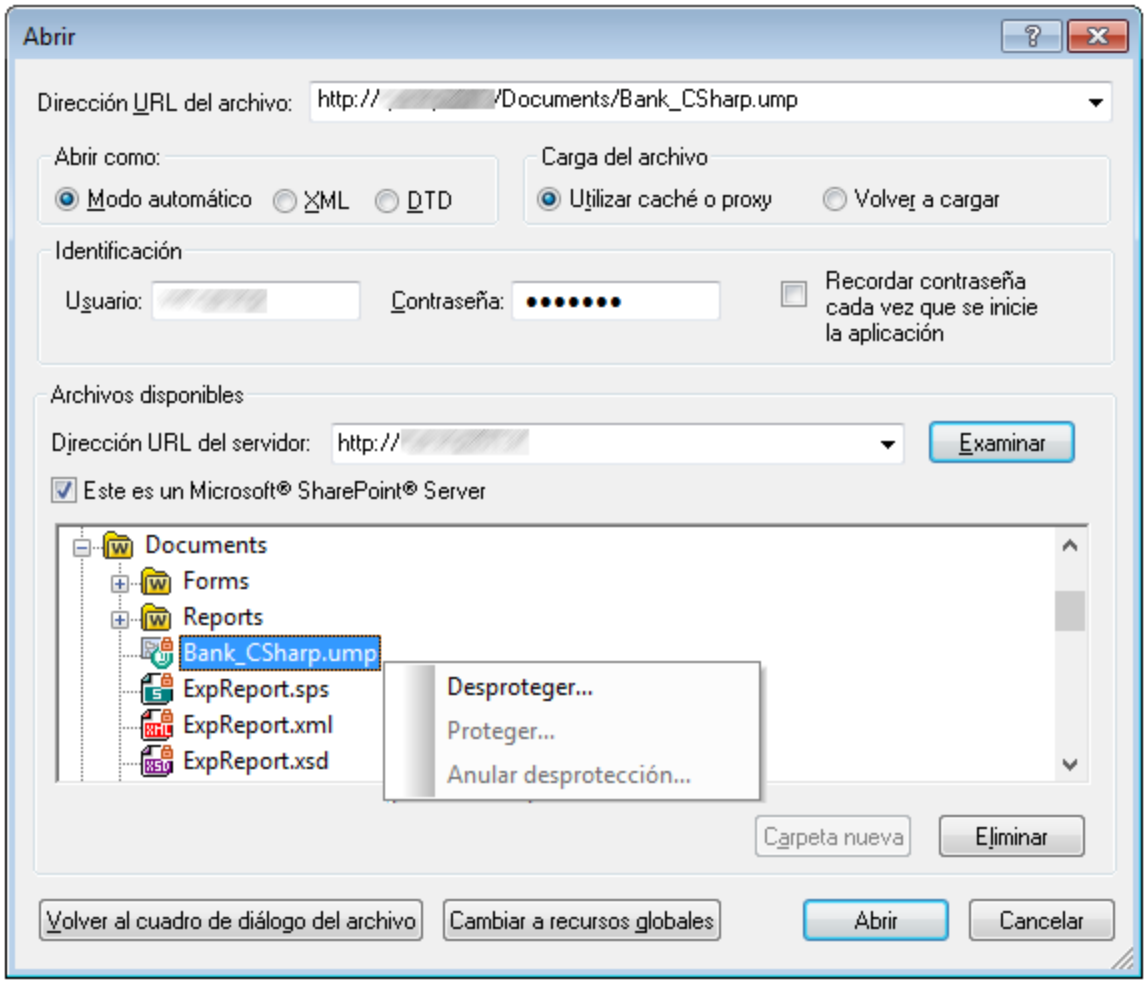
Si el servidor solicita la autenticación con contraseña, deberá introducir el nombre de usuario y la contraseña. Si desea que la aplicación recuerde el nombre de usuario y la contraseña la próxima vez que se inicie, introduzca estos datos en el cuadro de diálogo "Abrir" y marque la casilla *Recordar contraseña al cerrar aplicación*.

Si es improbable que el archivo elegido cambie en el futuro, seleccione la opción *Utilizar caché o proxy* para almacenar los datos en caché y acelerar la carga del archivo. De lo contrario, si desea que el archivo se vuelva a cargar cada vez que se abra UModel, entonces seleccione *Volver a cargar*.

En el caso de servidores compatibles con WebDAV (Sistema distribuido de creación y control de versiones web), podrá examinar archivos después de introducir la URL del servidor en el cuadro de texto *URL del servidor* y de hacer clic en **Examinar**.

Nota: la función **Examinar** solamente está disponible en servidores compatibles con WebDAV y en servidores Microsoft SharePoint.

Si se trata de un servidor Microsoft® SharePoint® Server, marque la casilla *Este es un Microsoft® SharePoint® Server*. Esto permite ver el estado de protección o desprotección del archivo en el panel de vista previa.



El archivo puede tener 3 estados:

	Protegido. El archivo se puede desproteger.
	Desprotegido por otro usuario. No se puede desproteger.
	Desprotegido localmente. Se puede editar y proteger.

Para poder modificar el archivo en UModel debe hacer clic con el botón derecho en el archivo y seleccionar **Desproteger** en el menú contextual. Cuando un archivo se desprotege en el servidor Microsoft® SharePoint® y se guarda en UModel, los cambios se envían al servidor. Para volver a proteger el archivo en el servidor debe hacer clic con el botón derecho en el archivo y elegir **Proteger** en el menú contextual (también puede iniciar sesión en el servidor y realizar la misma operación desde el explorador). Para descartar los cambios realizados en el archivo desde que se desprotegió basta con hacer clic con el botón derecho en el archivo y elegir **Deshacer desprotección** en el menú contextual (la misma operación puede llevarse a cabo desde el explorador).

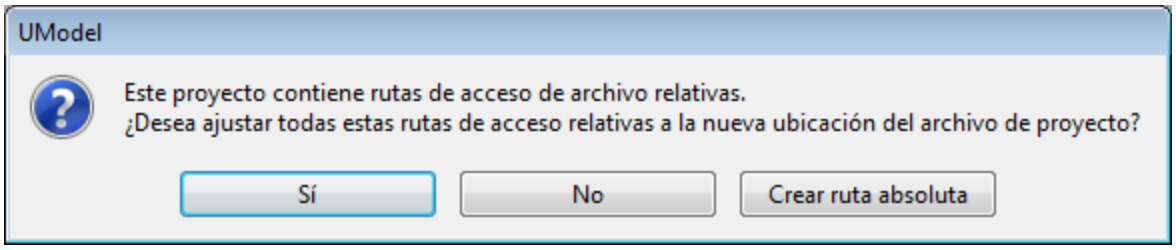
Es importante tener en cuenta que:

- Si un archivo ya está desprotegido por otro usuario, no podrá desprotegerlo.
- Si desprotege un archivo en una aplicación de Altova, no podrá desprotegerlo desde ninguna otra aplicación de Altova porque se considera desprotegido.

6.1.3 Mover proyectos a un directorio nuevo

Los proyectos de UModel y el código generado se pueden mover con facilidad a otro directorio (o a otro equipo) y desde allí se pueden volver a sincronizar. Hay dos maneras de hacer esto:

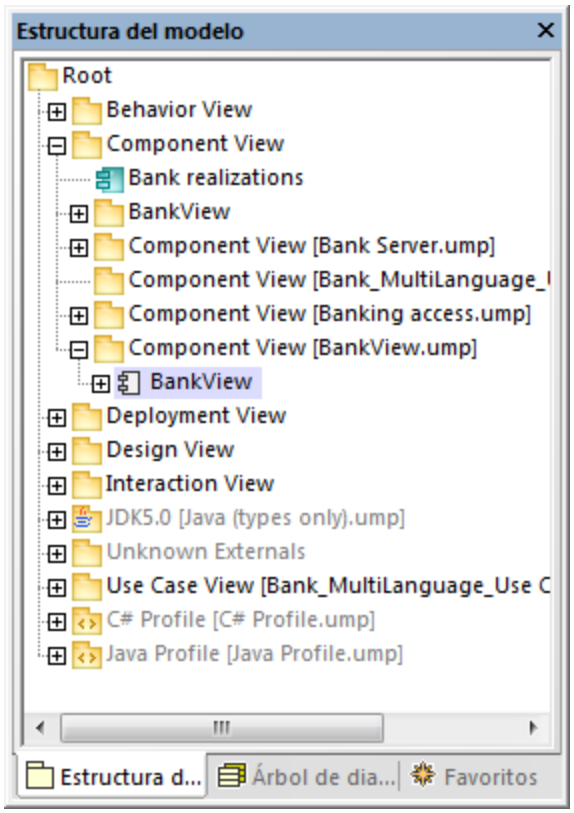
- Seleccionando el comando de menú **Archivo | Guardar como...** y haciendo clic en **Sí** cuando la aplicación pregunte si las rutas de acceso de los archivos se deben ajustar a la nueva ubicación del proyecto.



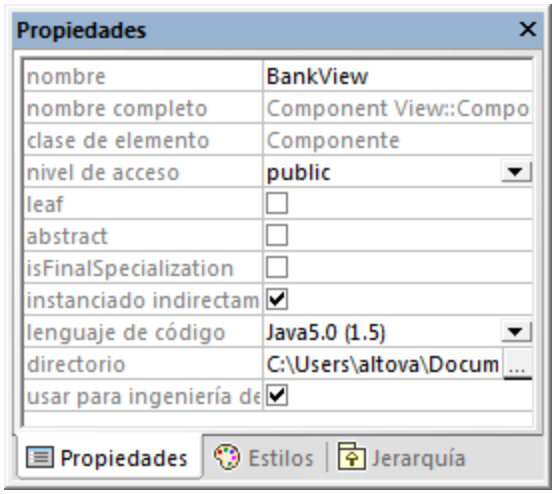
- Copiando el proyecto de UModel (*.ump) a una nueva ubicación y ajustando más tarde las rutas de generación de código de todos los componentes que participen en la generación de código.

El ejemplo de proyecto **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Bank_Multilanguage.ump** permite experimentar con este último método.

1. Encuentre el componente `BankView` en la ventana *Estructura del modelo*.



2. En la ventana *Propiedades* busque la propiedad *directorio* e introduzca la nueva ruta de acceso.



3. Vuelva a sincronizar el modelo y el código.

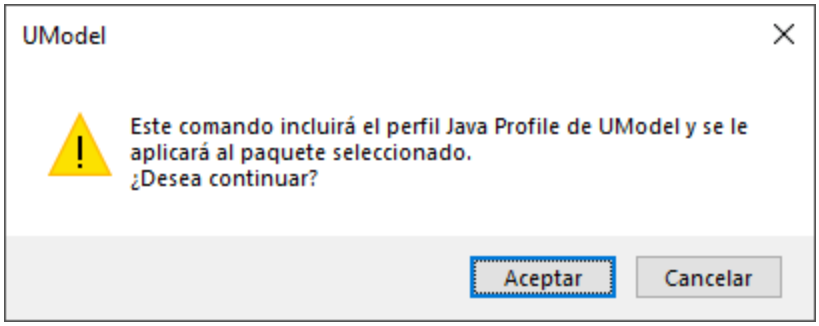
6.1.4 Aplicar perfiles de UModel

Por defecto, siempre que comience un proyecto de modelado nuevo en UModel, el proyecto no tiene información sobre qué aplicación o lenguaje de programación va a necesitar. Por eso, y para adaptar su proyecto de UML a un dominio o lenguaje, debe aplicar un perfil al proyecto.

Hay que distinguir entre dos tipos de perfiles:

- Perfiles integrados en UModel (incluidos C++, C#, VB.NET, Java, BPMN 1.0, BPMN 2.0, SysML, etc.).
- Perfiles personalizados que puede crear para expandir UML a su dominio o necesidades específicas.

Puede añadir a su proyecto cualquiera de los perfiles integrados seleccionando el comando de menú **Incluir | Subproyecto**. Además, UModel le pedirá que aplique un perfil integrado siempre que lleve a cabo una acción que requiera ese perfil en concreto. Por ejemplo, al hacer clic con el botón derecho en un paquete nuevo y seleccionar la opción del menú contextual **Ingeniería de código | Establecer como raíz de espacio de nombres de Java**, UModel le pedirá que aplique el perfil Java.



Para visualizar la lista completa de perfiles integrados de UModel o añadirlos a su modelo de forma manual, seleccione el comando de menú Incluir | Subproyecto. Consulte también Incluir subproyectos.

Para ver las instrucciones sobre cómo crear perfiles personalizados para ampliar o adaptar UML, consulte el apartado Crear y aplicar perfiles personalizados.

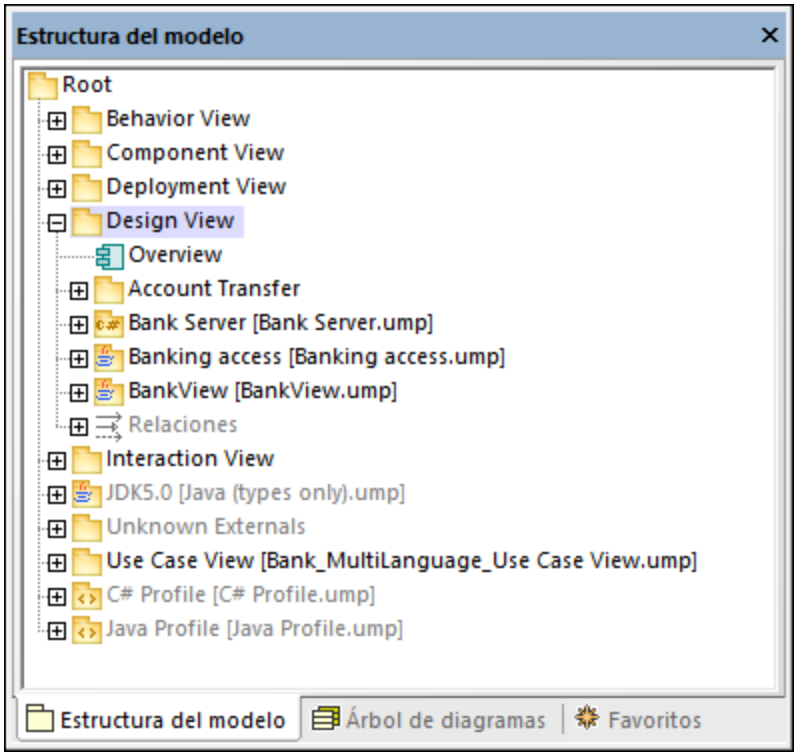
6.1.5 Dividir proyectos de UModel

Puede dividir proyectos de UModel en varios subproyectos, lo que permite que varios programadores editen simultáneamente distintas partes de un mismo proyecto. Los subproyectos son como los archivos estándar de proyecto de UModel y tienen la misma extensión `*.ump`. Cada subproyecto individual se puede añadir a un sistema de control de versiones. El proyecto de nivel superior es el proyecto principal.

Puede crear un subproyecto a partir de prácticamente cualquiera de los paquetes del proyecto principal. Puede escoger si el subproyecto se puede editar desde dentro del proyecto principal o si es de solo lectura. En este último caso el subproyecto solo se puede editar si lo abre como proyecto independiente.

Puede estructurar los subproyectos como prefiera, en una estructura horizontal o jerárquica o en una combinación de ambas. En principio esto permite convertir todos los paquetes de un proyecto principal en archivos de subproyecto.

En la Estructura del modelo los subproyectos aparecen con su correspondiente nombre de archivo y la extensión `.ump` a la derecha entre corchetes. Por ejemplo el proyecto de la imagen siguiente (se trata de **Bank_MultiLanguage.ump** del directorio **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**) incluye varios subproyectos.

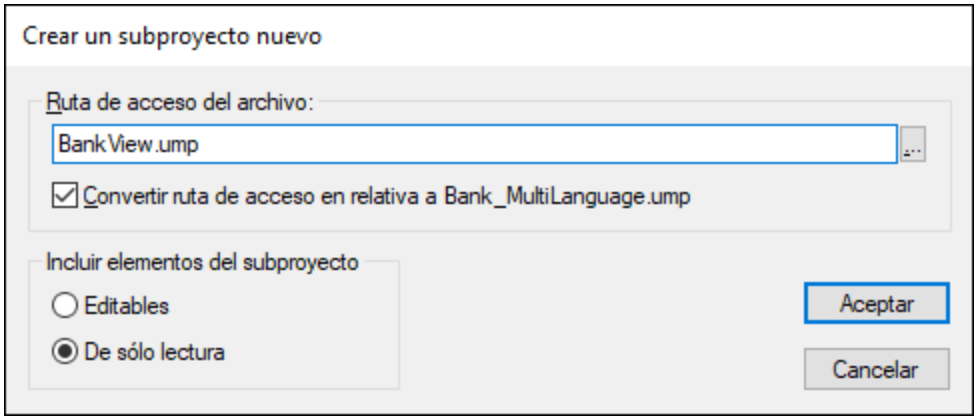


Durante el proceso de ingeniería de código se toman en consideración todos los componentes subordinados de un subproyecto. No existe ninguna diferencia entre un único archivo de proyecto y uno que consiste en varios subproyectos editables. Esto también se aplica a los diagramas UML: se pueden editar a nivel del proyecto principal o a nivel de subproyecto.

Nota: también puede compartir paquetes, así como los diagramas UML que puedan contener, entre distintos proyectos. Para más información consulte el apartado Compartir paquetes y diagramas.

Crear subproyectos

Para crear un subproyecto haga clic con el botón derecho en un paquete y seleccione el comando Subproyecto **| Crear subproyecto nuevo** en el menú contextual.



A continuación haga clic en **Examinar** y seleccione el directorio en el que se debe guardar el subproyecto.

Marque la opción *Editable* para poder editar el subproyecto desde el proyecto principal. (Si selecciona la opción *Solo lectura* no podrá editarlo desde el proyecto principal.)

Nota: puede cambiar la ruta de acceso del archivo del subproyecto siempre que quiera haciendo clic con el botón derecho en el subproyecto y seleccionando **Subproyecto | Editar ruta de acceso**.

Abrir y editar subproyectos

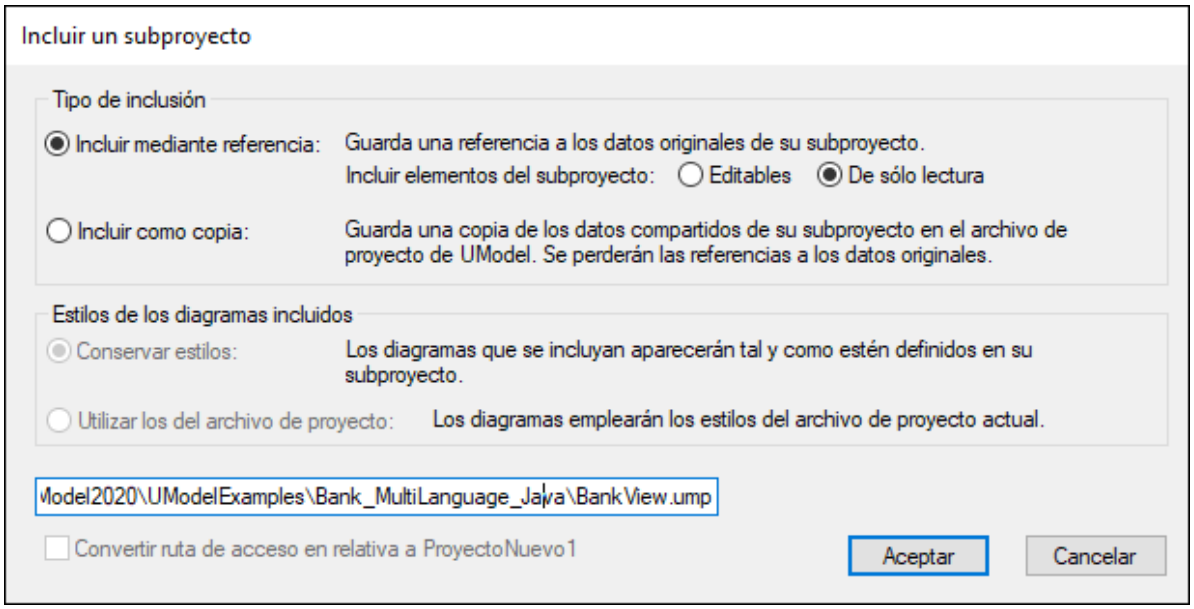
Puede abrir un subproyecto como proyecto independiente de UModel directamente desde el proyecto principal. Para que pueda hacerlo no debe haber ninguna referencia a otros elementos sin resolver. UModel comprueba si así es al crear el subproyecto a partir del proyecto principal y cada vez que se guarda un archivo.

Para abrir un subproyecto como proyecto independiente de UModel haga clic con el botón derecho en el paquete del subproyecto y seleccione **Subproyecto | Abrir como proyecto**. Esa acción inicia otra instancia de UModel y abre el subproyecto como proyecto principal. Las referencias sin resolver, si las hay, aparecen en la ventana *Mensajes*.

Reutilizar subproyectos

Los subproyectos en los que se dividió el proyecto principal se pueden usar en otros proyectos principales.

1. Abra el proyecto y después seleccione el comando de menú **Proyecto | Incluir subproyecto**.
2. Haga clic en el botón **Examinar** y seleccione el archivo `*.ump` que desea incluir.



3. Elija cómo se añade el archivo de subproyecto (mediante referencia o como copia).

Guardar proyectos

Cuando se guarda el archivo de proyecto principal, también se guardan todos los subproyectos editables (es decir, se guardan todos los datos de los paquetes compartidos de los subproyectos).

Por tanto, no debería crear/agregar datos (componentes) fuera de la estructura compartida/del subproyecto, si el subproyecto se definió como editable en un proyecto principal. Si existen datos fuera de la estructura del subproyecto, UModel emite una advertencia en la ventana *Mensajes*.

Guardar subproyectos

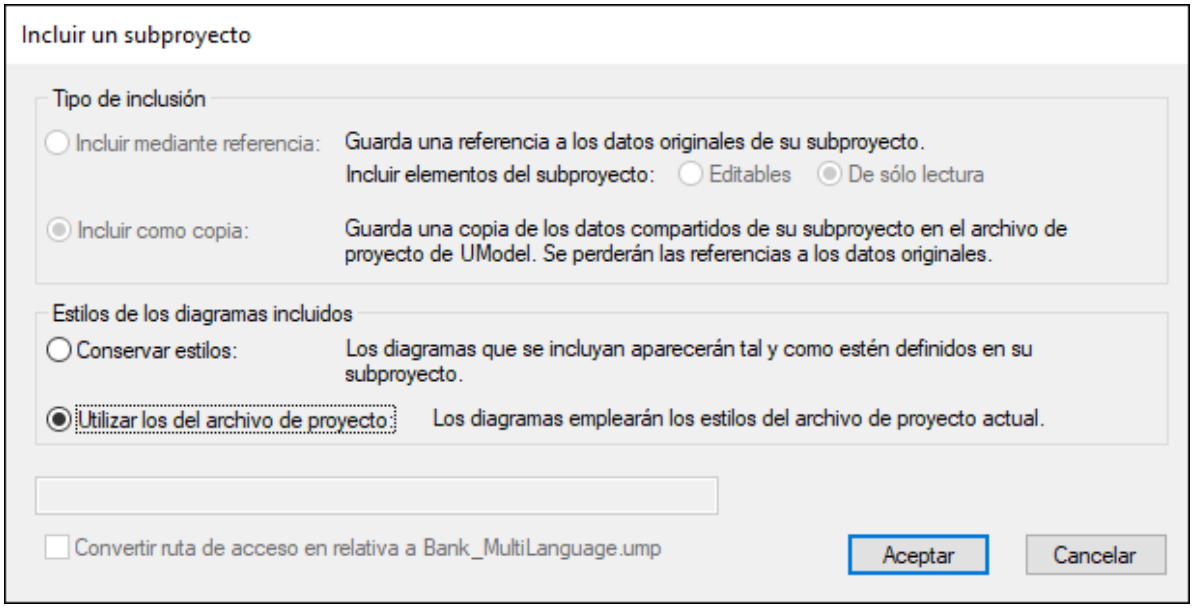
Cuando se guardan subproyectos, se guardan también todas las referencias a subproyectos del mismo nivel y subproyectos secundarios. Por ejemplo, si tenemos los subproyectos del mismo nivel `sub1` y `sub2` y `sub1` utiliza elementos de `sub2`, entonces al guardar `sub1` se guardan automáticamente las referencias a `sub2`.

Si abrimos `sub1` como proyecto principal, se entiende como proyecto autónomo y se puede editar sin referencias al verdadero proyecto principal.

Para volver a integrar los subproyectos en el proyecto principal

Puede volver a copiar subproyectos definidos previamente en el proyecto principal. Si el subproyecto no contiene ningún diagrama, entonces se reintegra de inmediato. Si existen diagramas se abrirá un cuadro de diálogo.

1. Haga clic con el botón derecho en el subproyecto y seleccione la opción **Subproyecto | Incluir como copia**.Esto abre el cuadro de diálogo "Incluir un subproyecto", donde puede definir qué estilo de diagramas se deben usar al incluir el subproyecto.



2. Por último seleccione la opción de estilo y haga clic en **Aceptar**.

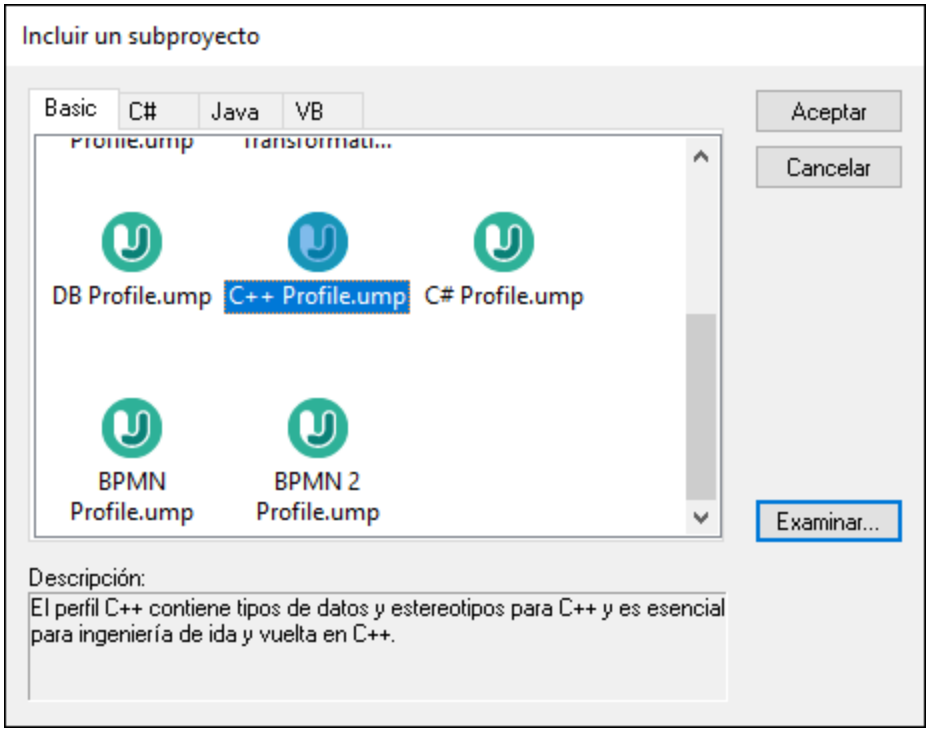
6.1.6 Incluir otros proyectos de UModel

Cuando necesite generar código a partir de un modelo o importar código fuente a un modelo, deberá incluir en el proyecto de UModel un perfil que corresponda al lenguaje de programación elegido (C#, Java, VB.NET).

Puede incluir proyectos de UModel dentro de otros con el comando **Proyecto | Incluir un subproyecto**. Como puede ver en la imagen siguiente, hay varios subproyectos .ump (perfiles de lenguaje necesarios para

las funciones de ingeniería de código) disponibles en la pestaña *Incluir*. En el resto de pestañas encontrará subproyectos .ump de los tipos C#, Java y VB.NET organizados por versión.

> Para que durante el proceso de ingeniería de código se reconozcan correctamente todos los tipos, asegúrese de que incluye tanto el perfil del lenguaje (por ejemplo, el **perfil C#**) como el proyecto del tipo en la versión que corresponda (por ejemplo, **Microsoft .NET 4.7.1 para C# 7.0**). De lo contrario en el proyecto se creará un paquete "Elementos externos desconocidos" que incluirá todos los tipos reconocibles. Observe que para C++ no hay proyectos "tipo", sino que solamente existe un perfil de lenguaje C++.



6.1.7 Compartir paquetes y diagramas

Puede compartir paquetes (y los diagramas UML que puedan contener) entre proyectos distintos de UModel. A su vez, los paquetes se pueden incluir en otros proyectos de UModel mediante referencia o como copia.

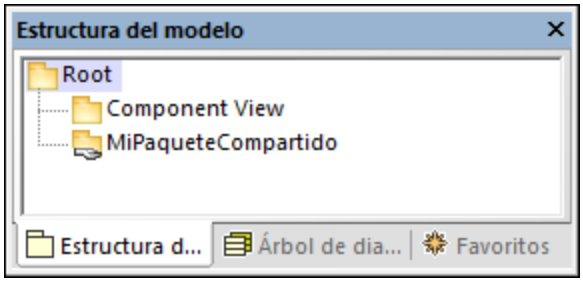
Además los subproyectos se pueden separar del proyecto principal en cualquier momento e incluirse otra vez en el proyecto principal como subproyectos editables o de solo lectura. Asimismo, los paquetes se comparten y se guardan como archivos de subproyecto. Los subproyectos también se pueden añadir al sistema de control de código fuente (*véase* *Trabajo en equipo con UModel*).

Notas

- No se puede compartir un paquete que tenga vínculos a elementos externos (elementos situados fuera del ámbito compartido).
- Cuando cree archivos de proyecto de UModel no use archivos de proyecto como plantilla/copia de otro archivo de proyecto en el que quiere compartir un paquete. Esto daría lugar a conflictos porque cada elemento debe ser único de forma global (ver información sobre identificadores UUID) y en este caso los dos proyectos tendrían elementos con el mismo identificador uuid.

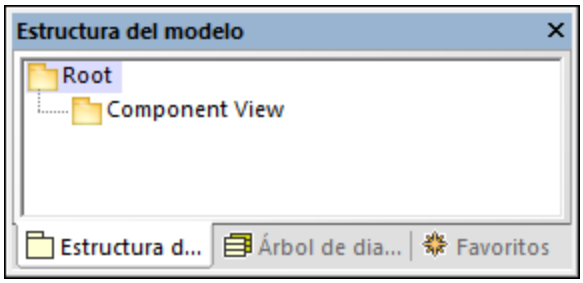
Para compartir un paquete con otros proyectos:

1. En la ventana *Estructura del modelo* haga clic con el botón derecho en un paquete y seleccione **Subproyecto | Compartir paquete**. El icono del paquete ahora es una carpeta sobre una mano: esto indica que el paquete está compartido. A partir de ahora este paquete se podrá incluir en cualquier otro proyecto de UModel.

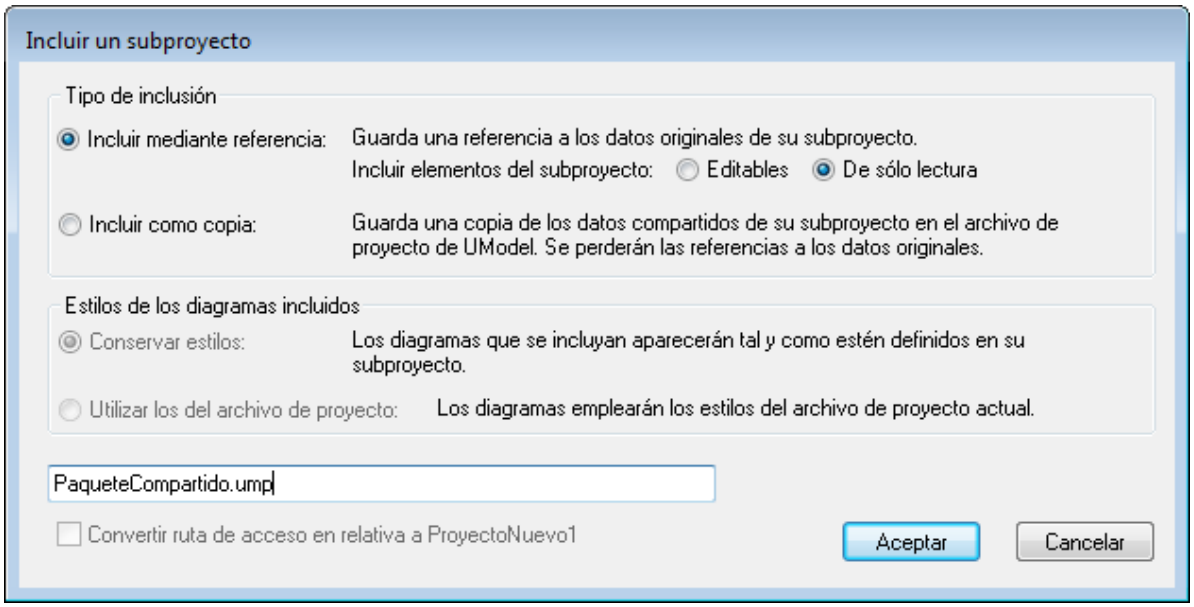


Para incluir/importar una carpeta compartida en un proyecto:

1. Abra el proyecto en el que desea incluir el paquete compartido (en este caso, un archivo vacío).

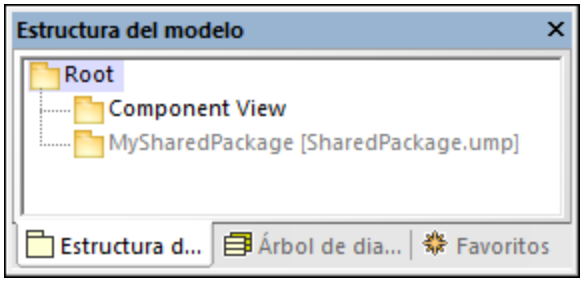


2. Seleccione la opción de menú **Proyecto | Incluir un subproyecto...**
3. Haga clic en el botón **Examinar**, seleccione el proyecto que incluye el paquete compartido y haga clic en **Abrir**. Aparece el cuadro de diálogo "Incluir un subproyecto", donde puede elegir si el paquete/proyecto se incluye mediante referencia o como copia.



4. Seleccione una de las dos opciones y haga clic en **Aceptar**.

El paquete compartido aparece ahora en el paquete nuevo. El proyecto de origen del paquete aparece entre paréntesis (**SharedPackage.ump**).

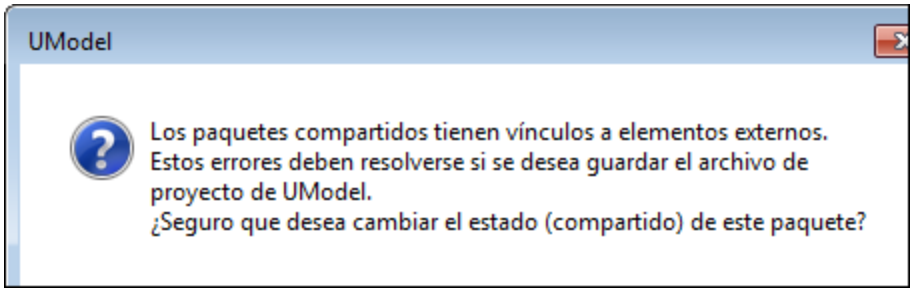


Notas:

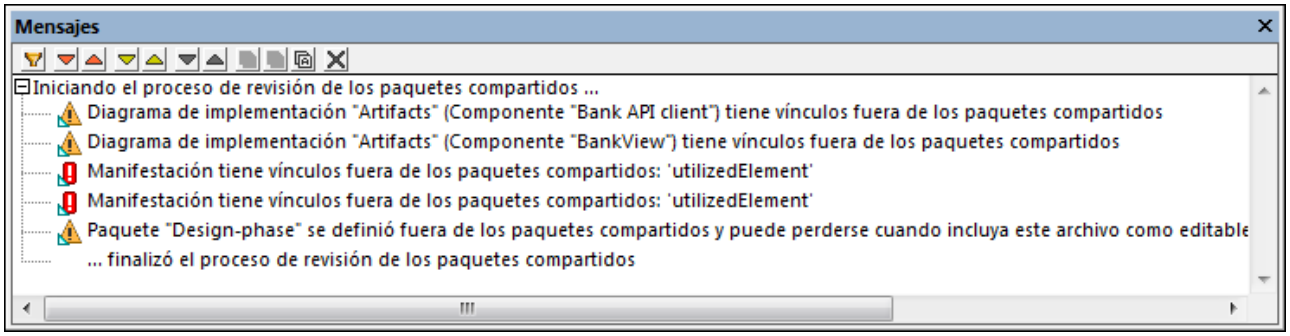
- Cuando incluya un proyecto de código fuente que contiene subproyectos, todos los subproyectos del proyecto de código fuente se incluirán también en el proyecto de destino.
- Las carpetas compartidas que se incluyeron mediante referencia se pueden cambiar a *Incluir como copia* en cualquier momento. Esto se consigue haciendo clic con el botón derecho en la carpeta y seleccione el comando **Subproyecto | Incluir como copia**.

Resolver enlaces a elementos externos

Si intenta compartir un paquete que tiene vínculos a elementos externos, aparece un aviso. Por ejemplo, el mensaje de la imagen siguiente aparece si intenta compartir el paquete `Deployment View` del proyecto de muestra **C:\Documents and Settings\Usuario\Mis Documentos\Altova\UModel2023\UModelExamples\TutorialBankView-start.ump**.



Haga clic en **Sí** para compartir el paquete de todas maneras a pesar de los errores. Si no lo quiere compartir, haga clic en **No**. La ventana *Mensajes* ofrece información sobre todos los vínculos externos.



Si hace clic en una entrada de la ventana *Mensajes*, el elemento correspondiente se resalta en la ventana *Estructura del modelo*.

6.1.8 Consejos para mejorar el rendimiento

Dado que algunos proyectos de modelado pueden alcanzar un tamaño considerable, hay varias maneras de mejorar el rendimiento del modelo:

- Para empezar, compruebe que utiliza el controlador más reciente de su tarjeta gráfica.
- Deshabilite la función de color de sintaxis (en la ventana *Estilos* asigne el valor `falso` a la propiedad usar color de sintaxis).
- Deshabilite el color de fondo degradado para los diagramas. Utilice en su lugar un color sólido (p. ej. seleccione el valor `blanco` para el estilo Diagrama - color de fondo en la ventana *Estilos*).
- Desactive la finalización automática (desde **Herramientas | Opciones | Edición de diagramas** puede desactivar la casilla *Habilitar ayudante de entrada automática).*

6.2 Generar código de programa

Tras diseñar el modelo de la aplicación en UModel (p. ej. uno o más diagramas de clases), tendrá la posibilidad de generar en el lenguaje que prefiera un proyecto prototipo que incluya todas las interfaces, clases, operaciones, etc. que haya definido. UModel permite generar código de programa en C++, C#, VB.NET o Java a partir de un modelo basándose en los elementos UML que encuentra en el proyecto de UModel (p. ej. interfaces, clases, operaciones, etc.). Este proceso también recibe el nombre de *ingeniería directa.* El código generado creará todos los objetos exactamente como se definieron en el modelo para que pueda seguir con su implementación.

La generación de código también es compatible con esquemas XML y bases de datos*. Por ejemplo, puede diseñar un esquema XML o una base de datos con UModel y después generar el archivo correspondiente a partir del modelo (o un script SQL si se trata de una base de datos). Cuando decida usar esta característica deberá consultar las tablas de correspondencia de la sección Correspondencias con elementos de UModel para saber qué elementos del esquema o de la base de datos corresponden a los elementos de UModel.

** para poder generar código a partir de una base de datos es necesario tener la edición Enterprise o Professional de Altova UModel.*

Requisitos

Para poder generar código el proyecto de UModel debe cumplir con estos requisitos básicos:

- en el proyecto debe haber un paquete que esté designado como raíz de espacio de nombres. La raíz de espacio de nombres puede ser un espacio de nombres en C++, C#, Java, VB.NET, XSD o de base de datos. Este paquete debe incluir todas las clases e interfaces a partir de las que se debe generar el código. Para más información consulte el apartado Definir un paquete como raíz de espacio de nombres.
- se debe añadir al proyecto un componente de ingeniería de código. Este componente debe ser realizado por todas las clases o interfaces a partir de las cuales se genera el código. Para más información consulte el apartado Agregar un componente de ingeniería de código.
- en el caso de las bases de datos debe existir una conexión con la base de datos (que se crea con el comando de menú **Proyecto | Importar base de datos SQL**). Tras establecer la conexión podrá diseñar o modificar la estructura de la base de datos en el modelo y confirmar los cambios en la base de datos con ayuda de un script SQL (*véase también Trabajar con bases de datos en UModel*).

También es recomendable que incluya uno de los subproyectos integrados de UModel del lenguaje (o la versión del lenguaje) que quiere usar (véase *Incluir otros proyectos de UModel*). Por ejemplo, si su aplicación tiene como destino una versión concreta de C#, Java o VB.NET, al incluir ese subproyecto podría usar los tipos de datos correspondientes mientra diseña clases, interfaces, etc. en UML.

Para aprender a crear un proyecto desde cero y a generar código consulte el apartado Ejemplo: generar código Java desde UModel y Ejemplo: generar código C++.

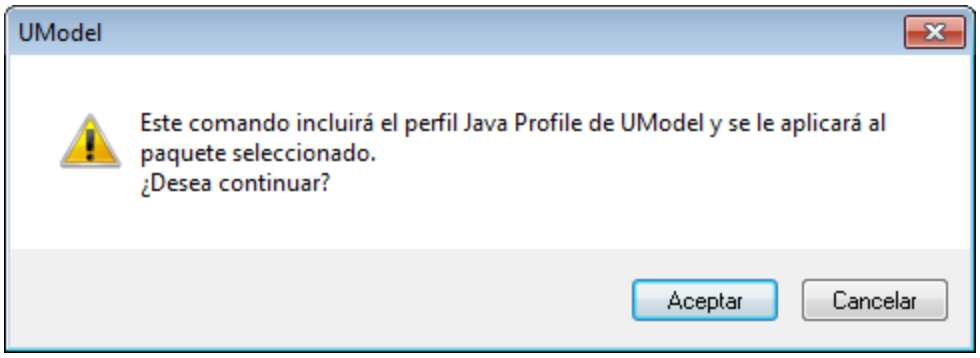
6.2.1 Definir un paquete como raíz de espacio de nombres

Para generar código de programa a partir de un proyecto de UModel debe designar un paquete del modelo como raíz de espacio de nombres.

Para definir un paquete como raíz de espacio de nombres:

- Haga clic con el botón derecho en el paquete (*en la Estructura del modelo*) y seleccione **Ingeniería de código | Establecer como raíz de espacio de nombres de <...>** en el menú contextual donde **<...>** equivale a una de estas opciones: C++, C#, Java, VB.NET, XSD, base de datos.

Cuando un paquete se define como raíz de espacio de nombres, UModel informa al usuario de que el perfil UML del lenguaje correspondiente también se agregará al proyecto y se aplicará al paquete seleccionado. Haga clic en **Aceptar** cuando reciba esta notificación:



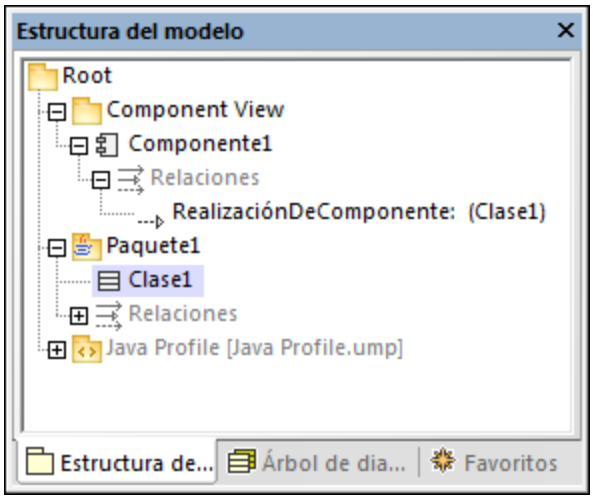
6.2.2 Agregar un componente de ingeniería de código

Para generar código de programa un proyecto de UModel debe contener un componente de ingeniería de código que indique todos los detalles de la generación de código (por ejemplo qué clases del proyecto se deben incluir al generar el código y cuál es el directorio de destino del código generado). Como se muestra a continuación, el componente debe cumplir con ciertos criterios para que genere el código correctamente:

- Se debe asignar al componente una ubicación física (directorio), que es el directorio en el que se genera el código.
- Las clases o interfaces que toman parte en la ingeniería de código se deben realizar con el componente.
- Se debe habilitar la propiedad `Usar para ingeniería de código` para el componente.

Para agregar un componente que realice las clases o interfaces deseadas:

1. Haga clic con el botón derecho en un paquete (en la *Estructura del modelo*) y seleccione **Elemento nuevo | Componente** en el menú contextual. Esto añade un componente nuevo al modelo.
2. En la *Estructura del modelo* haga clic en la clase o interfaz que debe realizar el componente y después arrástrela hasta el componente (en la imagen siguiente se arrastró la `Clase1` del `Paquete1` hasta el `Componente1`). Esto crea automáticamente una relación de `RealizaciónDeComponente` en la *Estructura del modelo*.

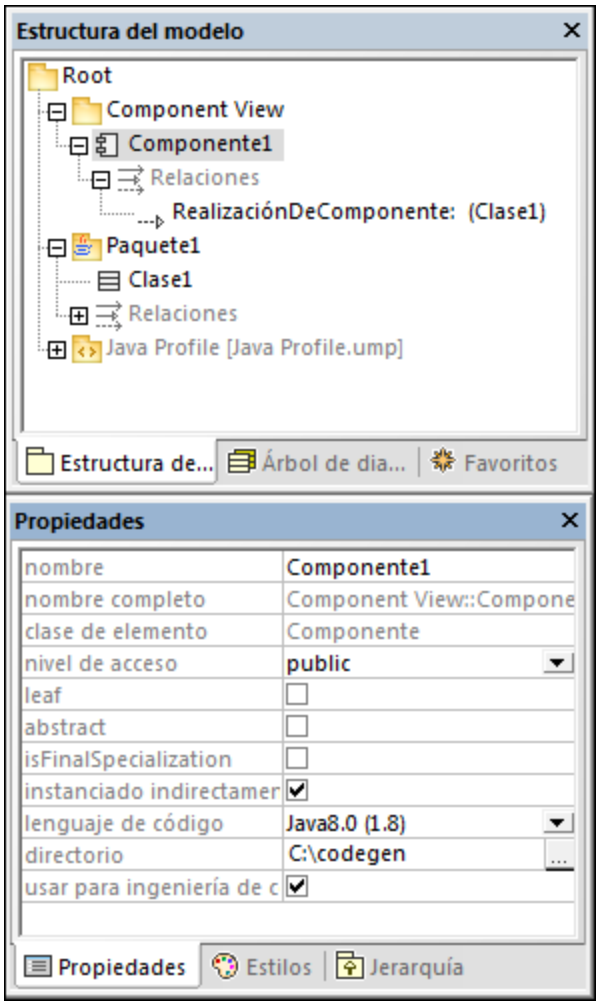


Esto también se puede hacer de otra manera: creando un diagrama de componentes y dibujando la relación `RealizaciónDeComponente` entre el componente y las clases o interfaces. Consulte el apartado Diagramas de componentes para más información.

Para preparar el componente para la generación de código:

1. Seleccione el componente en la *Estructura del modelo* (se supone que este componente ya se realiza con una clase o interfaz).
2. En la ventana *Propiedades* busque la propiedad `directorio` y asígnele como valor la ruta de acceso donde desea generar el código.
3. En la ventana *Propiedades* marque la casilla *usar para ingeniería de código*.

Por ejemplo, en la imagen siguiente, el componente `Componente1` del paquete `Component View` está configurado para generar código Java 8.0 en el directorio de destino **C:\codegen**:



6.2.3 Revisar la sintaxis del proyecto

Es importante revisar la sintaxis del proyecto antes de generar código a partir del modelo. La revisión sintáctica informa sobre errores o problemas que puedan impedir la generación de código. La sintaxis del proyecto se pude revisar con el comando de menú **Proyecto | Revisar la sintaxis del proyecto** (**F11**). Además, UModel revisa la sintaxis del proyecto automáticamente antes de cada actualización del código con el modelo. Los resultados de la revisión sintáctica (errores, advertencias y mensajes informativos) aparecen en la ventana *Mensajes*.

Durante la revisión sintáctica, se inspeccionan varios niveles del archivo de proyecto (*ver tabla más abajo*). Debe tener en cuenta que:

- en el apartado Requisitos de la generación de código encontrará información sobre errores sintácticos corrientes.
- en el caso de los componentes, la revisión solamente se realiza si se habilitó su propiedad `usar para ingeniería de código` en la ventana *Propiedades*.

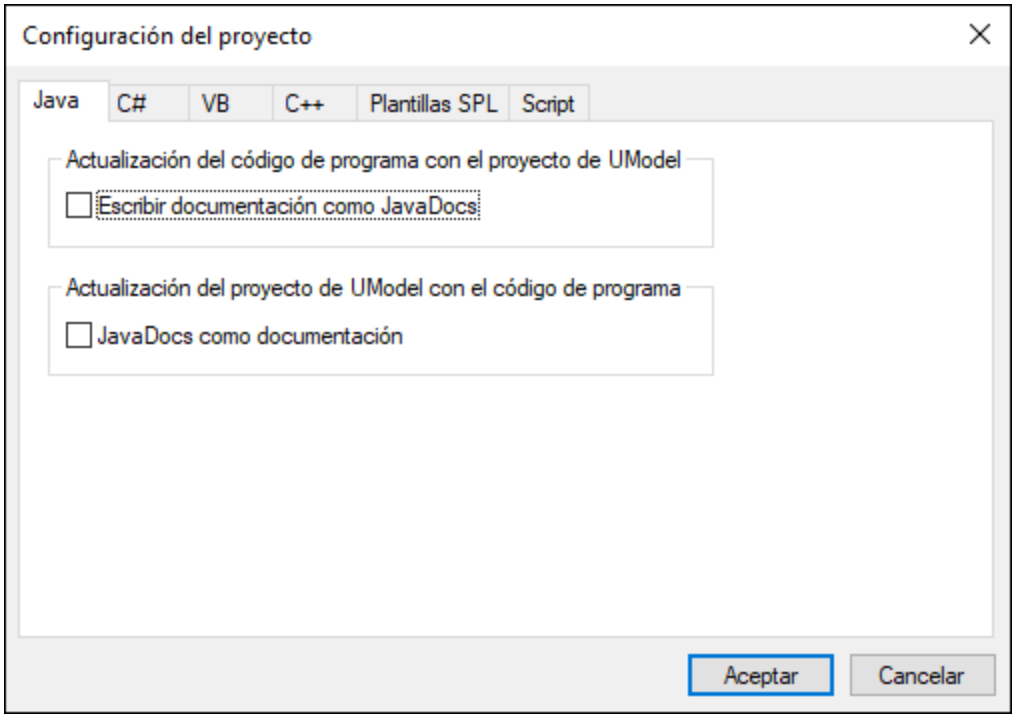
- en el caso de las clases, interfaces y enumeraciones, la revisión solamente si realiza si la clase, interfaz o enumeración está dentro de un espacio de nombres del lenguaje. Es decir, debe estar dentro de un paquete que esté definido como raíz de espacio de nombres.
- las restricciones de los elementos del modelo no se revisan porque no participan en el proceso de generación de código (*véase* *Restricción de elementos*).

Nivel	Se comprueba si...	Gravedad del error si no se cumple la condición
Proyecto	...existe al menos un paquete raíz de espacio de nombres.	Error
Componente	...se estableció el archivo de proyecto o directorio.	Error
	...este componente tiene una relación `RealizaciónDeComponente` con al menos una clase o interfaz.	Error
Clase	...se definió el nombre del archivo de código. **Nota:** esta comprobación no es relevante para clases anidadas.	*Error* si no se marcó la casilla *Generar los nombres de archivo de código que falten* en la pestaña **Herramientas | Opciones | Ingeniería de código**. *Advertencia* si se marcó dicha casilla.
	...se definió el tipo para el parámetro de operación.	Error
	...se definió el tipo para las propiedades.	Error
	...se definió el tipo de retorno de la operación.	Error
	...existen operaciones duplicadas (nombres y tipos de parámetro).	Error
	...existe una relación `RealizaciónDeComponente` con un componente. **Nota:** esta comprobación no es relevante para clases anidadas.	Advertencia
	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave)	Error
	...se produce una herencia múltiple.	Error
Operación de clase	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave)	Error
	...existe un parámetro de retorno.	Error
Parámetro de operación de	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave)	Error

Nivel	Se comprueba si...	Gravedad del error si no se cumple la condición
clase	...el tipo es válido.	Error
Interfaz	...se definió el nombre del archivo de código.	*Error* si no se marcó la casilla *Generar los nombres de archivo de código que falten* en la pestaña **Herramientas | Opciones | Ingeniería de código**. *Advertencia* si se marcó dicha casilla.
	...la interfaz está dentro del espacio de nombres de un lenguaje.	Error
	...se definió el tipo para las propiedades.	Error
	...se definió el tipo para los parámetros de operación.	Error
	...se definió el tipo de retorno de las operaciones.	Error
	...existen operaciones duplicadas (nombres y tipos de parámetro).	Error
	...las interfaces participan en una relación `RealizaciónDeComponente`	Advertencia
	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave)	Error
Operación de interfaz	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave)	Error
Parámetro de operación de interfaz	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave)	Error
Propiedades de interfaz	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave)	Error
Paquete	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave) **Nota:** esta comprobación es relevante si el paquete está dentro de un paquete raíz de espacio de nombres y tiene aplicado el estereotipo `<<namespace>>` desde la ventana *Propiedades*.	Error
Enumeración	...existe una relación `RealizaciónDeComponente` con un componente.	Advertencia

6.2.4 Opciones de generación de código

Cuando se genera código de programa en un proyecto de UModel, la configuración del proyecto puede adaptarse. Las opciones de configuración del proyecto se modifican en el cuadro de diálogo que aparece cuando se hace clic en **Proyecto | Configuración del proyecto** y estas opciones se guardan junto con el proyecto.



Las opciones se agrupan en pestañas distintas:

Pestaña	Opciones
Java	Marque la casilla *Escribir documentación como JavaDocs* para convertir la documentación de elementos de UModel en una documentación equivalente de tipo JavaDocs en el código Java que se genera.
C#	Marque la casilla *Escribir documentación como DocComments* para convertir la documentación de elementos de UModel en comentarios en el código C# que se genera.
VB	Marque la casilla *Escribir documentación como DocComments* para convertir la documentación de elementos de UModel en comentarios en el código VB.NET que se genera.
C++	Véase Opciones de importación de código.

Pestaña	Opciones
Plantillas SPL	Si quiere que UModel lea plantillas SPL desde una ruta de acceso personal, distinta a la ruta de acceso predeterminada, debe introducir aquí la ruta de acceso personal (*véase Plantillas SPL*).
Script	Las opciones de esta pestaña solamente son relevantes si desarrolló proyectos de scripting de UModel para controlar eventos o personalizar el comportamiento de los proyectos UModel. Consulte Editor de scripts para obtener más información.

Además de las opciones de configuración aquí descritas, hay algunas otras opciones que afectan a la generación de código. Estas opciones están en la pestaña *Ingeniería de código* del cuadro de diálogo "Opciones" (**Herramientas | Opciones**). Las opciones para configurar la generación de código a partir de modelos de UModel están en **Actualizar código de programa con proyecto de UModel**. Recuerde que esta configuración es local (es decir, solo afectará a la instalación actual de UModel y no se guardará con el proyecto).

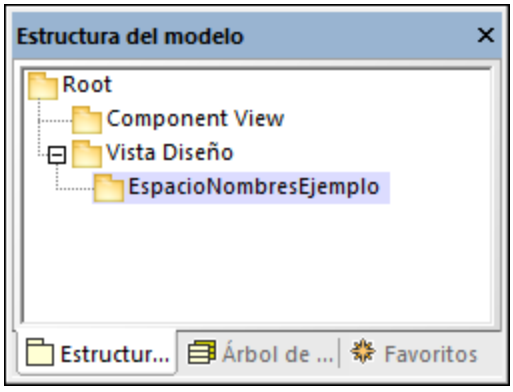
6.2.5 Ejemplo: generar código C#

En este ejemplo se muestra cómo generar código C# con UModel. Para ello primero debe crear un espacio de nombres C# de muestra que contenga unas cuantas clases, después configurar el proyecto para la generación de código y por último generar el código.

En este ejemplo la plataforma de destino es **.NET Standard 2.0 for C# 7.1**. Como se muestra a continuación, esto es posible gracias a un perfil integrado en UModel que define todos los tipos de **.NET Standard 2.0 for C# 7.1**. UModel también incluye perfiles integrados para versiones específicas de .NET Framework por si los necesita (*véase también Incluir otros proyectos de UModel*).

Crear un proyecto de UModel nuevo y su estructura

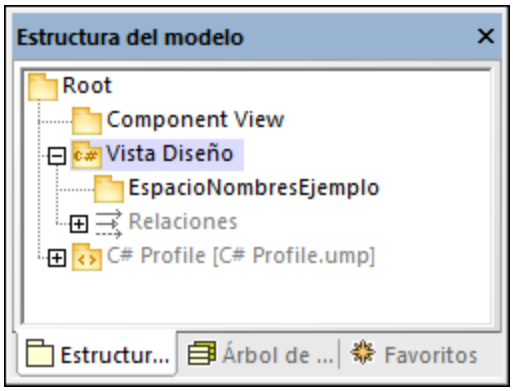
En el menú **Archivo** haga clic en **Nuevo**. Esto crea un proyecto vacío con dos paquetes predeterminados ("Root" y "Component View"). A continuación haga clic con el botón derecho en el paquete "Root" y cree más paquetes, como se muestra más abajo. (Si no conoce la interfaz gráfica de UModel consulte primero los apartados Tutorial de UModel y Cómo modelar.)



En este ejemplo el paquete `Vista diseño` actúa como contenedor de la parte del proyecto que corresponde al diseño (por ejemplo, clases y diagramas de clases), mientras que el paquete `EspacioNombresEjemplo` actúa

como espacio de nombres para todas las clases que se van a crear. Sin embargo, la estructura de los paquetes en general no es prescriptiva y puede organizar los paquetes de otra manera si lo prefiere.

Ingeniería de código
Haga clic con el botón derecho en el paquete `Vista diseño` y seleccione **Ingeniería de código | Establecer como raíz de espacio de nombres C#** en el menú contextual. Cuando UModel le pregunte si quiere aplicar el perfil C# al paquete, haga clic en **Aceptar** para confirmar. Esto incluye el perfil C# integrado de UModel en el proyecto.



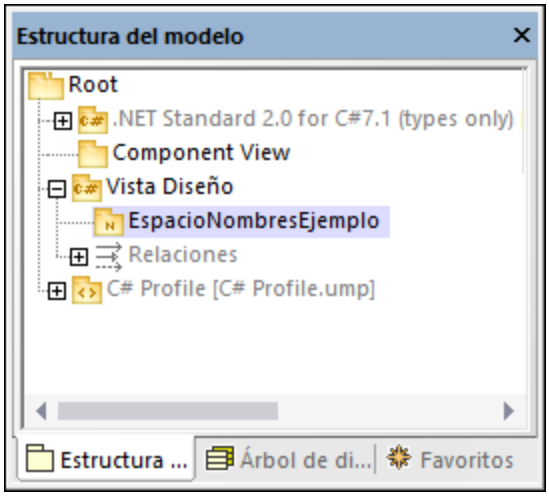
Definir EspacioNombresEjemplo como espacio de nombres

A continuación, haga clic en el paquete `EspacioNombresEjemplo` y marque la casilla `<<namespace>>` en la ventana **Propiedades**. Esto aplica el estereotipo `namespace` al paquete y cambia su icono a . Ahora puede crear clases bajo este espacio de nombres.

Incluir un subproyecto
En este punto el modelo incluye el perfil C#, que contiene los tipos de datos aplicables para C#, pero no los tipos específicos para .NET Standard 2.0, que se encuentran en un perfil distinto de UModel. Para añadir ese perfil al proyecto:

1. En el menú **Proyecto** haga clic en **Incluir un subproyecto**.
2. En la pestaña *C#* seleccione **.NET Standard 2.0 for C# 7.1 (types only)**.
3. Haga clic en **Aceptar**.
4. Cuando la aplicación le pregunte cómo debe incluir el subproyecto, seleccione **Incluir mediante referencia**.

Ahora el proyecto incluye también este otro perfil.



Crear clases C#

Puede crear clases directamente desde la *Estructura del modelo* o desde un diagrama de clases. Para este ejemplo vamos a crear un diagrama de clases desde la ventana *Árbol de diagramas*:

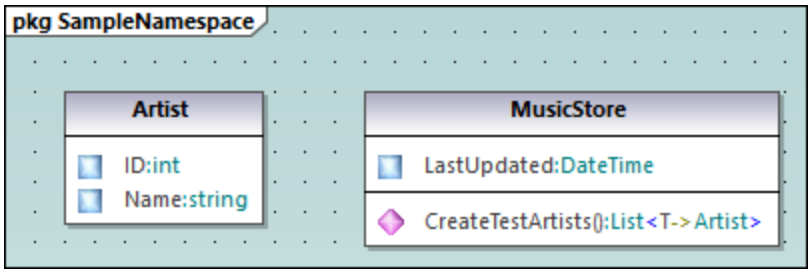
1. Abra el panel **Árbol de diagramas**.
2. Haga clic con el botón derecho en **Diagramas de clases** y seleccione **Diagrama nuevo | Diagrama de clases**.

En este ejemplo se asume que todas las clases se generan bajo el espacio de nombres `EspacioNombresEjemplo`. Por tanto, cuando la aplicación le pida que seleccione un propietario para el diagrama debe seleccionar el paquete `EspacioNombresEjemplo`. Si escoge un paquete distinto, cualquier elemento que añada al diagrama pertenecerá al mismo paquete que el diagrama (lo cual puede no ser su intención).

Crear las clases y su estructura

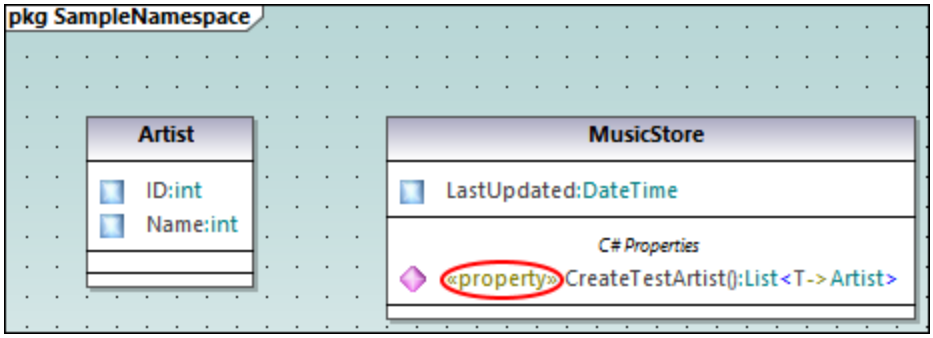
A continuación, cree las clases, los tipos y demás elementos necesarios para su modelo, como un diagrama simple que contenga la clase `Artist` y la clase `MusicStore` (*imagen siguiente*). Para ello siga estas instrucciones:

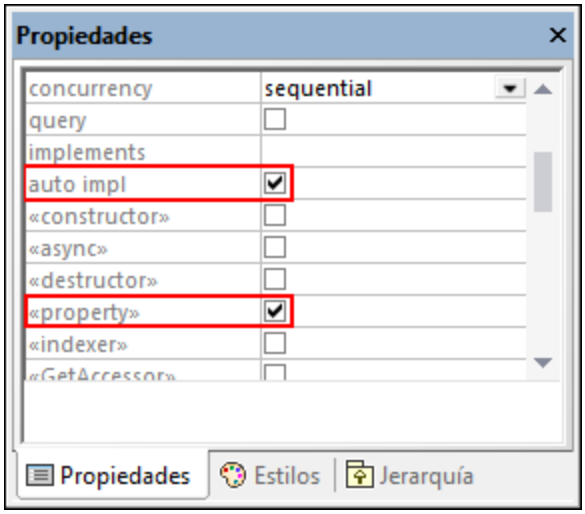
1. Haga clic con el botón derecho en la ventana *pkg SampleNamespace* y seleccione **Nuevo | Clase**.
2. Llame a la clase `Artist`.
3. Haga clic con el botón derecho en la caja `Artist` y cree dos propiedades: `ID`, de tipo `int`, y `Name`, de tipo `string`.
4. Cree la segunda clase, `MusicStore`.
5. Cree una propiedad llamada `LastUpdated` de tipo `DateTime`.
6. Cree una operación e introduzca su nombre y definición como se ve a continación.



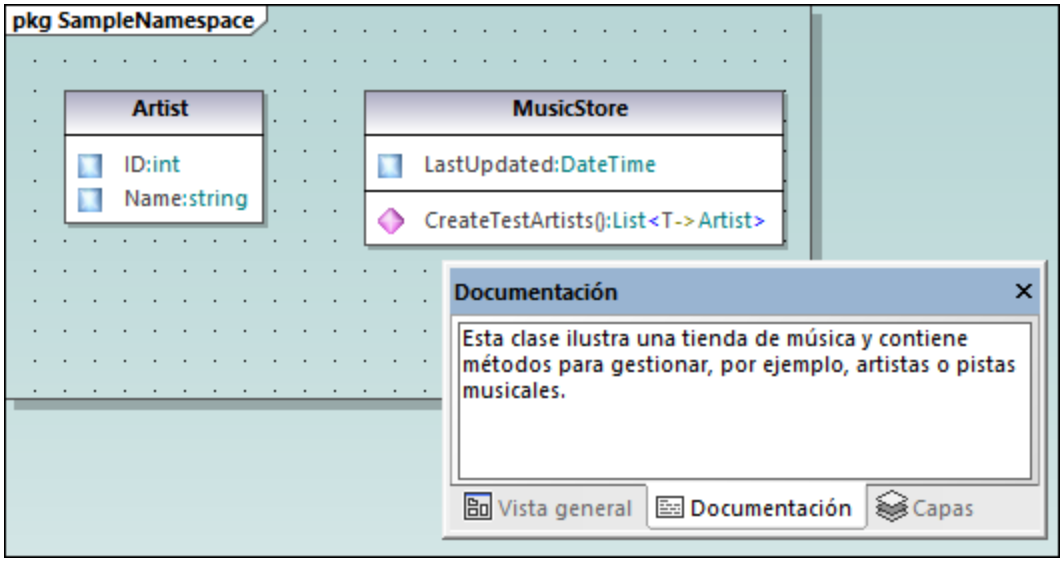
Para las instrucciones paso a paso sobre cómo diseñar clases y sus miembros consulte los apartados Diagramas de clases y Cómo modelar.

Sobre las propiedades C# autoimplementadas
En UModel puede ver si las propiedades C# se han autoimplementado. La opción de autoimplementación se habilita al marcar la casilla **`property`** (en este ejemplo para **`CreateTestArtist ()`**) en la ventana **Propiedades** (*imagen siguiente*).





Agregar documentación (opcional)
También puede hacer clic en la clase `MusicStore` del diagrama y añadir documentación tecleando el texto en el panel Documentación, donde puede generar comentarios en el código para esta clase.

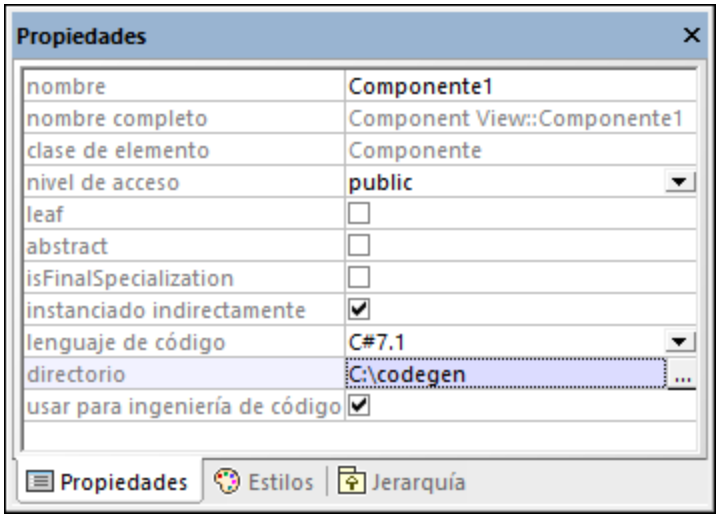


Configurar el proyecto para la ingeniería de código

Ahora vamos a configurar las opciones de la ingeniería de código. Para ello siga estos pasos:

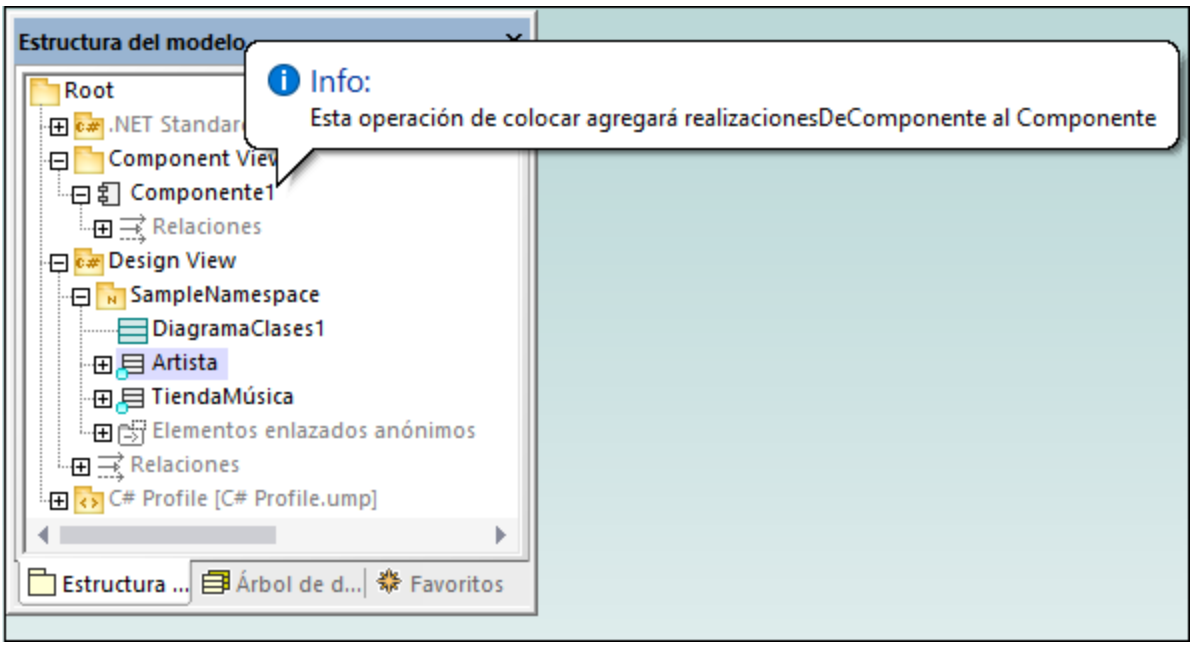
1. Guarde el proyecto en un directorio, si no lo ha hecho ya.
2. A continuación haga clic con el botón derecho en el paquete `Component View` en la Estructura del modelo y añádale un nuevo **Componente** (es decir, un componente de software).
3. Haga clic en ese componente de software nuevo y configure así las opciones del panel **Propiedades**:

   - Lenguaje del código del componente ("C# 7.1" en este ejemplo)
   - Directorio para la generación de código (C:\codegen en este ejemplo).
   - Asegúrese de que la propiedad usar para ingeniería de código está activada.



Crear una relación ComponentRealization

A continuación cree una relación `ComponentRealization` entre las clases a partir de las cuales se genera el código C#. Para ello: en el panel **Estructura del modelo** haga clic en la clase que debe realizar el componente (`Artista` en este ejemplo) y arrástrela hasta el componente de ingeniería de código (`Componente1`) (*imagen siguiente*). Repita este paso con la clase `MusicStore`.

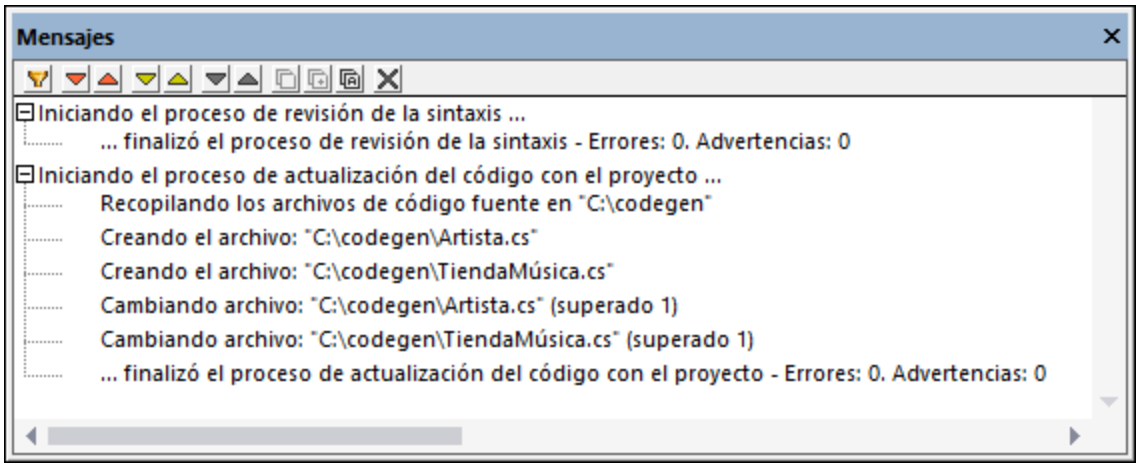


Nota: si olvidó crear una relación **RealizaciónDeComponente** para una clase, UModel sigue generando el archivo de código correspondiente, aunque aparecerán advertencias en la ventana Mensajes. Esta configuración puede cambiarse en **Herramientas | Opciones | Ingeniería de código** (en la casilla *Generar las realizacionesDeComponente que falten*).

Generar código C#

El último paso consiste en generar el código C#. Para ello siga estos pasos:

1. En el menú **Proyecto** haga clic en **Combinar el código de programa con el proyecto de UModel** (o pulse **F12**). Aparecerá un cuadro de diálogo en el que puede ajustar si los cambios en el modelo se deben combinar con los cambios en el código o sobrescribirlos (si procede). Para este ejemplo puede seleccionar **Sobreescribir** porque que estamos generando un proyecto nuevo.
2. Para incluir la documentación de la clase como comentarios en el código generado haga clic en **Proyecto | Configuración del proyecto** y después marque la casilla *Escribir documentación como DocComments*. Para más información consulte el apartado Opciones de generación de código.
3. Haga clic en **Aceptar**. El resultado de la ingeniería de código aparecerá en la ventana *Mensajes* (*imagen siguiente*).



Si añadió documentación a la clase `TiendaMúsica` verá que aparece en forma de comentarios en el código generado:

```
using System;
using System.Collections.Generic;
namespace SampleNamespace
{
   /// Esta clase ilustra una tienda de música y contiene métodos para gestionar, por
ejemplo, pistas musicales o artistas.
   public class MusicStore
   {
      public DateTime LastUpdated;
      public List<Artist> CreateTestArtists()
      {
         // PENDIENTE añadir implementación
      }
   }

}
```

6.2.6 Ejemplo: generar código Java desde UModel

Este apartado explica cómo crear un proyecto de UModel nuevo y generar código de programa a partir de él (un proceso conocido como *ingeniería directa*). El proyecto que vamos a crear será muy sencillo y estará formado por una sola clase. También aprenderemos a preparar el proyecto para la generación de código y a comprobar que la sintaxis del proyecto es correcta. Tras generar el código de programa, lo modificaremos fuera de UModel, añadiendo un método nuevo a la clase. Por último, en el siguiente apartado, se explica cómo combinar cambios realizados en el código con el proyecto de UModel original (un proceso conocido como *ingeniería inversa*).

El lenguaje de generación de código utilizado en este tutorial es Java, pero para los demás lenguajes de generación de código pueden seguirse instrucciones parecidas.

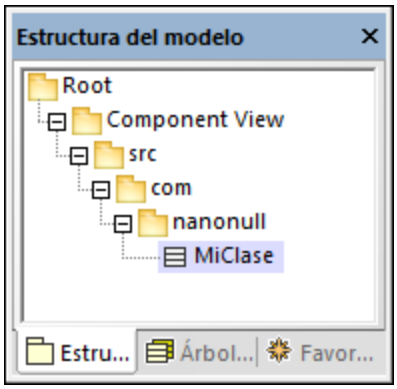
Crear un proyecto de UModel nuevo

Para crear un proyecto de UModel nuevo haga clic en el comando **Archivo | Nuevo** (o pulsando **Ctrl+N** o el botón **Nuevo** de la barra de herramientas).

El proyecto recién creado solamente contiene los paquetes predeterminados "Root" y "Component View". Estos dos paquetes no se pueden eliminar ni renombrar. "Root" es el nivel superior de agrupación para todos los demás paquetes y elementos del proyecto. El paquete "Component View", por su parte, es necesario para los procesos de ingeniería de código y suele almacenar componentes UML que serán realizados por clases o interfaces del proyecto. Sin embargo, hasta ahora no hemos creado ninguna clase.

Para empezar vamos a diseñar la estructura de nuestro programa:

1. Haga clic con el botón derecho en el paquete "Root" de la ventana *Estructura del modelo* y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "src".
2. Haga clic con el botón derecho en "src" y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "com"
3. Haga clic con el botón derecho en "com" y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "nanonull".
4. Haga clic con el botón derecho en "nanonull" y seleccione **Elemento nuevo | Clase** en el menú contextual. Cambie el nombre de la nueva clase por "MiClase".



Preparar el proyecto para la generación de código

Para generar código a partir de un modelo de UModel es necesario cumplir varios requisitos:

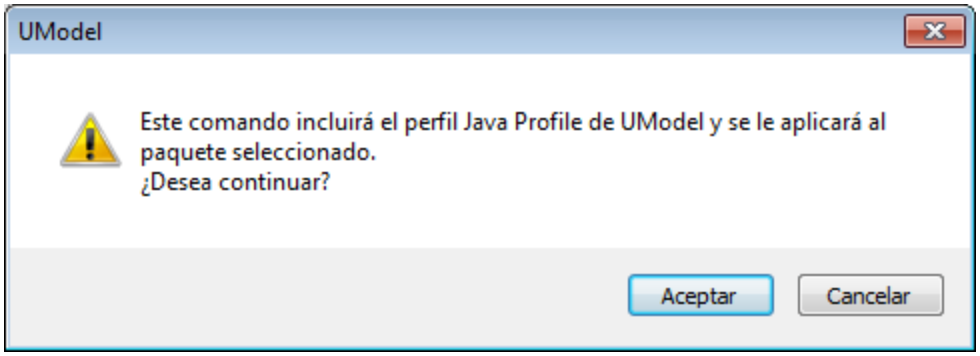
- Debe tener definido un paquete raíz de espacio de nombres Java, C# o VB.NET.
- Debe existir un componente que sea realizado por todas las clases o interfaces para las que se debe generar código.
- El componente debe tener asignada una ubicación física (un directorio). El código se generará en dicha ubicación.
- El componente debe tener habilitada la propiedad `usar para ingeniería de código`.

Estos requisitos se explican más abajo con más detenimiento, pero recuerde que puede comprobar en todo momento si el proyecto cumple con estos requisitos con solo usar la función de validación: haciendo clic en el comando **Proyecto | Revisar la sintaxis del proyecto** (**F11**)

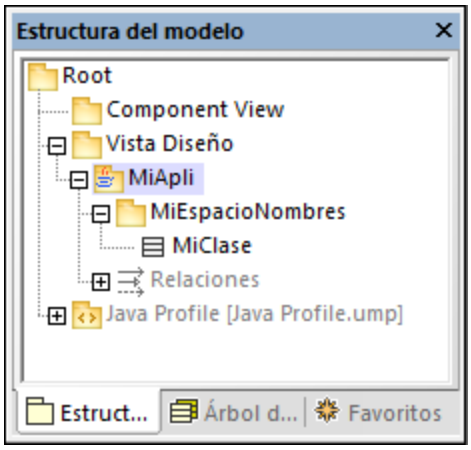
Si se valida el proyecto en este momento, la ventana Mensajes emite el error de validación "`No se encontró`

`la raíz de espacio de nombres. Utilice el menú contextual (submenú "Ingeniería de código") en la estructura del modelo para definir un paquete como raíz de espacio de nombres.`" Para resolver este problema asignaremos el paquete src como raíz de espacio de nombres:

1. Haga clic con el botón derecho en el paquete "src" y seleccione **Ingeniería de código | Establecer como raíz de espacio de nombres de Java** en el menú contextual.
2. Cuando la aplicación pida confirmación para incluir el perfil Java Profile de UModel en el paquete, haga clic en **Aceptar**.

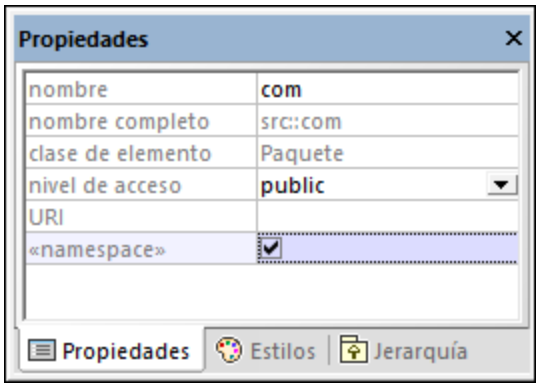


Observe que ahora el paquete tiene un icono distinto ( ) que nos indica que este paquete es una raíz de espacio de nombres de Java. Además, se añadió el perfil Java Profile al proyecto.



El espacio de nombres propiamente dicho se puede definir de la siguiente manera:

1. Seleccione el paquete "com" en la ventana *Estructura del modelo*.
2. En la ventana *Propiedades* habilite la propiedad `<<namespace>>`.

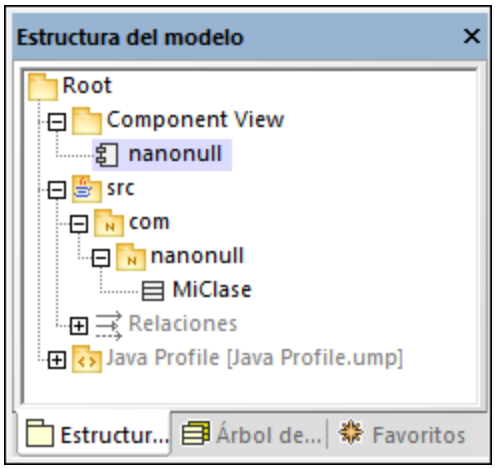


3. Repita el paso anterior con el paquete "nanonull".

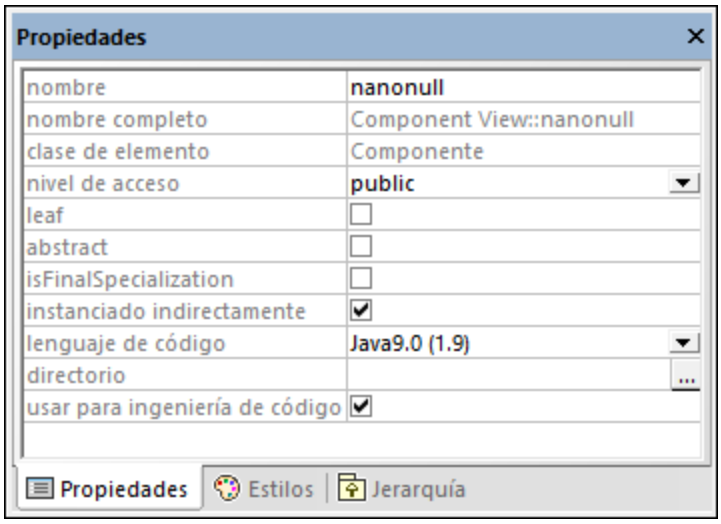
Observe que los paquetes "com" y "nanonull" ahora tienen un icono distinto ( ) que nos indica que estos paquetes son espacios de nombres.

Otro requisito para la generación de código es que los componentes sean realizados por una clase o una interfaz como mínimo. En UML un componente es una pieza del sistema. En UModel el componente nos permite especificar el directorio de generación de código y otras opciones de configuración. Si validamos el proyecto en este momento, aparecerá un mensaje de advertencia en la ventana Mensajes: "*MiClase no tiene una RealizaciónDeComponente para un componente. No se generará código*". Para resolver este problema basta con agregar un componente al proyecto:

1. En la ventana *Estructura del modelo* haga clic con el botón derecho en "Component View" y seleccione **Elemento nuevo | Componente** en el menú contextual.
2. Cambie el nombre del nuevo componente por "nanonull".



3. En la ventana *Propiedades* cambie el valor de la propiedad `directorio` por el directorio donde se debe generar el código (p. ej. "src\com\nanonull"). Observe que la propiedad `usar` para ingeniería de código está habilitada, lo cual es otro requisito imprescindible para la generación de código.



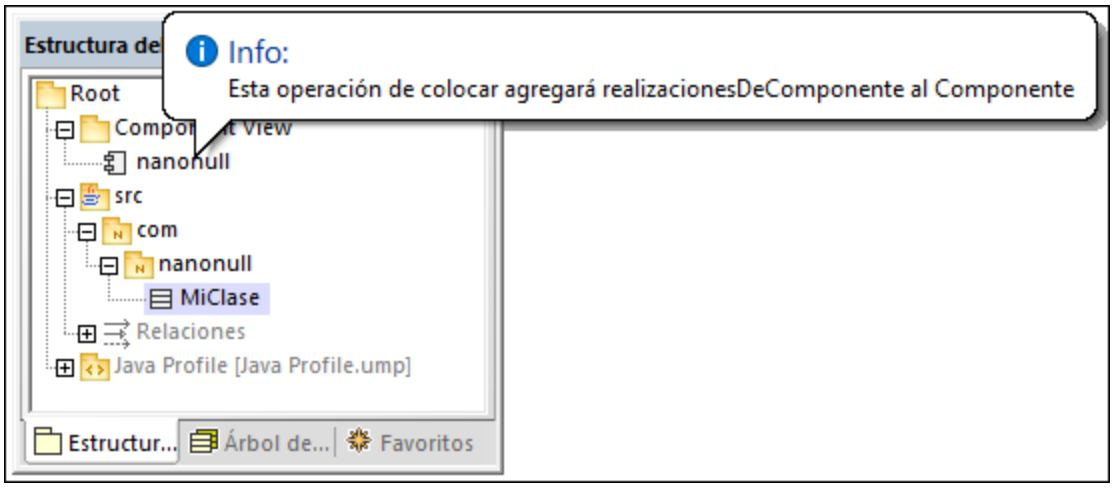
4. Elija un nombre para el proyecto de UModel y guárdelo en un directorio (en este ejemplo: **C:\UModelDemo\Tutorial.ump**).

Nota: el directorio de generación de código puede ser absoluto o relativo al proyecto .ump. Si es relativo, como en este ejemplo, una ruta como **src\com\nanonull** crearía todos los directorio en el mismo directorio en el que se guardó el proyecto de UModel.

En este caso hemos preferido generar código en una ruta de acceso que incluye el nombre de espacio de nombres para evitar mensajes de advertencia porque UModel muestra advertencias de validación si el componente está configurado para generar código Java en un directorio cuyo nombre no coincida con el nombre del espacio de nombres. En este ejemplo, el componente "nanonull" tiene la ruta de acceso "C:\UModelDemo\src\com\nanonull", por lo que no se emitirán advertencias de validación. Si quiere que UModel realice la misma comprobación con C# o VB.NET o si quiere deshabilitar la validación del espacio de nombres para Java, debe seguir estas instrucciones:

1. En el menú **Herramientas** haga clic en el comando **Opciones**.
2. Haga clic en la pestaña *Ingeniería de código*.
3. Marque la casilla correspondiente en el grupo de opciones *Usar espacio de nombres para la ruta del archivo de código*.

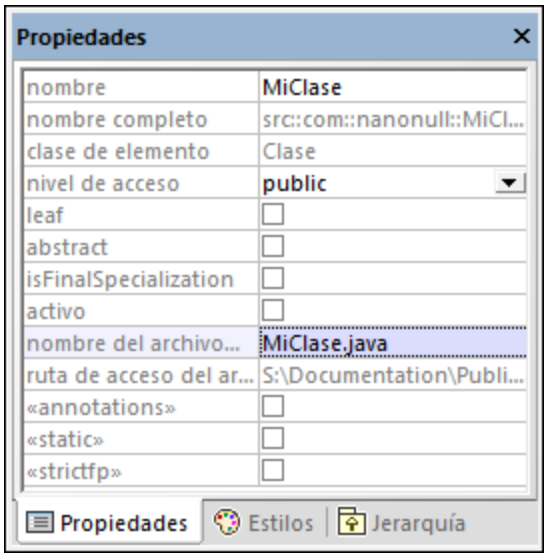
Ahora veamos cómo se puede crear la relación de realización de componente. En la ventana *Estructura del modelo* haga clic en la clase "MiClase" y arrástrela hasta el componente `nanonull`.



De este modo, al componente lo realiza la única clase del proyecto (MiClase). También puede crear la realización de componente desde un diagrama de componentes (*véase* *Diagramas de componentes*).

Lo siguiente que deberíamos hacer es dar un nombre de archivo a las clases o interfaces que participarán en la generación de código. De lo contrario, UModel generará el archivo correspondiente con un nombre de archivo predeterminado y la ventana Mensajes emitirá una advertencia (*no se configuró el nombre del archivo de código. Se generará un nombre predeterminado*). Para solucionar este problema:

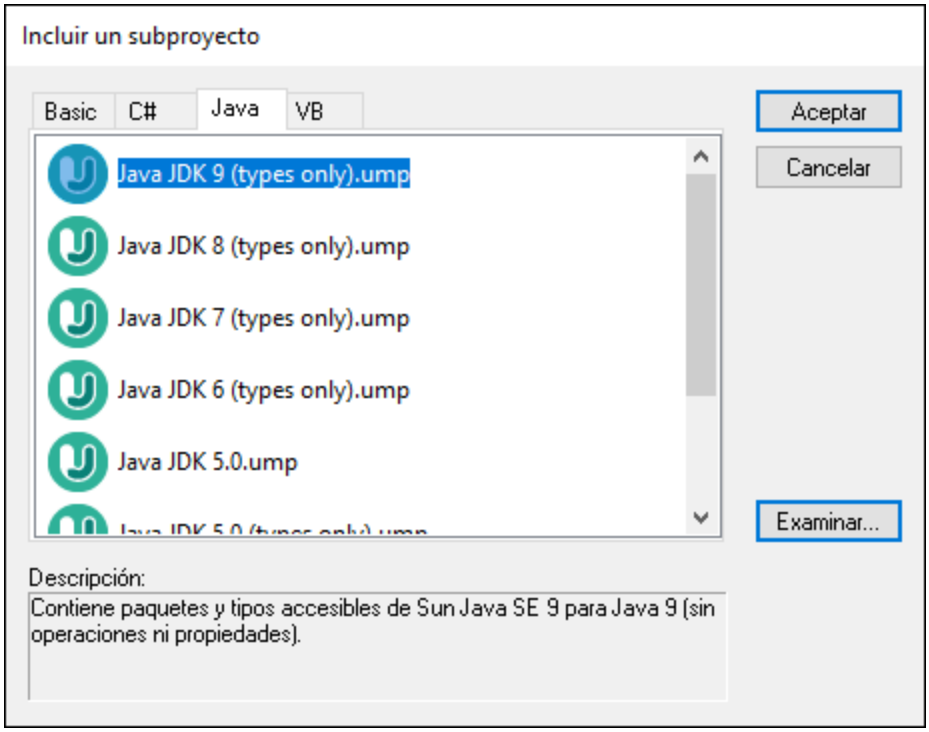
1. Seleccione la clase "MiClase" en la ventana *Estructura del modelo*.
2. En la ventana *Propiedades* cambie el valor de la propiedad `nombre del archivo de código` por el nombre correspondiente (p. ej. `MiClase.java`).



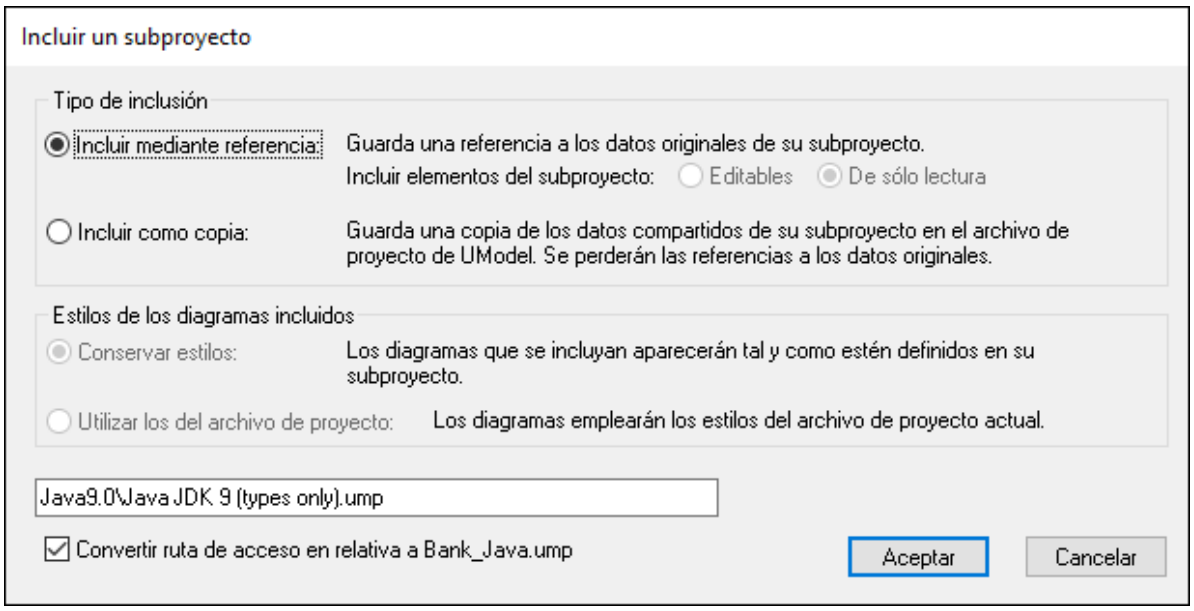
Incluir los tipos JDK

Aunque este paso es opcional, recomendamos que incluya los tipos del lenguaje JDK (Jave Development Kit) como subproyecto de su proyecto de UModel. De lo contrario, los tipos JDK no estarán disponibles cuando cree las clases o interfaces. Esto se puede hacer de la siguiente manera (las mismas instrucciones pueden seguirse para C#, C++ y VB.NET):

1. En el menú **Proyecto** haga clic en **Incluir un subproyecto**.
2. Haga clic en la pestaña **Java** seleccione el proyecto **Java JDK 9 (types only)**.



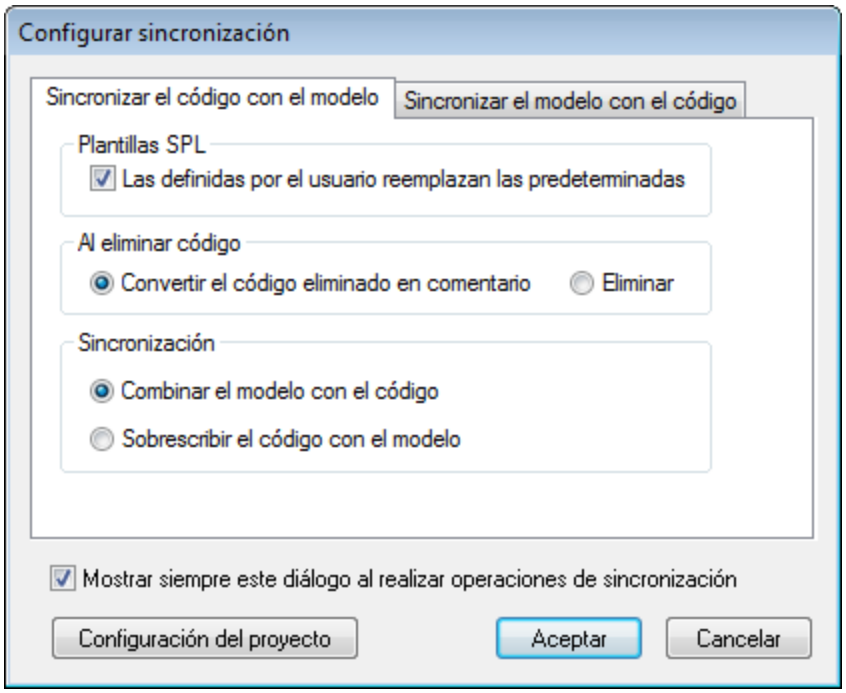
3. Cuando la aplicación pregunte si se incluye mediante referencia o como copia, elija la opción *Incluir mediante referencia*.



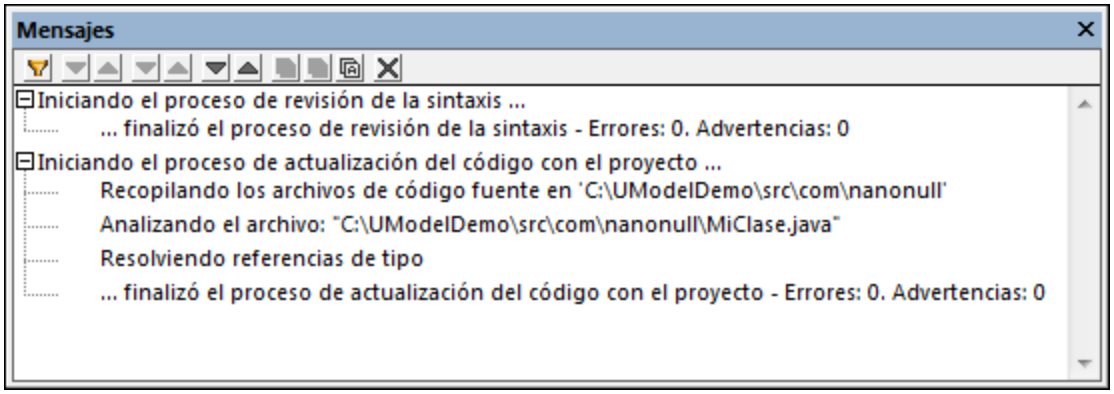
Generar código

Ahora que se cumplen todos los requisitos para la generación de código, podemos empezar el proceso:

1. En el menú **Proyecto** haga clic en el comando **Combinar el código de programa con el proyecto de UModel** (o pulse **F12**).



2. Deje las opciones de sincronización predeterminadas como están y haga clic en **Aceptar**. UModel revisa la sintaxis del proyecto automáticamente y en la ventana Mensajes aparecen los resultados de la revisión:



Modificar el código fuera de UModel

La generación de código de programa es el primero paso para empezar a desarrollar una aplicación o un sistema de software. En un proyecto real, el código sufrirá un gran número de modificaciones antes de llegar a ser un programa completo. Siguiendo con nuestro ejemplo, ahora abriremos el archivo generado **MiClase.java**

en un editor de texto y añadiremos un método nuevo a la clase, como se muestra a continuación. El archivo **MiClase.java** tendrá este aspecto:

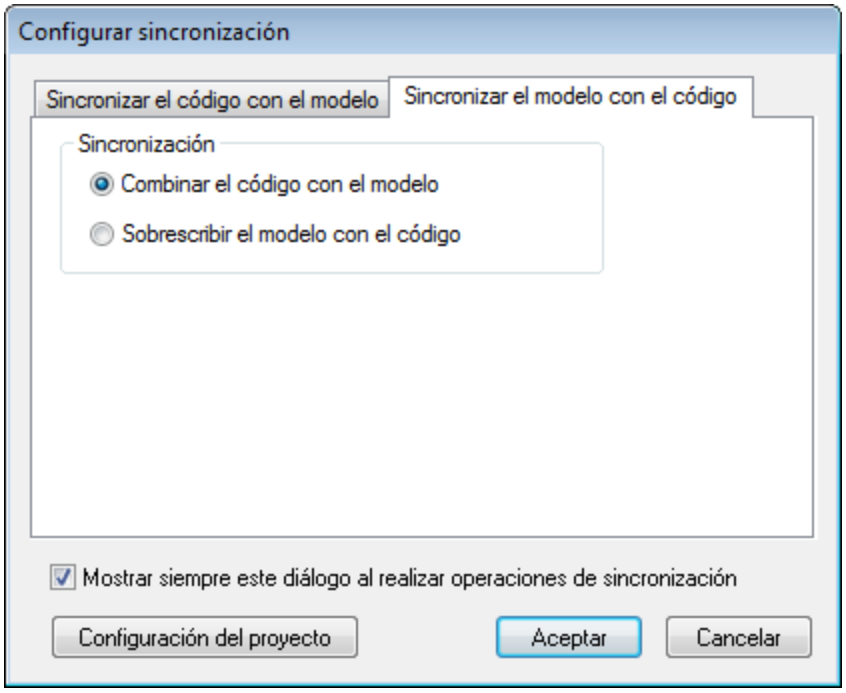
```
package com.nanonull;
public class MyClass{
   public float sum(float num1, float num2){
     return num1 + num2;
   }
}
```

MiClase.java

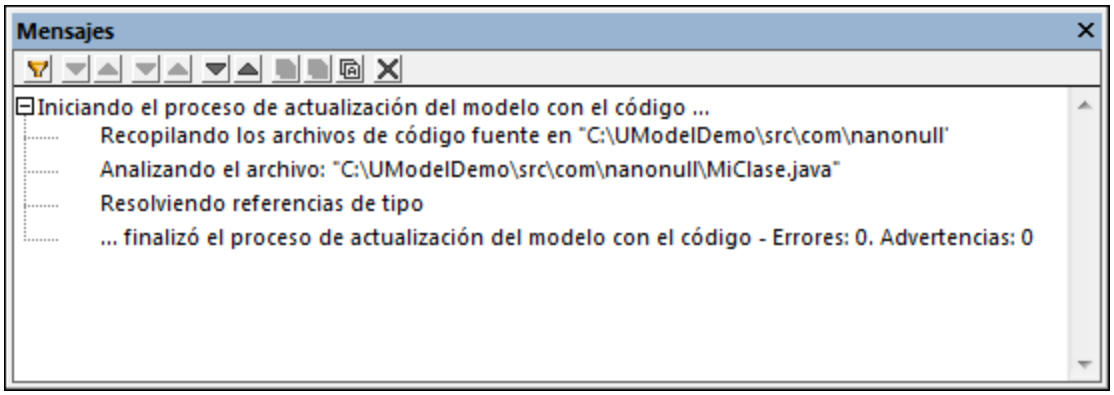
Combinar cambios realizados en el código con el modelo original

Ahora podemos combinar los cambios realizados en el código con el código original:

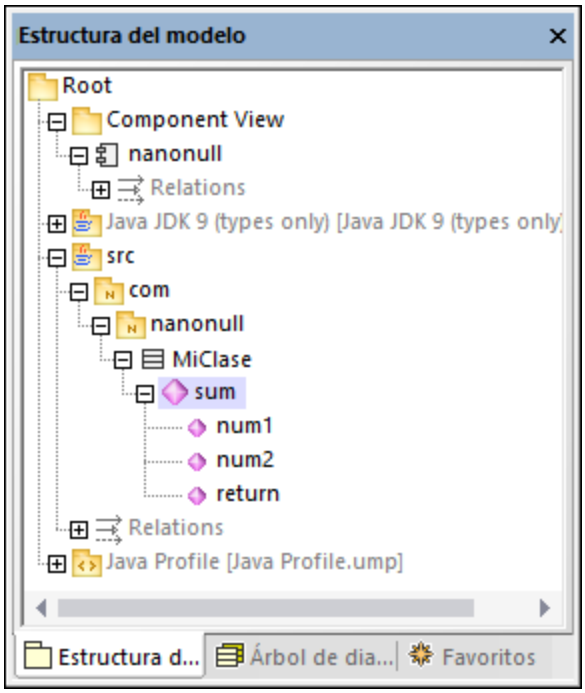
1. En el menú **Proyecto** haga clic en el comando **Combinar el proyecto de UModel con el código de programa** (o pulse **Ctrl+F12**).



2. Deje las opciones de sincronización predeterminadas como están y haga clic en **Aceptar**. UModel revisa la sintaxis del proyecto automáticamente y en la ventana Mensajes aparecen los resultados de la revisión:



La clase `sum` (que se obtuvo mediante ingeniería inversa desde el código) aparece ahora en la ventana *Estructura del modelo*.

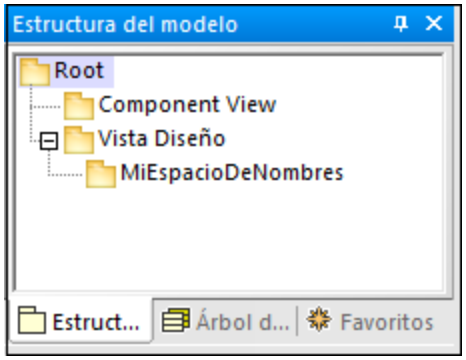


6.2.7 Ejemplo: generar código C++

Este ejemplo muestra cómo generar código C++ con UModel. Para ello, primero debe crear un proyecto simple en UModel, configurarlo para que genere código y generar el código.

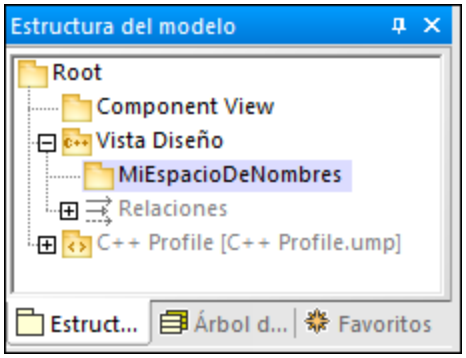
Crear un nuevo proyecto de UModel y su estructura

En el menú **Archivo** haga clic en **Nuevo**. Esto creará un nuevo proyecto vacío con dos paquetes predeterminados ("Root" y "Component View"). A continuación haga clic con el botón derecho en el paquete "Root" y cree varios paquetes más, como se muestra a continuación (si todavía cono conoce bien la interfaz gráfica del usuario de UModel, consulte primero Tutorial de UModel y Cómo modelar).



En este ejemplo el paquete "Vista Diseño" sirve como contenedor para los elementos que vayan a formar parte de su modelo (clases y diagramas de clases, por ejemplo), mientras que el paquete "MiEspacioDeNombres" sirve como espacio de nombres para todas las clases que se vayan a crear. Sin embargo, por lo general la estructura de paquetes no es en absoluto normativa, por lo que los puede organizar como quiera.

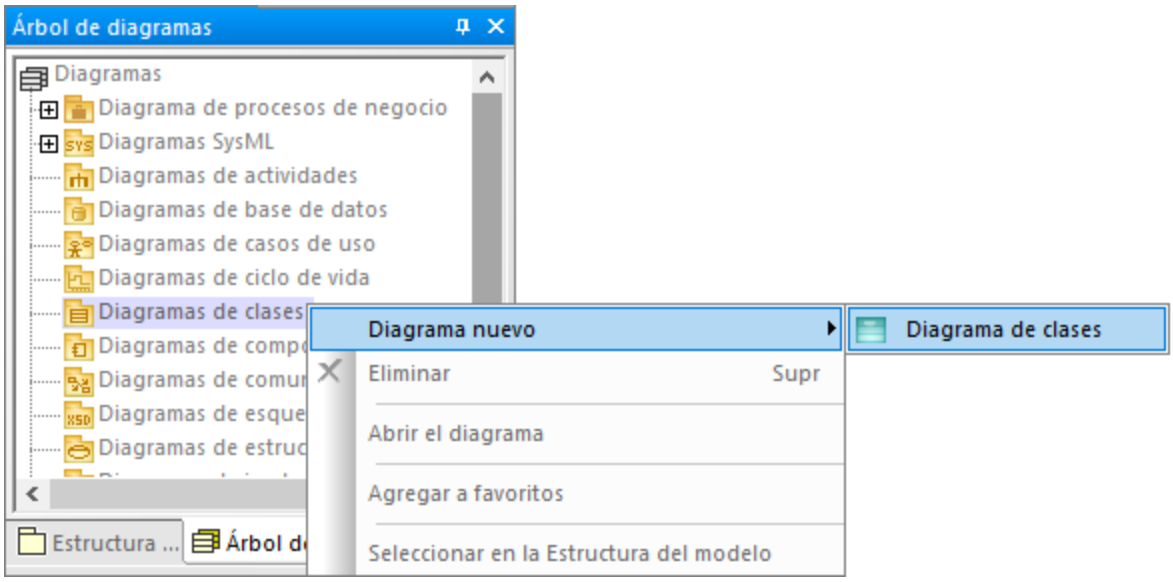
Haga clic con el botón derecho en el paquete "Vista Diseño" y seleccione **Ingeniería de código | Establecer como raíz de espacio de nombres C++** en el menú contextual. Cuando UModel le pregunte, confirme que quiere aplicar el perfil C++ al paquete haciendo clic en **Aceptar**. Ahora el perfil C++ integrado en UModel está incluido en el proyecto.



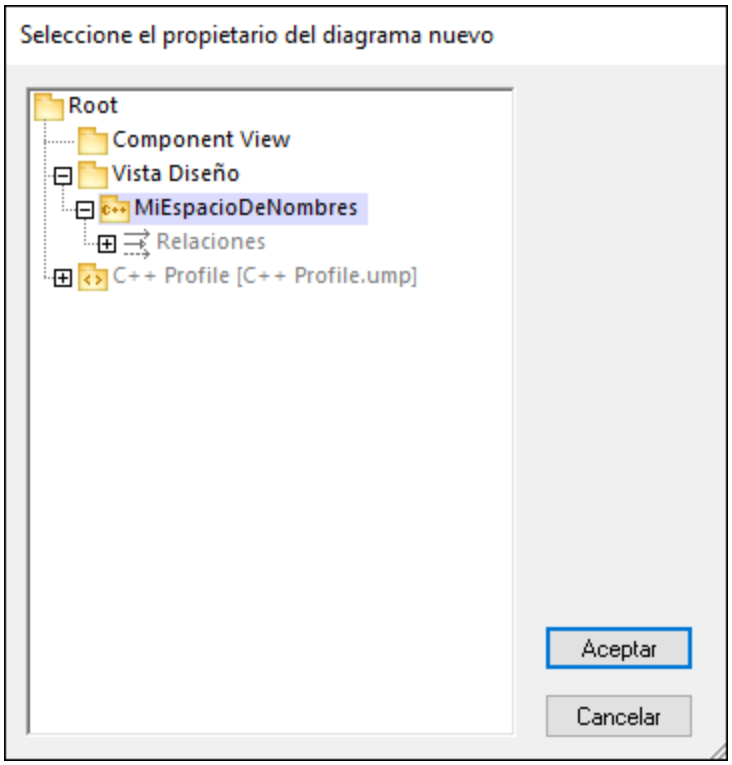
A continuación haga clic en el paquete "MiEspacioDeNombres" y marque la casilla `<<namespace>>` en la ventana Propiedades. Esto aplica el estereotipo "namespace" al paquete, cuyo icono cambia a . Ahora puede crear clases bajo este espacio de nombres.

Crear clases C++

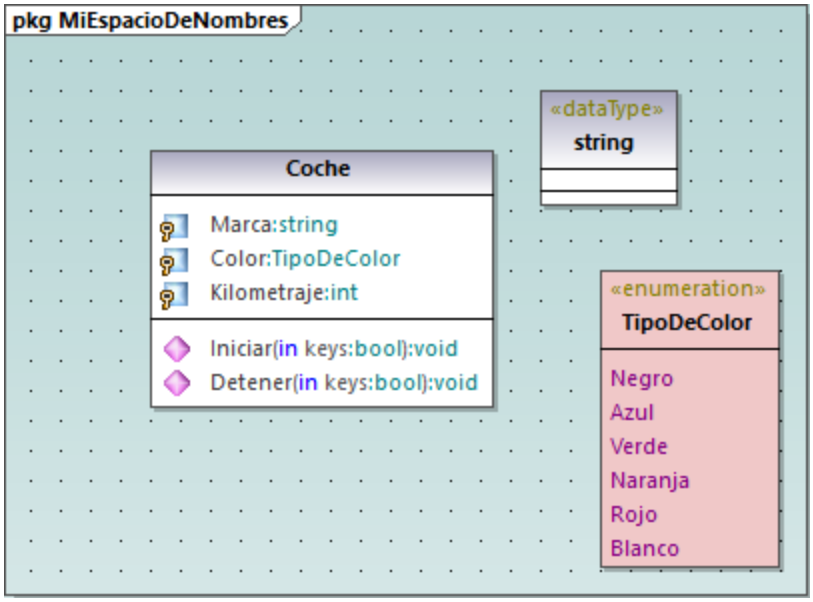
Puede escoger entre crear clases directamente desde la ventana *Estructura del modelo* o desde un diagrama de clases. Para este ejemplo crearemos un diagrama de clases desde la ventana *Árbol de diagramas*, como se muestra a continuación:



Este ejemplo asume que todas sus clases deben generarse bajo el espacio de nombres "MyNamespace". Por tanto, cuando el programa le pida que seleccione un propietario para el diagrama, seleccione el paquete "MyNamespace" (*imagen siguiente*). Si escoge un paquete distinto, cualquier elemento que añada al diagrama pertenecerá al mismo paquete que el diagrama (lo cual puede no ser su intención).



A continuación, cree las clases, los tipos y demás elementos necesarios para su modelo, como un diagrama simple que ilustre la clase `Coche`:



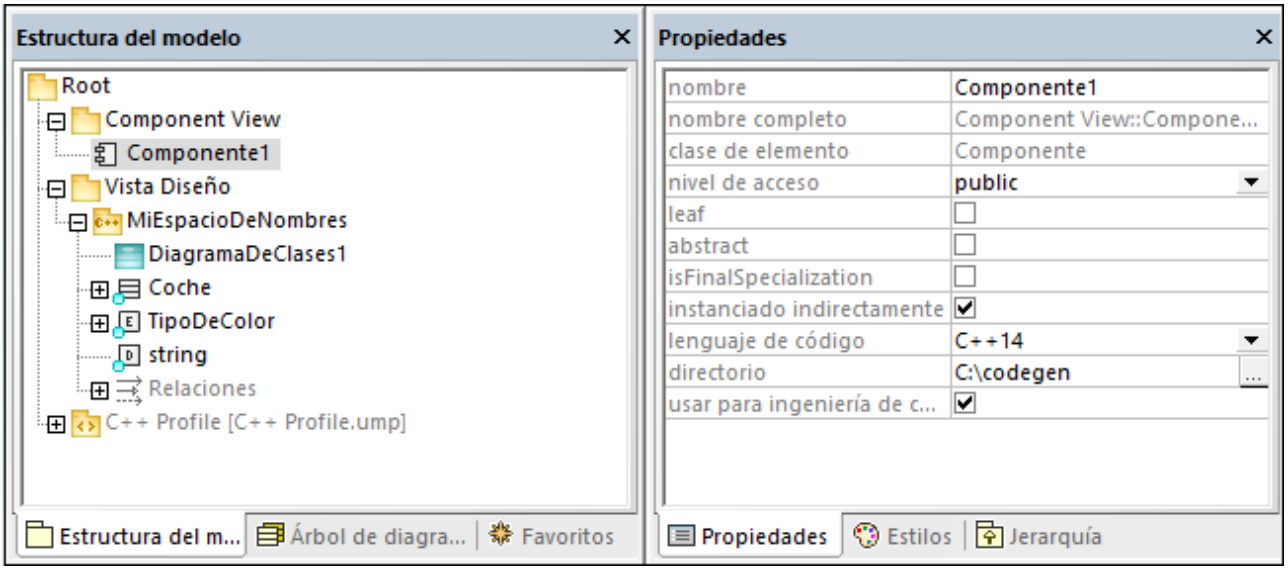
En el diagrama anterior son de especial interés la enumeración `TipoDeColor` y el tipo de datos `string`. Estos tipos no son fundamentalmente C++, por lo que no están incluidos en el perfil C++ integrado en UModel. Por esta razón deben crearse explícitamente en el modelo usando respectivamente los botones de la barra de herramientas **Enumeración** y **TipoDeDatos**. Los tipos fundamentales (como `int` o `bool`), por el contrario, están automáticamente disponibles para seleccionar a medida que usted teclea. Consulte también Finalización automática en clases. Para leer las instrucciones completas sobre diseño de clases y de sus miembros, consulte las secciones Clases de diagramas y Cómo modelar.

Configurar el proyecto para ingeniería de código

Haga clic con el botón derecho en el paquete "Component View" y añada un nuevo **Componente** (es decir, un componente de software). Haga clic en el nuevo componente de software y, en la ventana Propiedades, aplique la siguiente configuración a las propiedades:

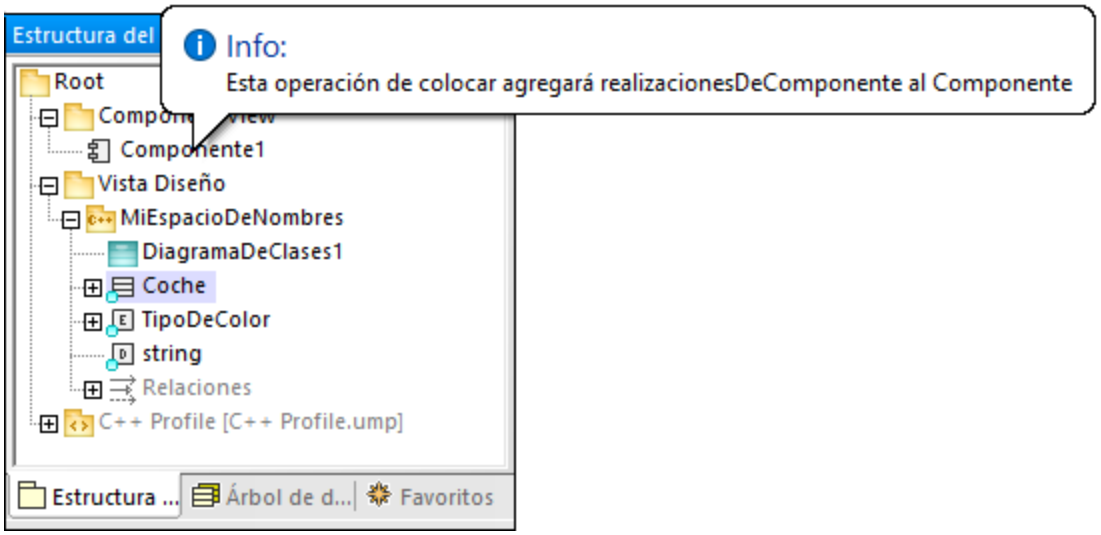
- Código del lenguaje del componente ("C++ 14" en este ejemplo)
- Directorio de generación de código ("C:\codegen en este ejemplo")

Asegúrese también de que la casilla de la propiedad "usar para ingeniería de código" está **marcada**.



A continuación, cree una relación **RealizaciónDeComponente** entre las clases desde las que se debe generar el código C++ (`Coche` y `TipoDeColor` en este ejemplo) y el componente de ingeniería de código. Puede hacer esto desde un diagrama de componentes o, más fácilmente, como sigue:

- En la ventana Estructura del modelo haga clic en la clase que el componente debe realizar (`Coche` y `TipoDeColor` en este ejemplo) y arrástrela hasta el componente de ingeniería de código (`Component1`).



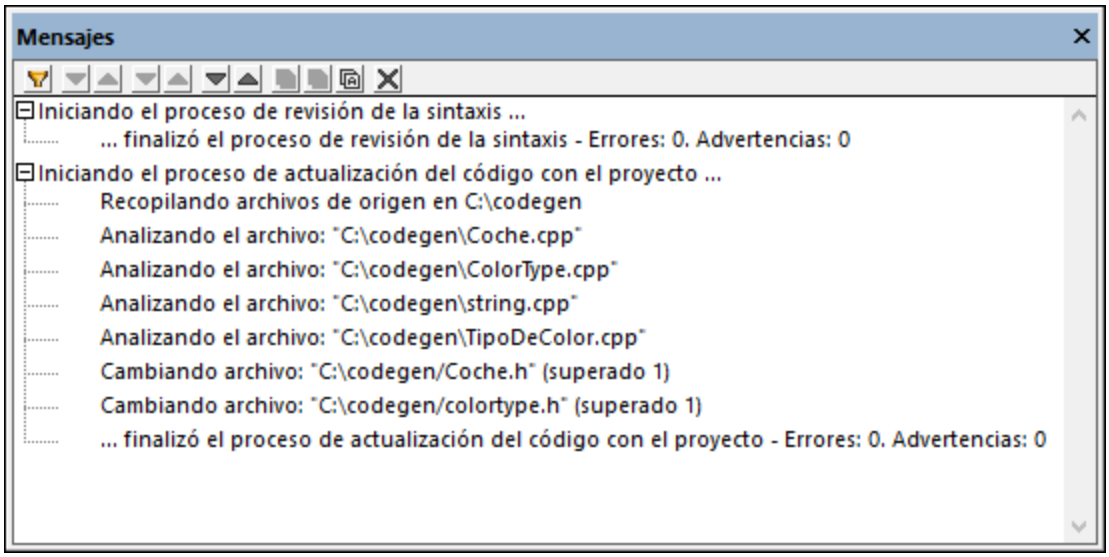
Haga lo mismo con la clase `ColorType`.

Nota: en caso de que olvide crear una relación **RealizaciónDeComponente** para una clase, UModel sigue generando el archivo de código correspondiente, aunque aparecerán advertencias en la ventana Mensajes. Esta configuración puede cambiarse en **Herramientas | Opciones | Ingeniería de código** (en la casilla "Generar las **realizacionesDeComponente** que falten").

Generar código C++

Ahora puede generar el código C++:

1. En el menú **Proyecto** haga clic en **Combinar el código de programa con el proyecto de UModel** (o pulse la tecla **F12**). Aparecerá un cuadro de diálogo en el que puede ajustar si los cambios en el modelo se deben combinar con los cambios en el código o sobrescribirlos (si procede). Para este ejemplo los valores predeterminados son correctos, ya que el código se genera por primera vez. Para obtener más información, consulte Configurar la sincronización del código.
2. Haga clic en **Aceptar**. El resultado de la ingeniería de código se mostrará en la ventana Mensajes.



6.2.8 Plantillas SPL

A la hora de generar código C++, C#, Java o VB.NET, así como esquemas XML, UModel usa un lenguaje de plantillas llamado SPL (Spy Programming Language). Las plantillas SPL imponen la sintaxis de los archivos de código que se generan. Estas plantillas SPL se pueden personalizar para, por ejemplo, modificar ligeramente la sintaxis del código que se genera. No obstante, solo tiene sentido editar las plantillas SPL cuando se trata de un lenguaje compatible con UModel. Si creara plantillas SPL completamente nuevas para otros lenguajes, podría generar código nuevo pero no actualizar el que ya existe (porque la sintaxis del lenguaje sería desconocida para UModel).

Las plantillas SPL están en el directorio **UModelSPL** relativo al directorio de instalación del programa.

> No modifique las plantillas SPL predeterminadas que vienen con UModel, ya que estas afectan directamente a la generación de código predeterminada. Si necesita personalizar la generación de código debe crear plantillas personalizadas (*ver más abajo*).

Las plantillas SPL solamente se utilizan si se genera código nuevo (es decir, si se añaden nuevas clases, operaciones, etc. al modelo). Las plantillas SPL no afectarán a ningún código ya existente.

El apartado Referencia de SPL ofrece una introducción al lenguaje SPL.

Para modificar las plantillas SPL que vienen con UModel:

1. Primero debe localizar las plantillas SPL que vienen con UModel. El directorio donde están guardadas es: **...\UModel2023\UModelSPL\Java\Default** (o **...\C#\Default**, **...\VB\Default**.)
2. Copie los archivos SPL que desea editar/modificar en el directorio primario. Por ejemplo, si quiere modificar el aspecto de una clase Java en el código generado, copie el archivo **Class.spl** de **...\UModel2023\UModelSPL\Java\Default** y péguelo en **...\UModel2023\UModelSPL\Java**.
3. Realice los cambios en los archivos `.spl` y guárdelos.

Para usar las plantillas SPL personalizadas:

1. Seleccione el comando de menú **Proyecto | Configurar sincronización**.
2. Marque la casilla *Las definidas por el usuario reemplazan las predeterminadas*.

6.3 Importar código fuente

Puede importar código de programa Java, C#, C++ y VB.NET en UModel mediante el proceso conocido como *ingeniería inversa.* Estos son los tipos de proyecto que se pueden importar a UModel:

- Proyectos Java (archivos de proyecto Eclipse .project, archivos de proyecto NetBeans project.xml y archivos JBuilder .jpx)
- Proyectos C# y VB.NET (archivos de proyecto Visual Studio .sln, .csproj, .csdprj, .vbproj y .vbp, así como Borland .bdsproj)
- Proyectos C++98, C++11 y C++14, C++17 (archivos de proyecto Visual Studio .vcxproj y .sln creados con Visual Studio 2010, 2012, 2013, 2015. 2017 y 2019).

Además de importar código fuente desde un proyecto también puede importar código desde un directorio de código fuente. Se trata del mismo proceso, pero importar un directorio de código fuente es la opción más útil cuando no se quieren usar los tipos de proyecto de la lista anterior. Para ver un ejemplo consulte el apartado Ingeniería inversa (del código al modelo).

El código fuente se puede importar a un proyecto de UModel nuevo y vacío o en un proyecto de UModel que ya exista. Durante la importación podrá especificar si los elementos importados deben sobrescribir los elementos del modelo o combinarse con ellos. También tendrá la opción de generar diagramas de clases y paquetes durante la importación de código.

El asistente de importación ofrece opciones de importación específicas dependiendo del tipo de plataforma (Java, .NET, C++). Por ejemplo, si el código Java/C#/VB.NET importado contiene comentarios, hay una opción para convertirlos en documentación de UModel. Consulte el apartado Opciones de importación de código para obtener más información.

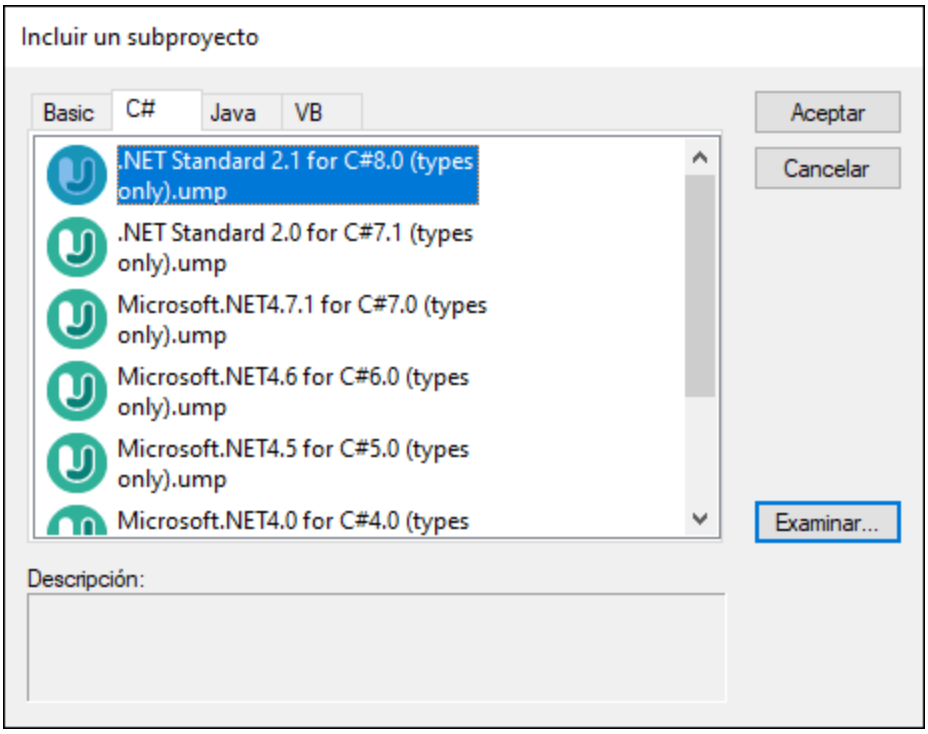
Cuando termine de importar el código C++, C#, VB.NET o Java en UModel, podrá modificar el modelo (p. ej. añadiendo clases nuevas o cambiando el nombre de propiedades y operaciones) o sincronizarlo con el código original (lo que se denomina proceso de *ingeniería de ida y vuelta*). Para más información consulte el apartado Sincronizar el modelo y el código fuente.

Requisitos

UModel viene con varios subproyectos integrados creados específicamente para la función de ingeniería de código. Estos subproyectos incluyen los tipos de datos correspondientes a cada tipo de lenguaje y de plataforma. Antes de intentar importar código fuente en un proyecto de UModel, recomendamos que incluya el subproyecto de UModel integrado que corresponda al lenguaje y a la plataforma elegidos (*véase Incluir otros proyectos de UModel*). Si no importa el subproyecto correspondiente, ciertos tipos de datos no serán reconocidos y se colocarán, después de la importación, en un paquete llamado Elementos externos desconocidos.

Para incluir un subproyecto con los tipos de datos del lenguaje necesario:

1. En el menú **Proyecto** haga clic en el comando **Incluir un subproyecto**.
2. Haga clic en la pestaña correspondiente (p. ej. Java 8.0, C# 6.0, VB 9.0) y después haga clic en **Aceptar**.



Debe tener en cuenta que:

- cuando se incluye un subproyecto de tipos de datos para un lenguaje, UModel añade automáticamente el perfil del lenguaje elegido al proyecto. El subproyecto del perfil (.ump) solo contiene los tipos más básicos y es distinto al subproyecto de tipos de datos (también un archivo .ump), que contiene muchas más definiciones de tipos.
- si realiza la importación sin incluir un subproyecto de tipos de datos, la operación de importación se llevará a cabo y UModel incluirá también el perfil del lenguaje en el proyecto automáticamente. Sin embargo, los tipos desconocidos se colocarán en el paquete Elementos externos desconocidos. Para evitarlo se recomienda incluir el subproyecto de tipos de datos para el lenguaje y la plataforma elegidos, tal y como se explica más arriba.
- para C++ no existe un subproyecto con todos los tipos de C++ posibles de la Standard Template Library (STL), sino que existe un perfil de lenguaje C++ con tipos básicos (fundamentales). Puede añadir este subproyecto manualmente, como se muestra más arriba. De otro modo se añadirá automáticamente al proyecto cuando importe código C++ o cuando haga clic con el botón derecho en un paquete y seleccione el comando Ingeniería de proyecto **| Establecer como raíz de espacio de nombres C++** en el menú contextual.

Importar código fuente desde un proyecto

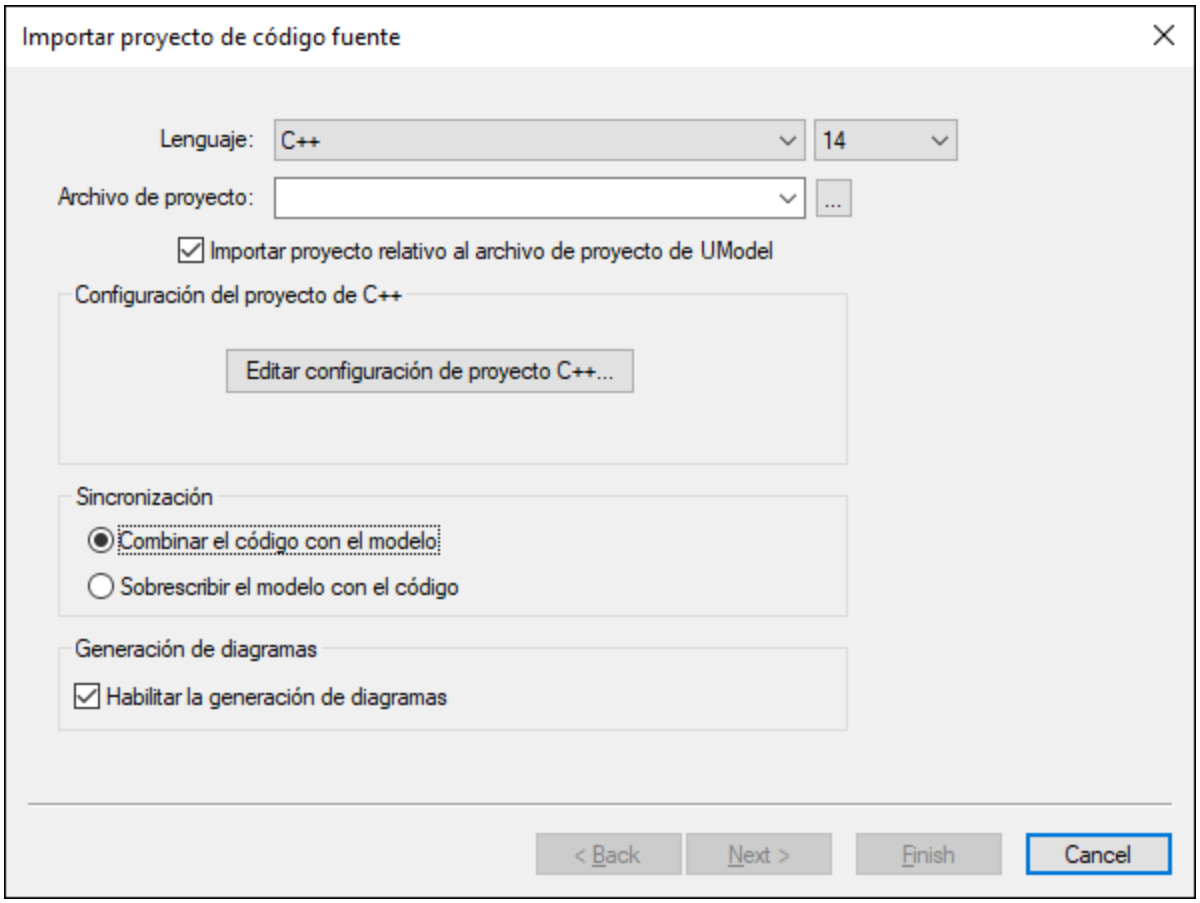
1. En el menú **Proyecto** haga clic en **Importar proyecto de código fuente** (si lo prefiere puede importar código desde un directorio con el comando **Importar directorio de código fuente**).
2. Seleccione la versión del lenguaje del proyecto de código fuente (p. ej. Java 8.0, C# 6.0 o C++14).
3. Haga clic en el botón **Examinar** ... y seleccione el archivo de proyecto de código fuente.
4. Configure las opciones de importación (*véase* *Opciones de importación de código*), que dependerán del lenguaje seleccionado en el paso nº2.
5. Haga clic en **Finalizar** para cerrar el asistente de importación.

Para ver un ejemplo más detallado consulte el apartado Ejemplo: importar un proyecto C#.

6.3.1 Ingeniería inversa en código C++

Al aplicar ingeniería inversa, los proyectos en C++ tienen un tamaño mucho mayor que los escritos en Java, C#, o VB.NET. En general se recomienda usar la función de ingeniería inversa para proyectos en C++ que sean pequeños o de tamaño mediano. Para proyectos en C++ grandes, la operación de importación podría tardar mucho (15 minutos o más).

Para importar proyectos en C++ a UModel, use el comando de menú Proyecto **| Importar proyecto de código fuente**.



Para importar proyectos en C++ escritos en un entorno de desarrollo integrado que no sea Visual Studio, use el comando de menú **Proyecto | Importar directorio de código fuente** en lugar de **Proyecto | Importar proyecto de código fuente**. Para esos proyectos debe indicar las directivas de preprocesador, las rutas y la configuración del compilador del cuadro de diálogo de importación (véase Opciones de importación de código).

Los directorios que se deben incluir para que se analicen se pueden definir a nivel del proyecto, desde Opciones de importación de código, o de forma global. Para añadir directorios de forma global, configure la variable de entorno `UMODEL_CPP_INCLUDE` como lista de directorios separados por punto y coma (;). Por ejemplo, puede añadir la ruta "C:\example\include" como sigue:

1. Abra el Panel de control y comience a teclear "variables de entorno" en el campo de búsqueda.
2. Haga clic en **Editar las variables de entorno del sistema**.

3. Haga clic en **Variables del entorno**.
4. Haga clic en **Nueva** y añada una nueva variable llamada `UMODEL_CPP_INCLUDE` y el valor `C:\example\include`
5. Haga clic en **Aceptar** para cerrar todos los cuadros de diálogo.
6. Reinicie UModel.

Para proyectos en C++ escritos con Visual Studio, las directivas de preprocesador y las rutas se detectan automáticamente desde los archivos .vcproj. Las ediciones Visual Studio 6.0 a Visual Studio 2019 son compatibles con Microsoft Visual C++ Compiler (la compatibilidad se refiere al dialecto de código usado en la fuente .cpp los archivos; debe guardar su proyecto de Visual Studio como Visual Studio 2010, 2012, 2013, 2015, 2017 y 2019 para que se pueda importar).

Tenga en cuenta que:

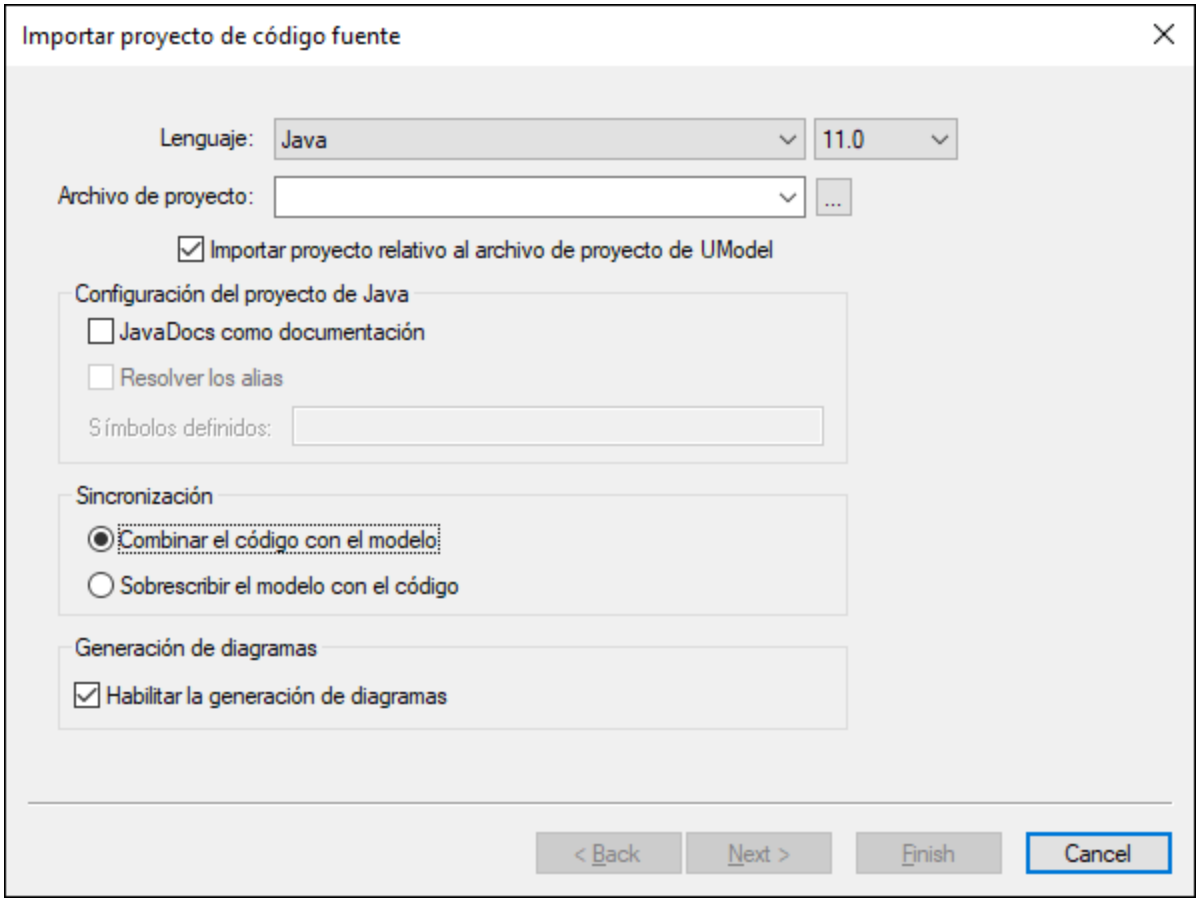
- Si UModel encuentra un tipo de datos desconocido durante el proceso de importación, la ventana Mensajes mostrará advertencias y el tipo de datos aparecerá en el modelo como `int`. Esto no ocurre en C# o Java, donde los tipos de datos desconocidos se colocan en el paquete "Elementos externos desconocidos".
- Cuando importa código C++ a UModel se añade automáticamente al proyecto un perfil para C++ integrado en UModel. El perfil incluye los tipos de datos básicos (fundamentales) para C++ y los estereotipos necesarios para la ingeniería de código, y es similar a los perfiles disponibles para otros lenguajes.

La compatibilidad con atributos C++ es limitada. Solo se reconocen los atributos integrados como `[[noreturn]]`, `[[carries_dependency]]`, `[[deprecated]]`. Los atributos personalizados (definidos por e usuario) se ignorarán.

Una vez que el código se ha importado a UModel, puede modificarlo desde el modelo y luego propagar los cambios al código (ingeniería de ida y vuelta). Como con otros lenguajes de ingeniería de código, la implementación del código fuente original (por ejemplo, cuerpos de método) no cambia después de aplicar ingeniería de ida y vuelta. Sin embargo, cualquier tipo de datos o nombres de miembros que haya cambiado en el modelo (por ejemplo, clases renombradas) se reflejarán en el código. Para obtener más información, consulte Ejemplo: generar código C++ y Sincronizar el modelo y el código fuente.

6.3.2 Opciones de importación de código

Cuando se importa código de programa en un proyecto de UModel, la aplicación ofrece varias opciones de importación (*ver más abajo*). Estas opciones se pueden configurar en el cuadro de diálogo que aparece cuando se ejecuta el comando **Proyecto | Importar proyecto de código fuente** o **Proyecto | Importar directorio de código fuente**.



Cuadro de diálogo "Importar proyecto de código fuente"

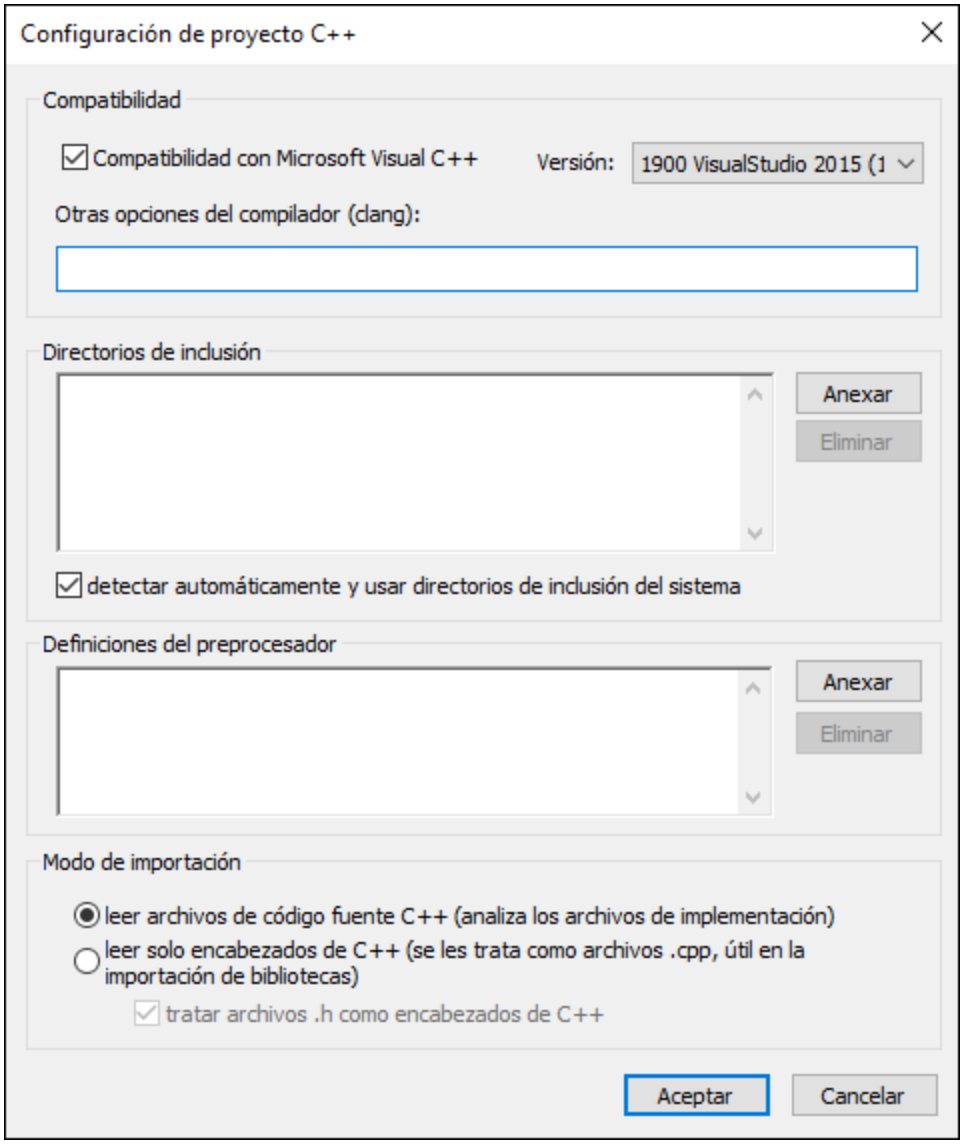
La mayoría de las opciones de este cuadro de diálogo pueden modificarse en cualquier momento, no solo durante la fase de importación de código (*véase* Configurar la sincronización del código).

Las opciones que aparecen a continuación afectan a todos los tipos de proyecto, independientemente del lenguaje o de la plataforma:

Opción	Descripción
Importar proyecto relativo al archivo de proyecto de UModel	Esta opción está marcada por defecto, lo que significa que se establecerá una dependencia de ruta relativa entre el proyecto de UModel y el proyecto de código fuente que se importa. Cuando finaliza la importación se genera automáticamente un componente UML en el proyecto de UModel (se puede ver en la ventana *Estructura del modelo* y es secundario de `Component View`). Este componente realiza las interfaces o clases a las que se debe aplicar ingeniería de código y especifica las opciones de ingeniería de código, incluida la ruta de acceso del proyecto o directorio del código fuente. Esta ruta será relativa si se marcó la

Opción	Descripción
	casilla *Importar proyecto relativo al archivo de proyecto de UModel.* Si no se marcó, será una ruta de acceso absoluta.
Combinar el código con el modelo *Sobrescribir el modelo con el código*	Si elige la opción *Combinar el código...*, los conflictos entre nombres (p. ej. entre nombres de paquetes o clases) se resolverán anexando un número al elemento que se importa. Si elige la opción *Sobrescribir el modelo...* y se dan conflictos entre nombres, el elemento importado tendrá prioridad sobre el que existe en el proyecto (es decir, el del proyecto se sobrescribe).
Habilitar la generación de diagramas	Marque esta casilla si quiere generar diagramas de clases y paquetes a partir de las clases importadas. Si marca esta casilla, el asistente de importación incluirá una pantalla más donde podrá personalizar el aspecto de los diagramas que se generen.

Las siguientes opciones solo se pueden aplicar a proyectos C++:



Cuadro de diálogo "Configuración de proyecto C++"

Opción	Descripción
Modo de importación	La opción *leer archivos de código fuente C++* analizará todos los archivos del proyecto de código fuente. Es la opción predeterminada. Si desea importar bibliotecas C++ solamente, entonces seleccione *leer solo encabezados de C++*, lo cual también acelerará la operación de importación. Los archivos `.h` se tratan por defecto como encabezados de C++. Desactive la casilla *tratar archivos .h como encabezados de C++* si el proyecto de código fuente utiliza otra extensión para los archivos de encabezado.

Opción	Descripción
Compatibilidad	Esta opción solamente es relevante cuando se importa código C++ compilado con Visual Studio y sirve para especificar la versión del compilador Microsoft Visual C++. Elija la versión del compilador (dialecto del código) que utiliza su proyecto de Visual Studio C++. Tenga en cuenta que esta opción se refiere al dialecto del código de los archivos de código fuente. El proyecto o solución de Visual Studio propiamente dicho debe estar guardado con Visual Studio 2010 o superior para que se pueda importar.
Otras opciones del compilador (clang)	Internamente UModel utiliza la versión de compilador `clang` 3.8 para leer código C++. En este cuadro de texto puede definir opciones de análisis de código adicionales. Consulte la documentación de `clang` en http://releases.llvm.org/3.8.1/tools/docs/UsersManual.html#command-line-options.
Directorios de inclusión	En este panel puede especificar en qué directorios UModel debe buscar clases C++ cuando aplique ingeniería inversa al código C++. Especificar los directorios de inclusión es opcional si el proyecto de código fuente se escribió con Visual Studio.
Definiciones del preprocesador	En este panel puede especificar las directivas previas de procesador C++ necesarias para compilar el código. Si el proyecto se escribió con Visual Studio, las directivas se detectan automáticamente.

Las opciones que aparecen a continuación solamente están disponibles para proyectos C# y VB.NET:

Opción	Descripción
DocComments como documentación	Marque esta casilla si desea convertir los comentarios detectados en el código C# en documentación de elementos de UModel (*véase Documentación*).

Resolver los alias	Esta casilla está marcada por defecto. Si su código C# o VB.NET contiene alias de espacios de nombres o de clases (ver código más abajo), recomendamos que marque esta casilla. De lo contrario, puede que durante la importación UModel no detecte automáticamente las asociaciones y dependencias que conllevan clases y espacios de nombres con alias (por lo que no estarían presentes en el modelo).
	
	`using Q = System.Collections.Generic.Queue<String>;`
	`Q myQueue;`
	
	Ejemplo de un alias en código C#
	
	Durante la importación de código fuente, los alias que puedan suponer un conflicto se añaden al paquete Elementos externos desconocidos del proyecto de UModel si su uso no está del todo claro.

Opción	Descripción
	Cuando actualice el código con el modelo (ingeniería de ida y vuelta), los alias se conservarán tal y como están en el código generado. La opción *Resolver los alias* se puede cambiar en cualquier momento y no solo durante la importación (véase *Configurar la sincronización del código*). Si habilita esta opción después de la operación de importación, UModel le pedirá que vuelva a actualizar el proyecto con el código porque esta opción también afecta a la ingeniería directa.

Símbolos definidos	Si su código C# o VB.NET incluye símbolos que están definidos por directivas previas de procesador como `#if`, `#endif`, puede hacer que UModel las tenga en cuenta durante la ingeniería inversa:
	
	`#if DEBUG`
	`static void DisplayMessage()`
	`{`
	`Console.WriteLine("Please wait...");`
	`}`
	`#endif`
	
	Ejemplo de símbolo de compilación condicional en código C#
	
	Por ejemplo, si aplica ingeniería inversa al código anterior, el método `DisplayMessage()` solo se importará en el modelo si se especificó el símbolo `DEBUG`.
	
	Para especificar símbolos de compilación condicionales, introdúzcalos en el cuadro de texto *Símbolos definidos*, separándolos con puntos y comas.
	
	Durante el proceso de ingeniería inversa, UModel mostrará todos los símbolos utilizados en la ventana *Mensajes*.

Las opciones que aparecen a continuación solamente están disponibles para proyectos Java:

Opción	Descripción
JavaDocs como documentación	Marque esta casilla si desea convertir los comentarios tipo JavaDocs detectados en el código en documentación de elementos de UModel (*véase Documentación*). **Nota:** solamente se convierten los comentarios relacionados con clases, interfaces, operaciones y propiedades Java.

6.3.3 Ejemplo: importar un proyecto C#

Este ejemplo explica cómo importar en UModel un ejemplo de solución C# creado con Visual Studio. La solución está disponible en un archivo ZIP en esta ubicación: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Tutorial\Anagram_CSharp.zip**.

No es necesario compilar la solución con Visual Studio antes de importarla, pero sí debe descomprimir el archivo **Anagram_CSharp.zip** en una carpeta.

El objetivo de este ejemplo es aplicar ingeniería inversa a la solución C# y crear un proyecto de UModel a partir de ella. Cuando importe el código elija la opción de generar diagramas de clases y de paquetes automáticamente.

Paso nº1: crear un proyecto nuevo

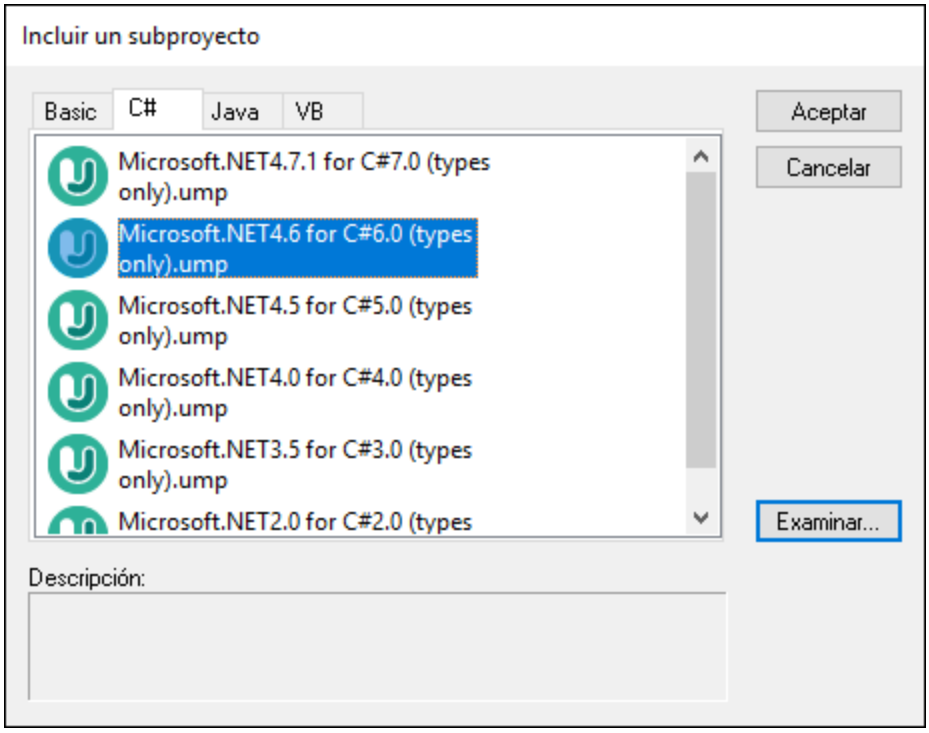
En UModel puede importar código fuente a los proyectos:

- En el menú **Archivo** haga clic en **Nuevo** (**Ctrl+N**) o haga clic en el botón **Nuevo** de la barra de herramientas.

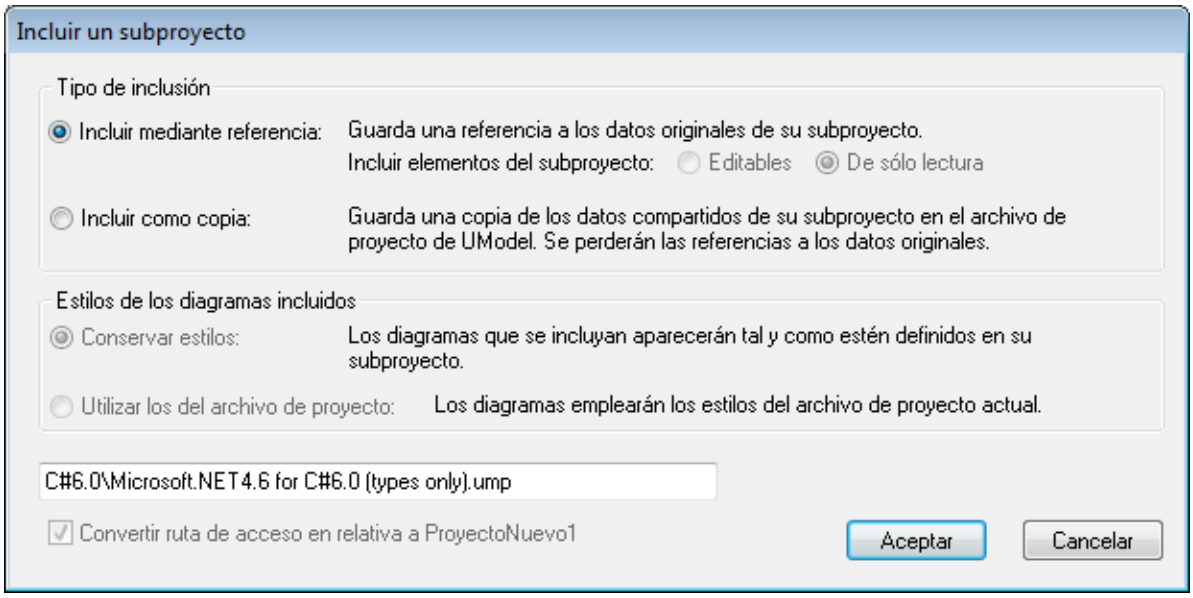
Paso nº2: incluir los tipos del lenguaje C#

El proyecto de origen se escribió en C# con Visual Studio 2015, así que incluiremos un proyecto de UModel integrado que contenga los tipos del lenguaje C# 6.0 (porque la versión de C# que corresponde a Visual Studio 2015 es la versión 6.0). Es posible que versiones anteriores de C# también funcionen con nuestro ejemplo de solución C#.

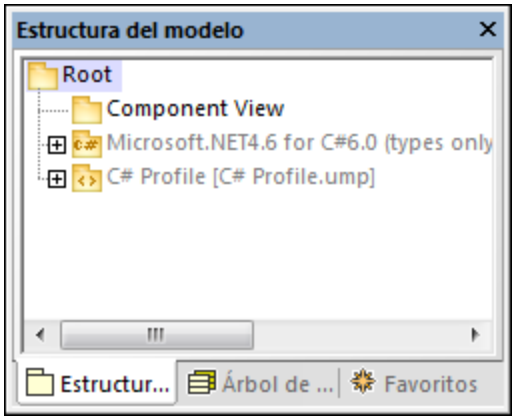
1. En el menú **Proyecto** haga clic en **Incluir un subproyecto**.
2. Haga clic en la pestaña *C#*.



3. Seleccione el proyecto **Microsoft .NET 4.6 for C# 6.0 (types only).ump** y haga clic en **Aceptar**.
4. Cuando deba elegir el tipo de inclusión (mediante referencia o como copia), deje la configuración predeterminada como está.

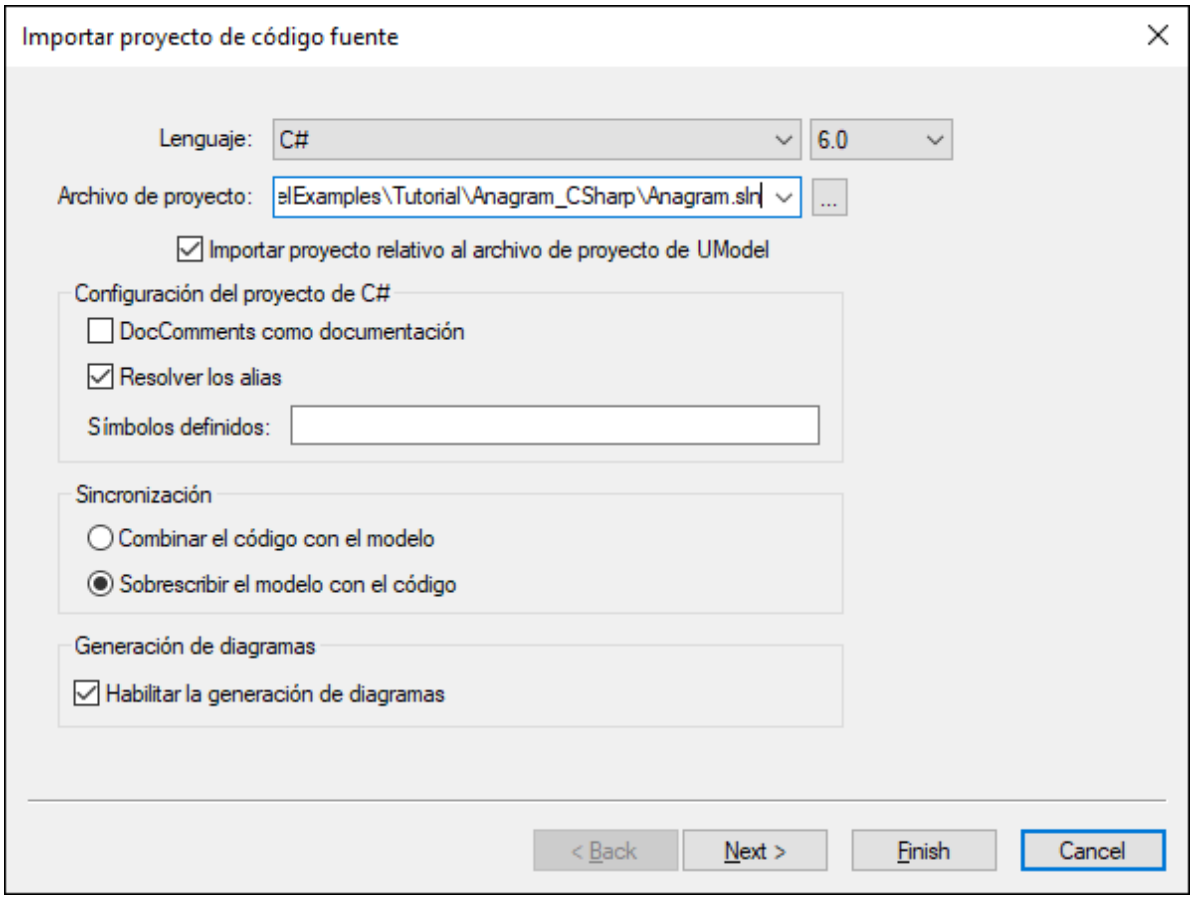


Cuando termina la operación, tanto los tipos del lenguaje C# como el perfil del lenguaje C# se incluyen en el proyecto y pueden verse en la *Estructura del modelo*.

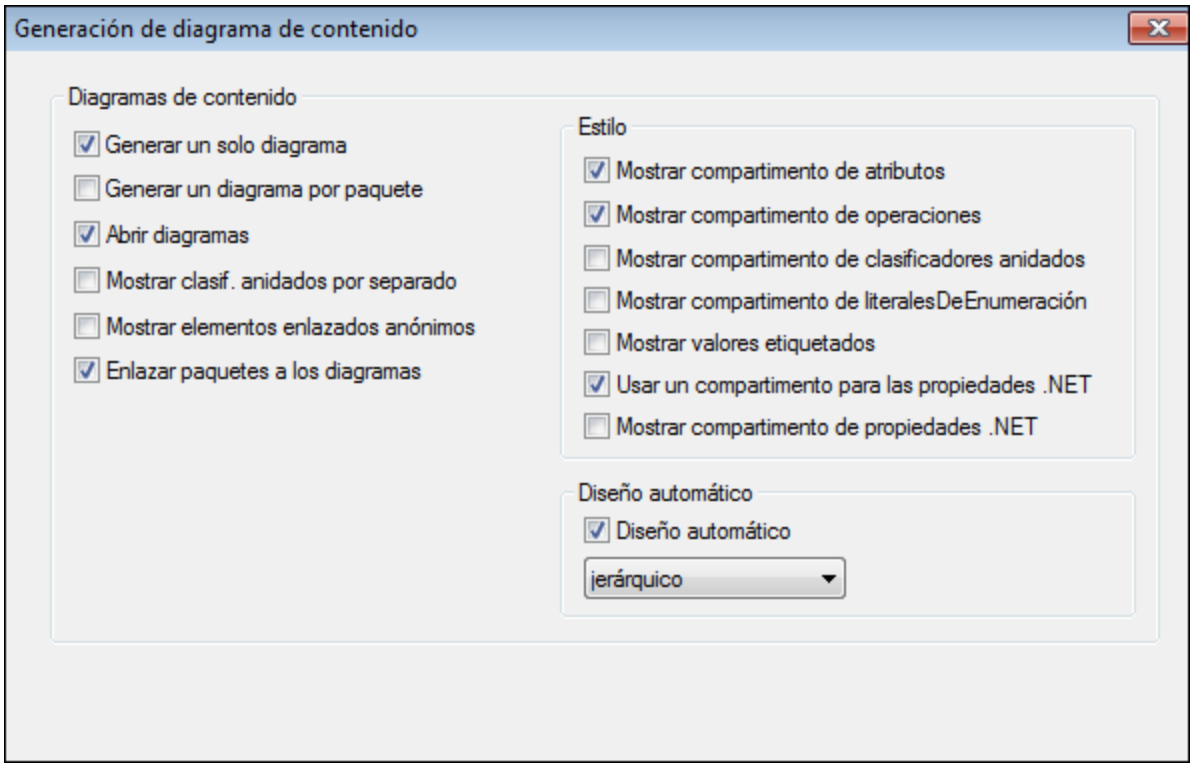


Paso nº3: importar la solución C#

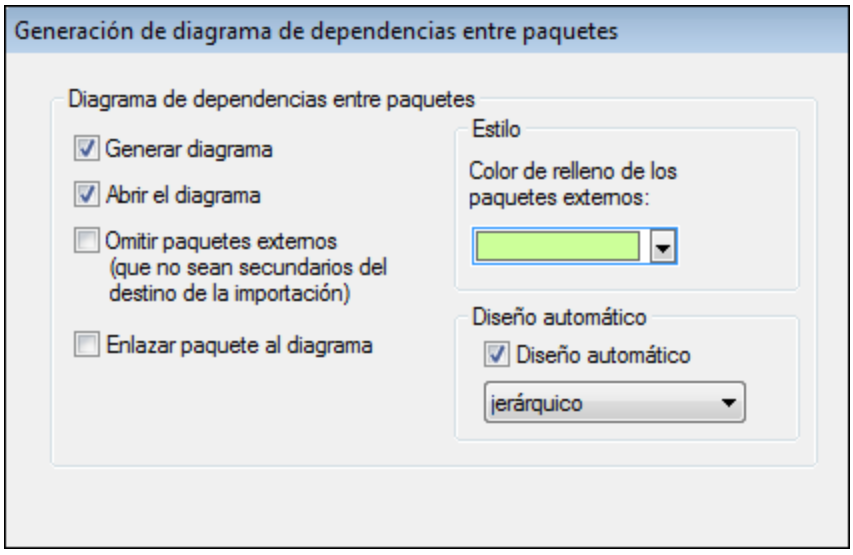
1. En el menú **Proyecto** haga clic en **Importar proyecto de código fuente**.



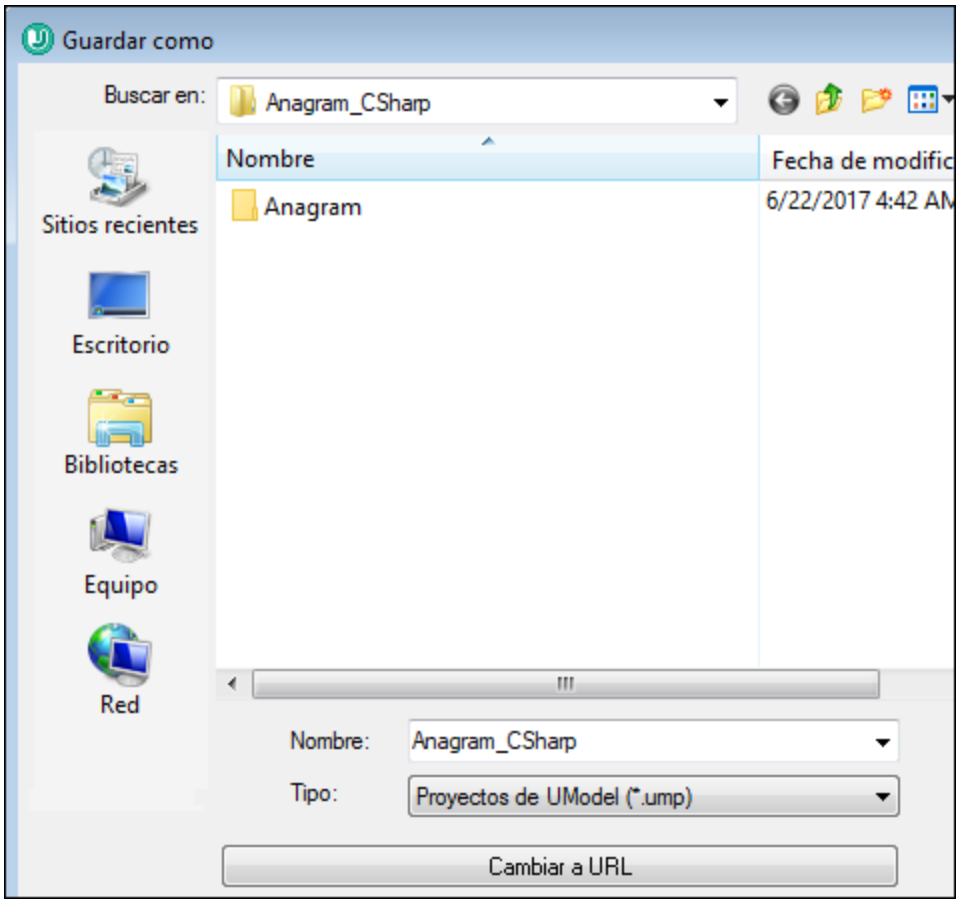
2. Seleccione el lenguaje **C# 6.0**.
3. Haga clic en el botón **Examinar** ... del campo *Archivo de proyecto* y navegue hasta el archivo de solución `.sln`.
4. Marque la casilla *DocComments como documentación* (esto importará los comentarios de código de operaciones o propiedades al modelo).
5. Como estamos importando código a un proyecto de UModel nuevo, seleccione la opción *Sobrescribir el modelo con el código* (la otra opción, *Combinar el código con el modelo*, es preferible cuando se importa código a un proyecto que ya existe).
6. Haga clic en **Siguiente**.
7. Seleccione las opciones de generación de diagramas que aparecen en la imagen siguiente y haga clic en **Siguiente**. Estas opciones afectan a los diagramas de clases que se generan automáticamente cuando se importa código.



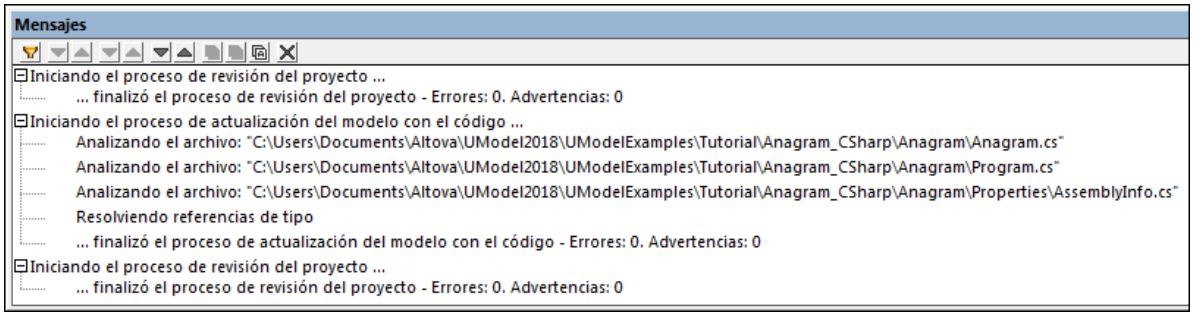
8. Seleccione las opciones de generación de diagramas que aparecen en la imagen siguiente y haga clic en **Finalizar**. Estas opciones afectan a los diagramas de paquetes que se generan automáticamente cuando se importa código.



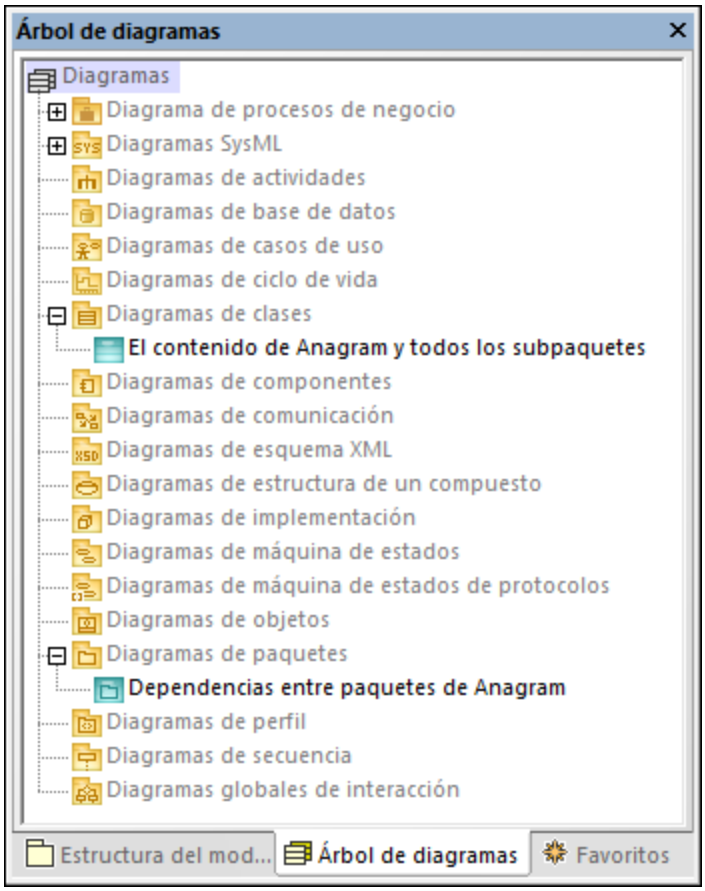
9. Introduzca un nombre, seleccione una carpeta de destino para el nuevo proyecto de UModel y haga clic en **Guardar** (este cuadro de diálogo muestra por defecto la carpeta donde está la solución que se está importando).



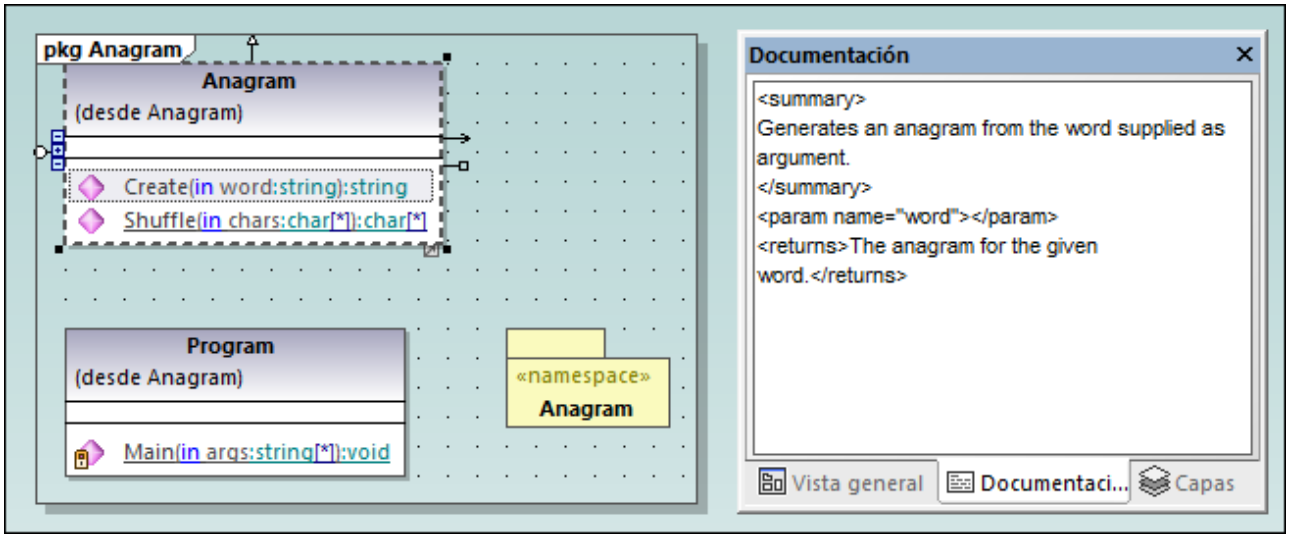
El progreso de la operación de ingeniería inversa aparece en la ventana *Mensajes*.



Cuando finaliza la importación de código, todos los diagramas generados se abren automáticamente (porque esta fue la opción seleccionada). Todos los diagramas generados aparecen en la ventana *Árbol de diagramas*:



Como elegimos la opción de generar documentación a partir del código, la documentación importada aparece en la ventana *Documentación* cuando hacemos clic en la operación `Create` de la clase `Anagram`, por ejemplo.



6.4 Importar binarios Java, C# y VB.NET

UModel permite la importación de binarios C#, Java y VB.NET, lo cual es de gran utilidad cuando se trabaja con binarios de terceros o si el código fuente original ya no está disponible. Debe tener en cuenta que:

- para importar archivos binarios Java, es necesario tener instalada una versión compatible de Java Runtime Environment (JRE) o del kit de desarrollo JDK. La importación de tipos es compatible con todos los archivos de clases Java `.class` o `.jar` que apunten a estos entornos (es decir, que cumplan la especificación de Java Virtual Machine). También se pueden importar archivos binarios que apunten a otros equipos virtuales Java como OpenJDK, SapMachine, Liberica JDK, entre otros (*véase Añadir tiempos de ejecución Java personalizados*).

- para importar archivos binarios C# o VB.NET, es necesario tener instalado .NET Framework, .NET Core, .NET 5 o .NET 6. Al importar binarios .NET se recomienda seleccionar la opción **cualquiera (usar desensamblador)** en el cuadro de diálogo de la importación. Una vez se importan los archivos, los tipos que no se reconozcan se colocan en el paquete "Elementos externos desconocidos". Para evitar que haya tipos que no se reconozcan (o para que haya menos) debe aplicar el perfil de UModel específico del lenguaje del código que esté usando (por ejemplo, el perfil ".NET Standard 2.1 Profile" para C# 8) *antes de la importación*. Consulte también Aplicar perfiles de UModel.

- no es posible importar binarios confusos.

La siguiente tabla enumera los métodos disponibles para importar tipos binarios en un proyecto de UModel.

C#, VB.NET	**Java**
Importar archivo de ensamblado (.dll, .exe)	Importar fichero de archivos de clase (.jar, .zip)
Importar ensamblado desde el Caché global de ensamblados (GAC)	Importar archivo de clase (.class) desde una carpeta raíz del paquete
Importar ensamblado desde las referencias de Visual Studio .NET	Importar archivo de clase desde una ruta de clases
	Importar archivos de clases desde Java runtime*

** No es compatible con Java 9 o versiones más recientes.*

Puede importar archivos binarios ejecutando el comando de menú **Proyecto | Importar tipos binarios**. Otra opción es generar diagramas de clases y de paquetes con UModel a partir de los tipos importados. Para ver ejemplos consulte los apartados Ejemplo: importar ensamblados del .NET GAC y Ejemplo: importar archivos Java .class.

Los archivos binarios también se pueden importar desde la línea de comandos (*véase Interfaz de la línea de comandos de UModel*) y mediante programación con la API de UModel (véase Importing Binary Types Programmatically).

Al importar archivos binarios en un proyecto de UModel puede especificar varias opciones de importación, entre otras:

- puede importar tipos con referencias, además de los tipos definidos en el archivo binario. También puede restringir la importación de tipos a paquetes Java específicos y espacios de nombres .NET.
- puede omitir los miembros de tipos al importar. Por ejemplo, puede importar clases e interfaces sin sus propiedades o métodos.
- puede importar tipos conforme a sus modificadores de accesibilidad (privados o públicos). Por ejemplo, puede importar solo clases públicas y omitir las clases privadas, protegidas e internas.

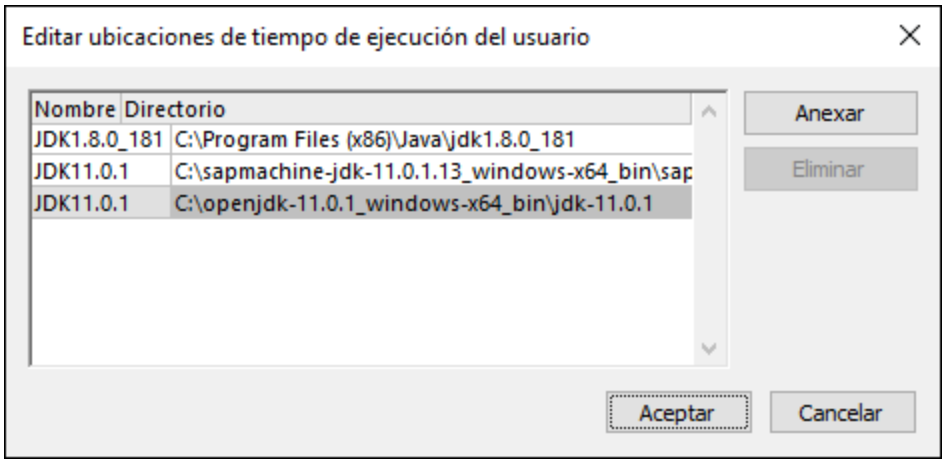
Para ver todas las opciones consulte el apartado Opciones de importación de tipos binarios.

6.4.1 Añadir tiempos de ejecución Java personalizados

Por defecto, UModel detecta los JDKs y JREs que estén instalados en el equipo local. En consecuencia, estos aparecen en la lista de tiempos de ejecución de Java cuando se inicia el asistente para importar archivos binarios. Esto ocurre con los JDKs y JREs de Oracle, que vienen con un instalador y se registran solos en el sistema al ser instalados. Sin embargo, hay otras distribuciones de máquinas virtuales Java que no tienen instalador; estas se deben añadir de forma manual a UModel. Entre ellas se encuentran Oracle OpenJDK, SapMachine, etc.

Para añadir tiempos de ejecución personalizados a UModel:

1. En el menú **Proyecto**, haga clic en **Importar tipos binarios**.
2. Seleccione **Java** como lenguaje.
3. Expanda la lista desplegable **Runtime** y seleccione **Editar ubicaciones de tiempo de ejecución java del usuario**.
4. Haga clic en **Anexar** y navegue hasta el directorio del JDK correspondiente.



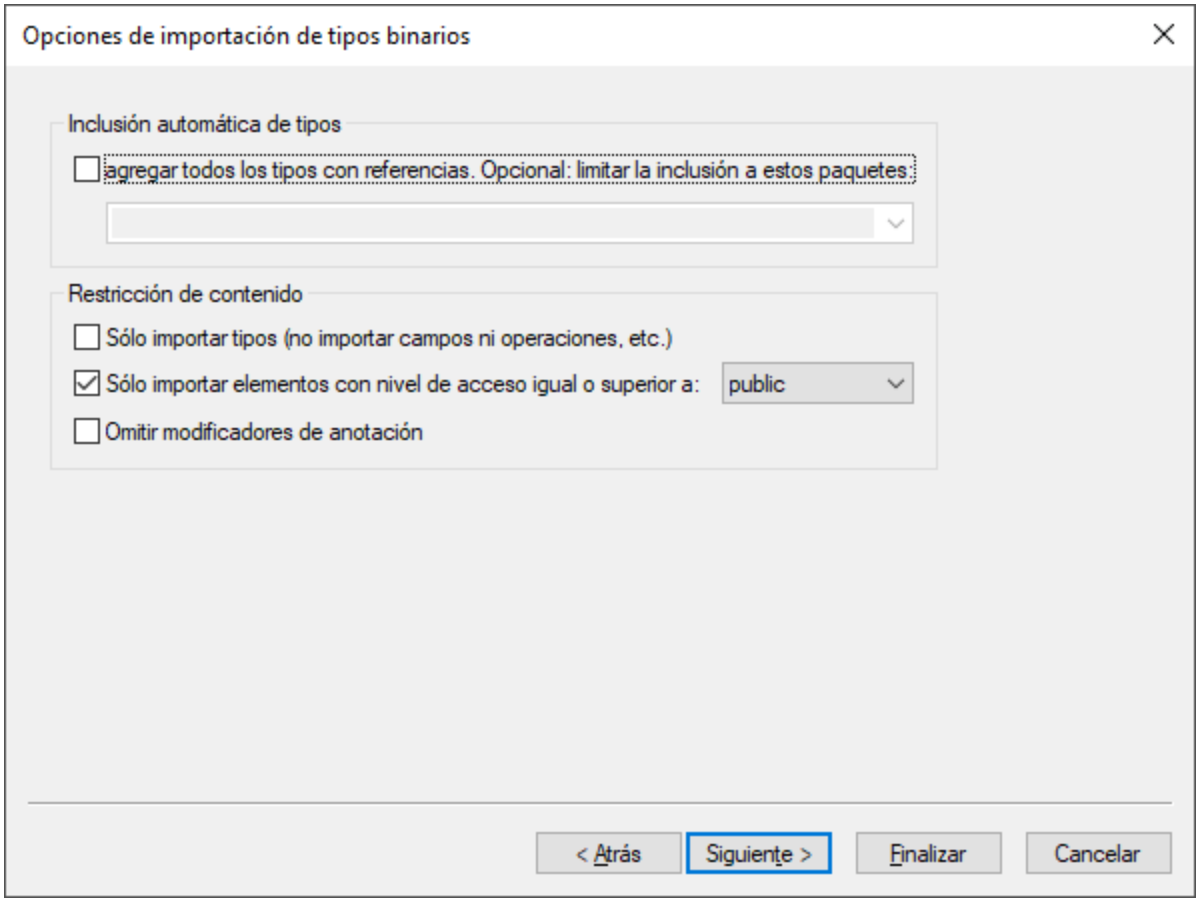
5. Haga clic en **Aceptar**.

Ahora aparecerá el tiempo de ejecución seleccionado en la lista **Runtime**, donde puede seleccionarlo siempre que necesite importar archivos binarios que apunten a ese tiempo de ejecución.

Observe que esta configuración afecta solamente a la importación de archivos binarios. Para información sobre cómo añadir una ruta de acceso a un equipo virtual java para usarlo con conectividad JDBC y generación e importación de código Java, consulte Java Virtual Machine Settings.

6.4.2 Opciones de importación de tipos binarios

Cuando importe tipos binarios en un proyecto de UModel puede que necesite configurar o cambiar las opciones que enumeramos a continuación. Estas opciones están disponibles en el cuadro de diálogo que aparece al ejecutar el comando de menú **Proyecto | Importar tipos binarios**. Tenga en cuenta que el cuadro de diálogo puede varias según si importa archivos binarios Java o .NET.



Cuadro de diálogo "Importar tipos binarios"

Inclusión automática de tipos

Los binarios .NET o Java pueden referirse a varios ensamblados o paquetes externos. Seleccione la opción **agregar todos los tipos con referencias** si quiere importar todos los tipos a los que hacen referencia los tipos incluidos en el archivo binario.

Para importar tipos con referencias solo para paquetes Java o espacios de nombres .NET específicos introduzca esos paquetes o espacios de nombres en la caja de texto adyacente. Para separar distintos paquetes o espacios de nombres, use coma, punto y coma o espacio.

Por ejemplo, imaginemos que el archivo .dll .NET de origen hace referencia a tipos de los espacios de nombres `System.Reflection` y `System.Data`. Si quiere importar tipos del espacio de nombres

`System.Reflection` pero no de `System.Data`, seleccione la opción **Agregar todos los tipos con referencias. Opcional: limitar la inclusión a estos paquetes** e introduzca "System.Reflection" en la caja de texto.

Restricción de contenido

Seleccione la opción **Solo importar tipos** para omitir miembros como campos, operaciones, propiedades, etc.

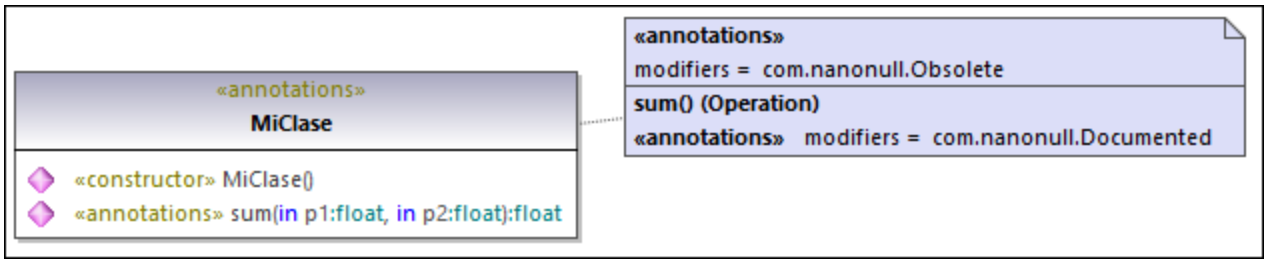
Seleccione la opción **Solo importar elementos con nivel de acceso igual o superior a** para importar tipos y miembros de tipos en función de su visibilidad. La siguiente tabla enumera la visibilidad de los tipos, empezando por los que menos tienen. Por ejemplo, si selecciona "private" se importarán todos los tipos, mientras que si selecciona "public" solo se importarán los tipos públicos y los miembros de tipos.

Nota: si la casilla no está marcada se importarán todos los tipos independientemente de su visibilidad.

.NET	Java
private	private
internal	package (default visibility when no explicit modifier exists)
protected	protected
public	public

La opción **Omitir secciones de atributo** se puede aplicar a binarios .NET. Por defecto, UModel importa los atributos C# o VB.NET que detecta en el archivo binario. Seleccione la opción **Omitir secciones de atributo** si no quiere importar atributos. De lo contrario, a los miembros que contuvieran atributos en el código fuente original se les aplicará el estereotipo `<<attributes>>` una vez haya importado el archivo binario en el modelo. Si se importan atributos puede mostrarlos en el diagrama como valores etiquetados haciendo clic con el botón derecho en la clase del diagrama y seleccionando **Mostrar valores etiquetados | Todos** en el menú contextual. Para más información consulte Estereotipos y valores etiquetados.

La opción **Omitir modificadores de anotación** se puede aplicar a binarios en Java. Por defecto, UModel importa las anotaciones que detecta en el archivo binario, siempre que su política de retención esté definida como `RUNTIME` (no `CLASS` o `SOURCE`). Si no quiere importar anotaciones, seleccione la opción **Omitir modificadores de anotación**. Si se importan anotaciones, los miembros que las contuvieran en el código fuente original tendrán el estereotipo `<<annotations>>` y las anotaciones aparecerán como valores etiquetados, como se muestra más abajo.



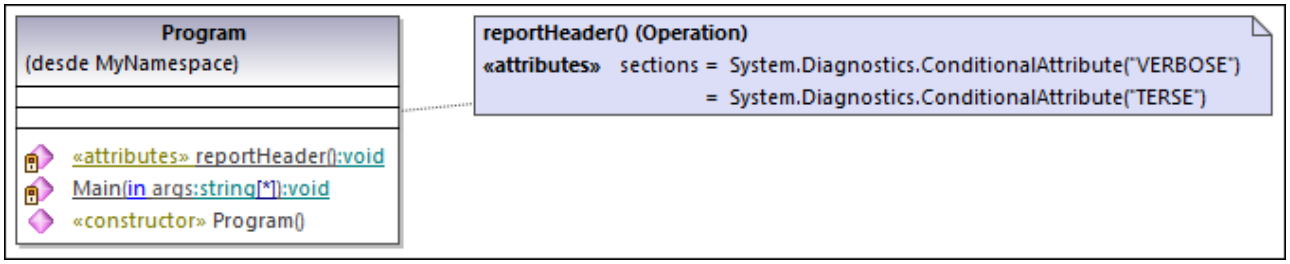
Estilos de secciones de atributos

Estas opciones se aplican únicamente a los binarios .NET. Como ya hemos mencionado, si los tipos o los miembros de tipos contenían atributos, estos se importarán como valores etiquetados en UModel.

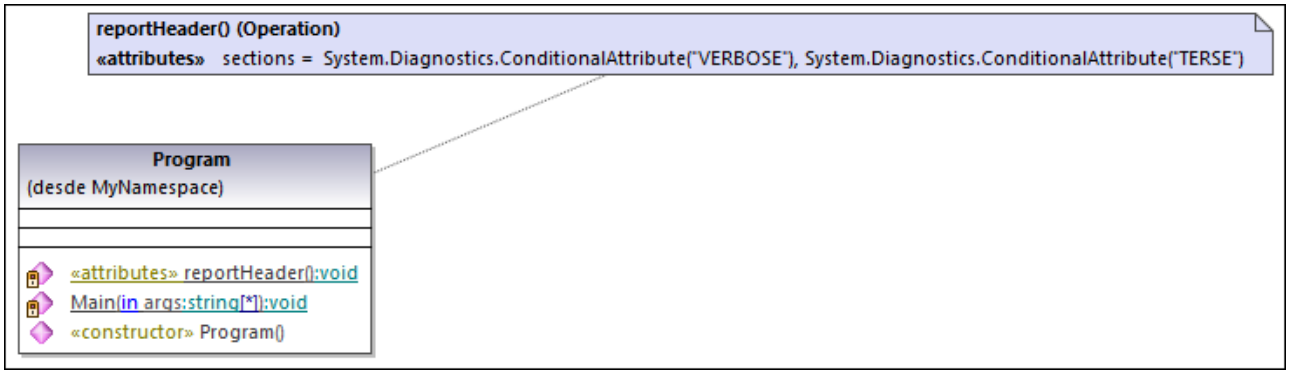
La opción **Crear un solo atributo por sección de atributos** se explica mejor con un ejemplo. Imaginemos que el código fuente original en C# definió un método con dos atributos:

```
using System;
using System.Diagnostics;

namespace MyNamespace
{
    class Program
    {
        [Conditional("VERBOSE"), Conditional("TERSE")]
        static void reportHeader()
        {
            Console.WriteLine("This is the header");
        }

        static void Main(string[] args)
        {
            reportHeader();
        }
    }
}
```

Si la opción **Crear un solo atributo por sección de atributos** está habilitada al importar el archivo binario, entonces cada atributo aparecerá en una línea aparte dentro del elemento "Valores etiquetados":



En caso contrario los atributos aparecerían separados por comas



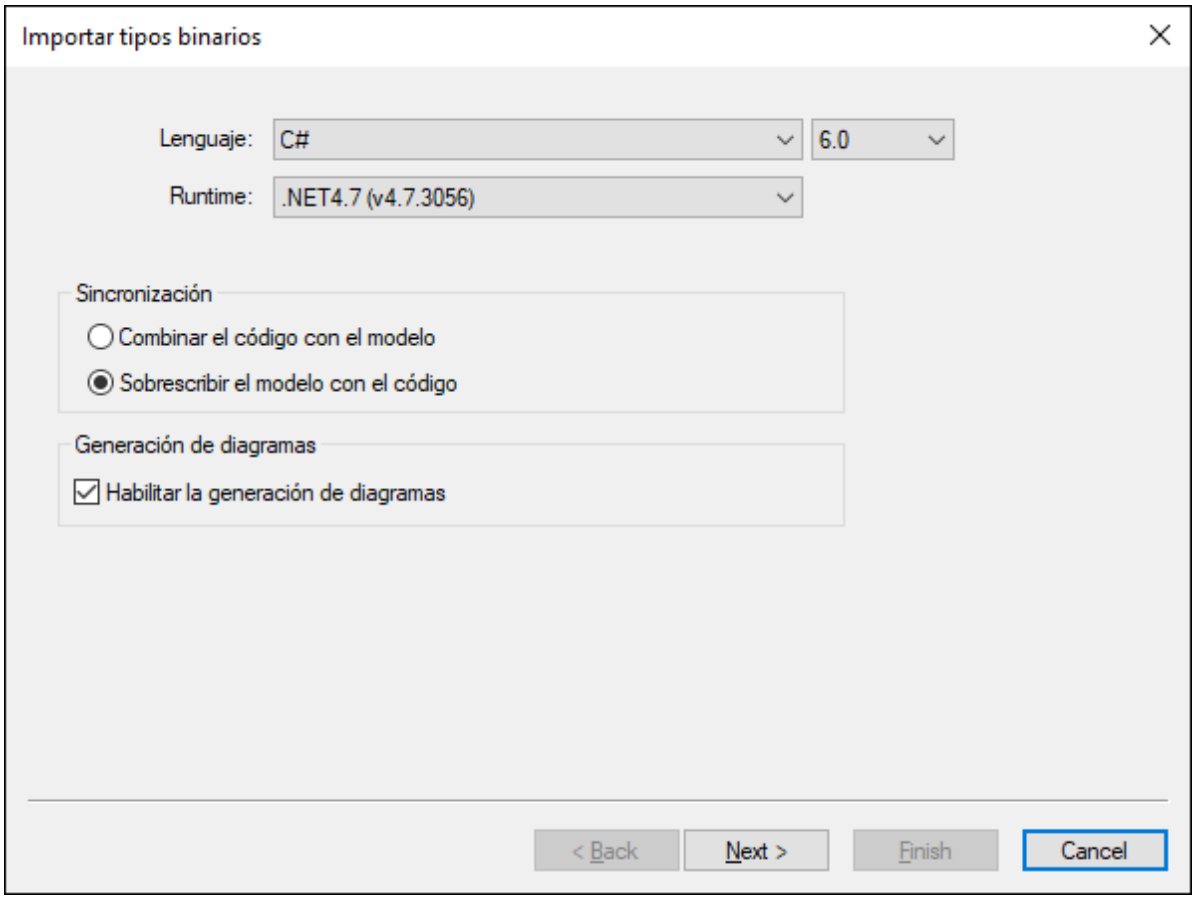
Por último, la opción **Omitir el sufijo "Attribute" en los nombres de tipo de atributo** elimina el sufijo "Atribute" de los tipos de atributos. Por ejemplo, si se selecciona esta opción, un tipo de atributo definido en el código original como `System.Xml.Serialization.XmlTypeAttribute` se importaría como `System.Xml.Serialization.XmlType`.

6.4.3 Ejemplo: importar ensamblados del .NET

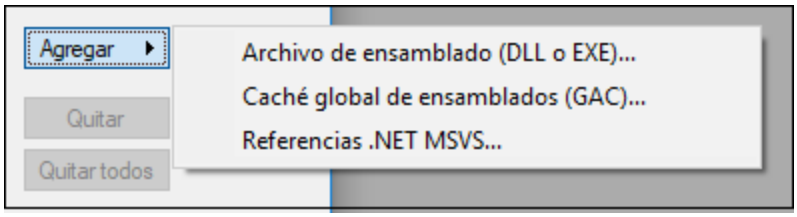
Este ejemplo explica cómo importar tipos binarios desde el Caché global de ensamblados .NET (GAC) a un proyecto de UModel en C#. Los pasos son parecidos si quiere importar tipos binarios desde un archivo independiente .dll o .exe. Para aprender a importar archivos Java `.class` consulte el apartado Ejemplo: importar archivos Java .class.

Para importar archivos binarios desde el Caché global de ensamblados .NET:

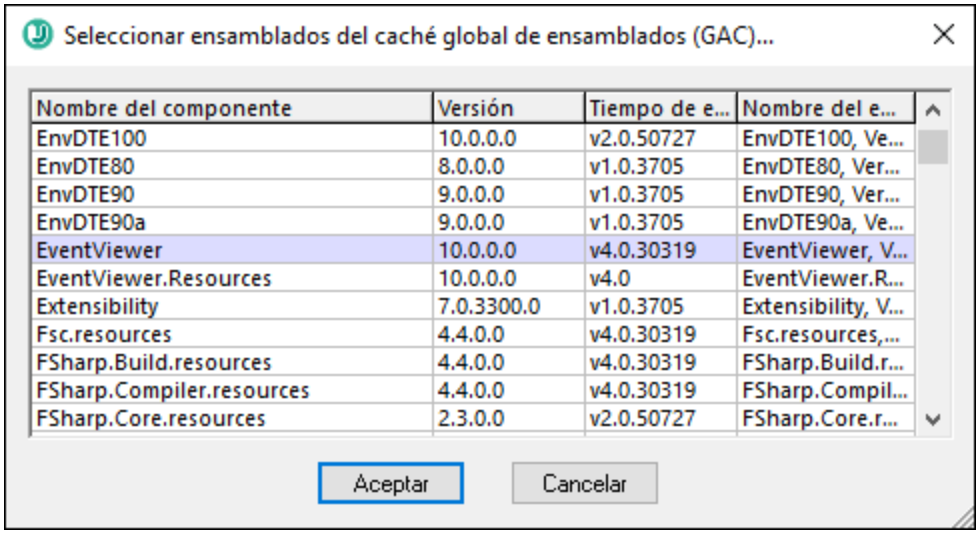
1. En el menú **Proyecto**, haga clic en **Importar tipos binarios** (*imagen siguiente*).



2. Elija el lenguaje de destino del proyecto de UModel (C#, VB.NET, Java). En este ejemplo seleccionamos C# porque estamos importando un ensamblado .NET GAC.
3. Si quiere indicar una versión específica del lenguaje para el proyecto de UModel que va a importar, selecciónela en la lista desplegable de al lado. En este ejemplo hemos seleccionado C# **7.3**.
4. También puede seleccionar una versión de .NET runtime en la lista desplegable **Runtime**. La opción predeterminada es *cualquiera (usar desensamblador)*. En este caso UModel elige la API más apropiada para el binario que se importó.
5. Si importa tipos binarios en un proyecto nuevo, seleccione **Combinar el código con el modelo** o **Sobrescribir el modelo con el código**.
6. Si quiere generar diagramas de clase y diagramas de paquetes a partir de los tipos binarios importados, marque la casilla *Habilitar la generación de diagramas*. Si lo hace habrá más diagramas disponibles en los pasos siguientes (véanse también los apartados Generar diagramas de clases y Generar diagramas de paquetes al importar código o binarios).
7. Haga clic en **Siguiente**.
8. Haga clic en **Agregar | Caché global de ensamblados (GAC)** (*imagen siguiente*). Tenga en cuenta que la opción **Caché global de ensamblados (GAC)** solo está disponible para .NET Framework 2.x-4.x. La GAC no es relevante para .NET Code, .NET 5 ni versiones posteriores. Para más información consulte la documentación de Microsoft. Para importar ensamblados para .NET Core, .NET 5 y .NET 6 debe extraer los archivos necesarios de la GAC. Después haga clic en **Agregar | Archivo de ensamblado (DLL(EXE)**, seleccione los archivos manualmente y agréguelos al proyecto.



9. Seleccione un ensamblado del cuadro de diálogo. En este ejemplo hemos seleccionado el ensamblado "EventViewer" (*imagen siguiente*).



10. Seleccione los tipos que quiere importar y haga clic en **Siguiente**. Para más información sobre otras opciones del cuadro de diálogo "Importar tipos binarios" consulte las notas de más abajo.
11. Seleccione las opciones de importación aplicables (*véase* Opciones de importación de tipos binarios).
12. Si habilitó la generación de diagramas en los pasos anteriores, haga clic en **Siguiente** y configure las opciones de la generación de diagramas. En caso contrario, haga clic en **Finalizar**.

UModel lleva a cabo la conversión y muestra un registro del progreso en la ventana *Mensajes*. Si no es posible convertir los archivos binarios, es posible que el mensaje de error dé más información. Por ejemplo, puede que el archivo binario que quiere importar esté apuntando a un tiempo de ejecución más reciente que el que está seleccionado en el cuadro de diálogo "Importar tipos binarios". En este caso seleccione una versión del tiempo de ejecución más reciente y vuelva a intentarlo.

Notas:

- La caja de texto **Reemplazo de la variable PATH...** solo se puede aplicar a Java. También puede pegar en ella las rutas de clases Java que se deban consultar además de las que se leen en la variable de entorno `CLASSPATH` (o haga clic en **Agregar** y navegue hasta las carpetas en cuestión).
- La caja de texto **usar el contexto de "sólo reflexión"** solo se puede aplicar si importa archivos binarios C# o VB.NET. Esto es útil al importar una biblioteca (dll, etc) con dependencias que no se pueden resolver o cargar. Al marcar esta casilla no se ejecutará ningún código inicializador estático, lo que provocaría errores en la importación.

6.4.4 Ejemplo: importar archivos Java .class

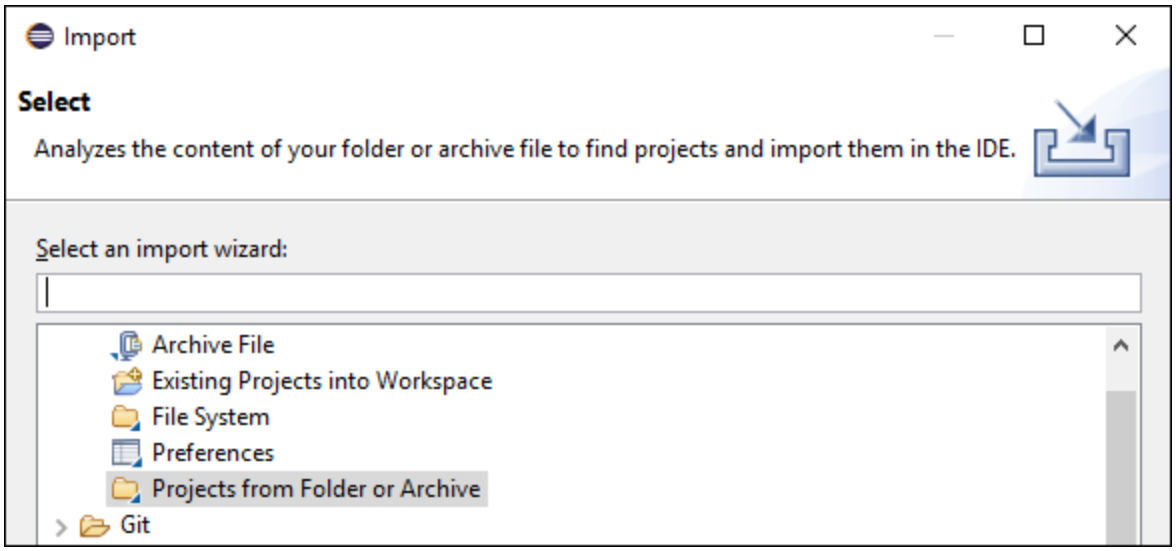
Este apartado explica cómo importar en UModel archivos Java `.class` compilados. En este ejemplo los archivos Java `.class` de origen provienen de un proyecto Java que es un tutorial creado en UModel, pero también puede usar sus propios archivos `.class` como alternativa.

Compilar código Java generado con UModel (opcional)

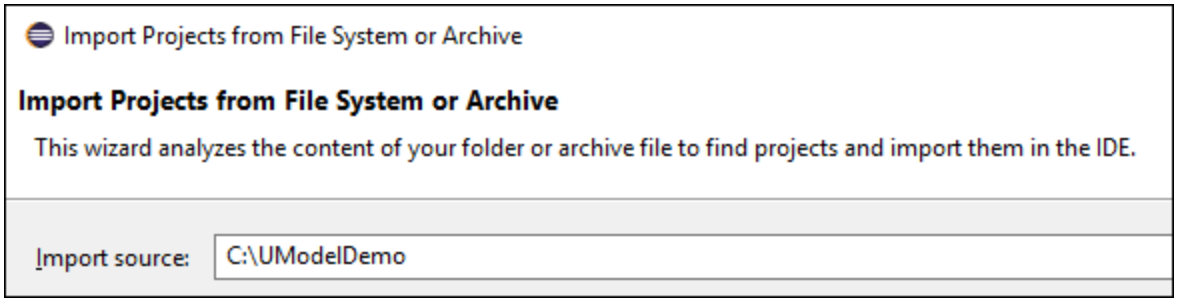
A continuación explicamos cómo usar Eclipse para compilar un proyecto Java de ejemplo generado con UModel. Observe que este paso es opcional y que el objetivo es obtener archivos `.class` compilados, por lo que puede omitir el paso si ya tiene un proyecto de Java que contenga archivos `.class`.

En este ejemplo hemos escogido Eclipse como entorno de compilación por comodidad, pero puede usar la línea de comandos de Java o cualquier otro entorno de desarrollo integrado de Java para obtener el mismo resultado.

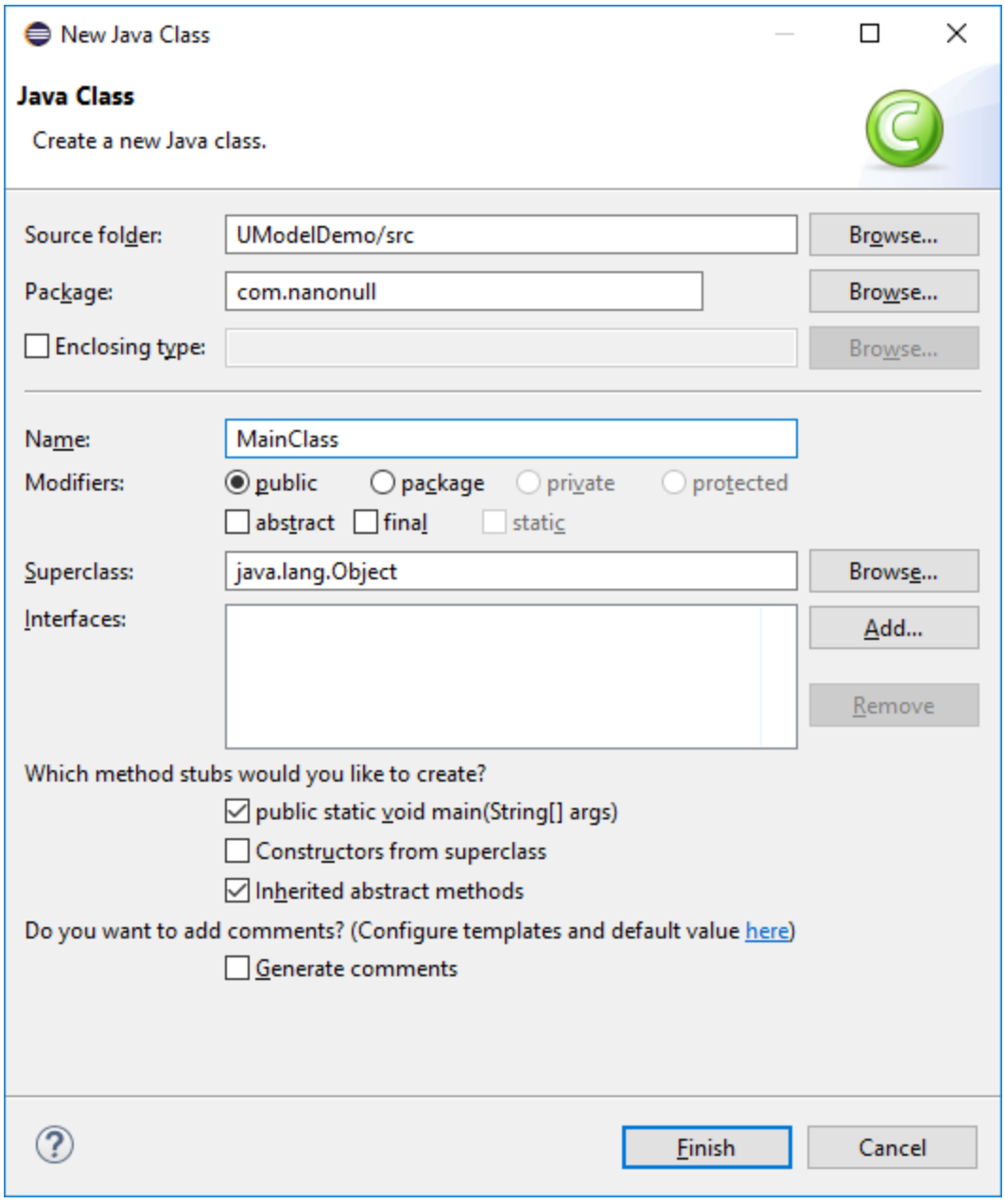
1. Si no lo ha hecho ya, cree un proyecto de Java simple con UModel, como se muestra en el apartado Ejemplo: generar código Java desde UModel. Se trata de un ejemplo muy simple que crea un paquete de Java con una única clase. Cuando complete el ejemplo encontrará el código fuente Java en el directorio **C:\UModelDemo\src**.
2. Ejecute Eclipse. En el menú **Archivo**, haga clic en **Importar**.



3. Seleccione **Projects from Folder or Archive** y haga clic en **Next**.



4. Introduzca como directorio **C:\UModelDemo** y haga clic en **Finish**.
5. Haga clic con el botón derecho en el paquete **com.nanonull** en el explorador de paquetes de Eclipse y seleccione **New | Class** del menú contextual.
6. Introduzca un nombre de clase ("MainClass" en este ejemplo) y marque la casilla **public static void main...**.



7. En el menú **Run**, haga clic en **Run**.

Con este último paso ha terminado de compilar el proyecto Java generado desde UModel. Ahora los archivos `.class` compilados deberían encontrarse en el subdirectorio **bin** del directorio de su proyecto.

Por último, tome nota de qué versión de Java se usó en la compilación (la necesitará si quiere importar tipos binarios más tarde). Por defecto, si no modificó las propiedades de su proyecto Eclipse es posible que la

versión de Java usada para compilar los archivos fuera la predeterminada que viene con Eclipse. Para ver cuál es la versión predeterminada de Java en su Eclipse siga estos pasos:

1. En el menú **Window**, haga clic en **Preferences**.
2. Haga clic en **Java** y después en **Installed JREs**.

Importar archivos Java .class

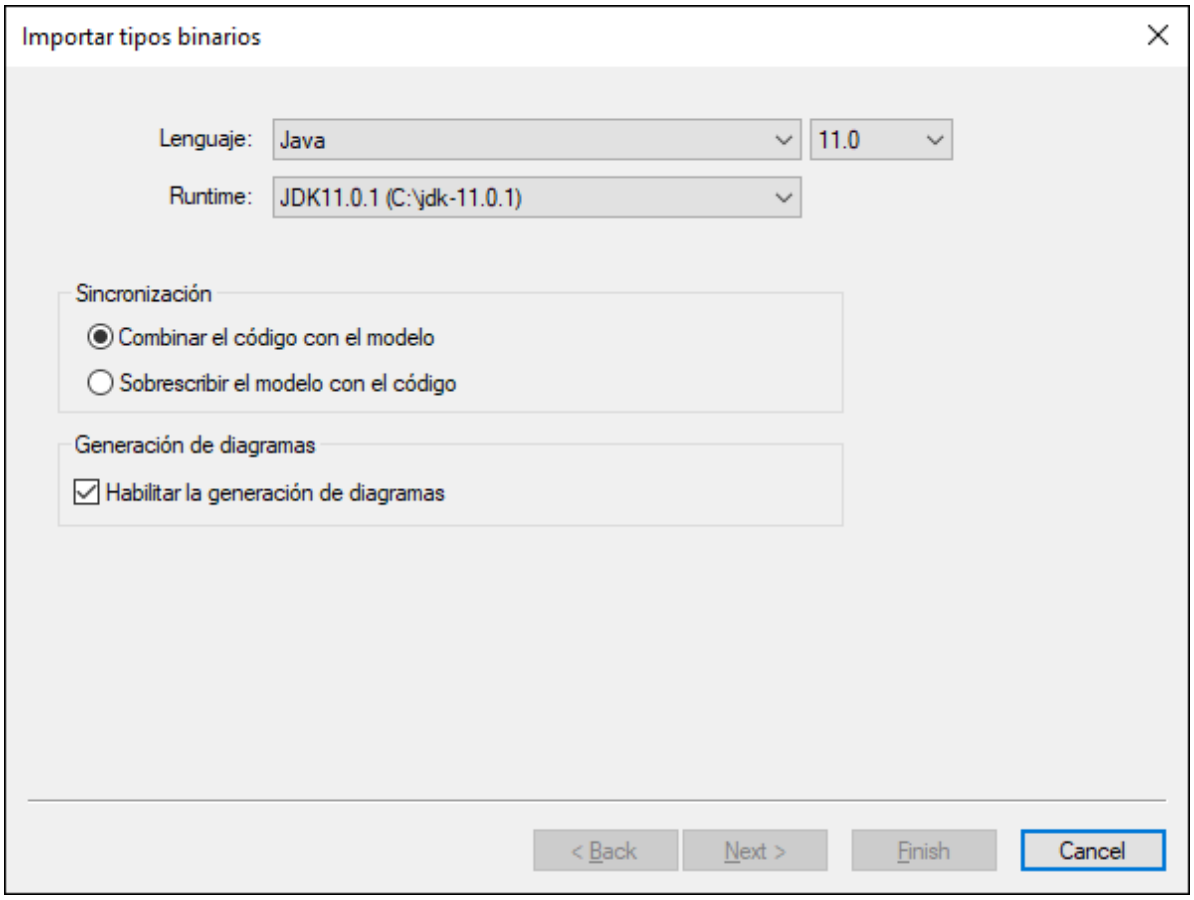
Si ya tiene archivos `.class` como los que hemos compilado en el apartado anterior, ahora puede proceder a importarlos en UModel.

1. Cree un proyecto de UModel nuevo o abra uno que ya existe. En este ejemplo vamos a importar tipos binarios en un proyecto nuevo.
2. Si su proyecto no contiene ya los tipos Java JDK, siga estos pasos:

    a. En el menú **Proyecto**, haga clic en **Incluir subproyecto**.
    b. Haga clic en la pestaña *Java* y seleccione **Java JDK (types only)**.
    c. Cuando la aplicación lo solicite, seleccione **Incluir mediante referencia**.

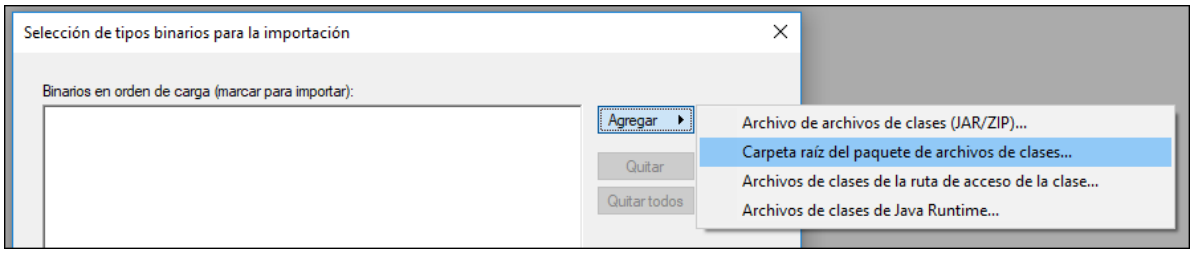
Nota: este es un paso opcional que normalmente evita que el paquete "Unknown externals" aparezca en el proyecto una vez que se haya completado la importación.

3. En el menú **Proyecto**, haga clic en **Importar tipos binarios**.
4. Seleccione **Java** como lenguaje y elija la versión de Java en que se debe compilar el código (por ejemplo, 11.0).
5. Seleccione el tiempo de ejecución que debe usar UModel para extraer la información de los archivos binarios (la llamada "reflexión"). La versión del tiempo de ejecución debe ser igual o superior a la versión de Java seleccionada en el paso anterior.

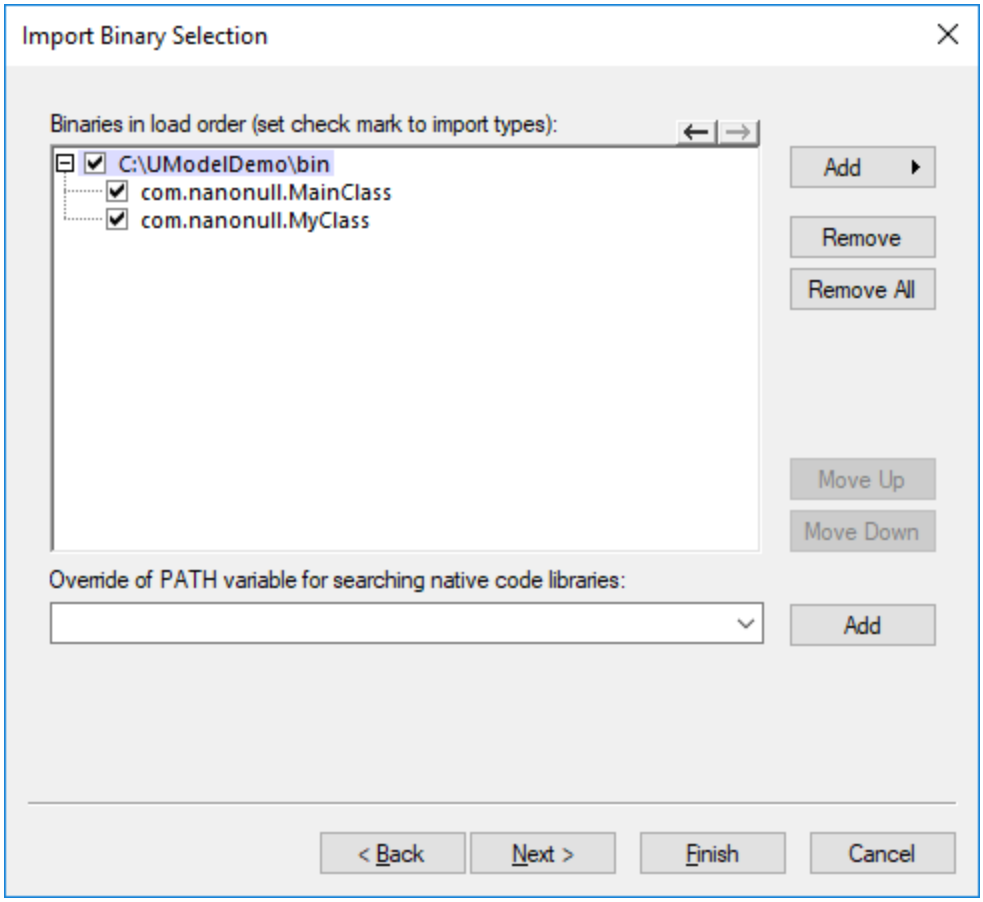


Nota: la lista desplegable **Runtime** solo contiene los JDKs y JREs que detecte automáticamente. Si su JDK o JRE no aparece en la lista, seleccione la entrada **Editar ubicaciones de tiempo de ejecución java del usuario** y navegue hasta el directorio de su equipo en el que está instalada la distribución correspondiente (véase Añadir tiempos de ejecución Java personalizados).

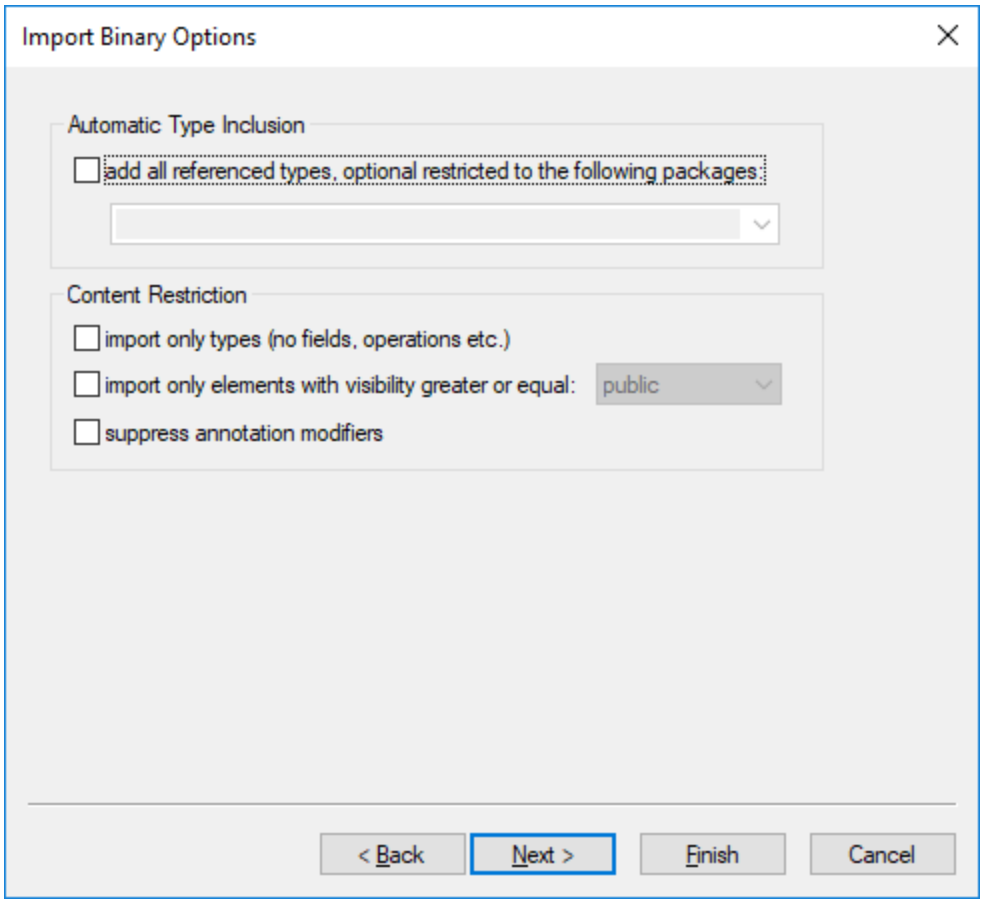
6. Si importa tipos binarios a un proyecto nuevo, seleccione **Combinar el código con el modelo** o **Sobrescribir el modelo con el código**. En caso contrario, seleccione **Combinar el código con el modelo**.
7. Si quiere generar diagramas de clase y diagramas de paquetes a partir de los tipos binarios importados, marque la casilla *Habilitar la generación de diagramas*. Si lo hace habrá más diagramas disponibles en los pasos siguientes (*véanse también los apartados Generar diagramas de clases y Generar diagramas de paquetes al importar código o binarios*).
8. Haga clic en **Siguiente**.



9. En este ejemplo vamos a importar archivos Java .class de un paquete raíz. Seleccione Agregar | Carpeta raíz del paquete de archivos de clases y navegue hasta el directorio **C:\UModelDemo\bin**. If this directory does not exist, make sure to compile the project first, as shown in the first part of this tutorial.



10. Seleccione las clases que quiere importar y haga clic en **Siguiente**.



11. Seleccione las opciones de importación aplicables (*véase* Opciones de importación de tipos binarios).
12. Si habilitó la generación de diagramas en los pasos anteriores, haga clic en **Siguiente** y configure las opciones de la generación de diagramas. En caso contrario, haga clic en **Finalizar**.

UModel lleva a cabo la conversión y muestra un registro del progreso en la ventana *Mensajes*. Si no es posible convertir los archivos binarios, es posible que el mensaje de error dé más información. Por ejemplo, puede que el archivo binario que quiere importar esté apuntando a un tiempo de ejecución más reciente que el que está seleccionado en el cuadro de diálogo "Importar tipos binarios". En este caso seleccione una versión del tiempo de ejecución más reciente y vuelva a intentarlo.

6.5 Sincronizar el modelo y el código fuente

UModel ofrece una función para sincronizar el modelo con el código y viceversa. El código se puede combinar y sincronizar por niveles, tal y como se describe a continuación (a nivel de proyecto, de paquetes o de clases).

> Cuando UModel (Enterprise o Professional) se ejecuta como complemento para Eclipse o Visual Studio, la sincronización entre código y modelo tiene lugar automáticamente. La sincronización manual solamente se puede llevar a cabo a nivel de proyecto. La opción para actualizar clases y paquetes por separado no está disponible. Para más información consulte Complemento de UModel para Visual Studio o Complemento de UModel para Eclipse.

Al hacer clic con el botón derecho dentro del árbol de diagramas (en una clase, por ejemplo), el menú contextual ofrece comandos de combinación o sincronización de código en un submenú llamado **Ingeniería de código**:

- **Combinar código de programa con *** de UModel...**
- **Combinar *** de UModel con el código de programa...**

> *** es un proyecto, un paquete, un componente, una clase, etc., dependiendo de qué tipo de elemento se seleccione en el árbol de diagramas.

Dependiendo de las opciones definidas en **Proyecto | Configurar sincronización**, estos comandos también se pueden llamar así:

- **Sobrescribir código de programa con *** de UModel...**
- **Sobrescribir *** de UModel con el código de programa...**

Para actualizar todo el proyecto (pero no las clases, los paquetes ni demás elementos locales), puede utilizar estos comandos del menú **Proyecto**:

- **Combinar (o sobrescribir) el código de programa con el proyecto de UModel**
- **Combinar (o sobrescribir) el proyecto de UModel con el código de programa**

En adelante nos referiremos a estos comandos como *comandos de sincronización de código*.

Para sincronizar el código a nivel de proyecto o paquete raíz tiene dos opciones:

- Haga clic con el botón derecho en el paquete `Raíz` del *Árbol de diagramas* y seleccione el comando de sincronización de código correspondiente.
- En el menú **Proyecto** haga clic en el comando de sincronización de código correspondiente.

Para sincronizar el código a nivel de paquete:

1. Mantenga pulsadas las teclas **Mayús** o **Ctrl** mientras hace clic en los paquetes que desea sincronizar.
2. Haga clic con el botón derecho en la selección y elija el comando de sincronización de código correspondiente.

Para sincronizar el código a nivel de clase:

1. Mantenga pulsadas las teclas **Mayús** o **Ctrl** mientras hace clic en las clases que desea sincronizar.
2. Haga clic con el botón derecho en la selección y elija el comando de sincronización de código correspondiente.

Para evitar resultados no deseados al sincronizar modelo y código debe tener en cuenta estas posibilidades:

En el menú **Proyecto**, si hace clic en **Sobrescribir proyecto de UModel con el código de programa**.	• Esto busca en todos los directorios (archivos de proyecto) de todos los lenguajes de código diferentes que están definidos en el proyecto. • En la ventana *Mensajes* aparece la entrada `Recopilando archivos fuente en...`.
Si hace clic con el botón derecho en una clase o interfaz en la ventana *Estructura del modelo* y selecciona **Ingeniería de código | Sobrescribir clase de UModel con el código de programa**.	• Esto actualiza solamente la clase seleccionada (interfaz) del proyecto. • Sin embargo, si el código fuente contiene clases que son nuevas o clases que se modificaron después de la última sincronización, estos cambios no se añadirán al modelo.
Si hace clic con el botón derecho en un componente en la ventana *Estructura del modelo* (dentro del paquete `Component View`) y selecciona y selecciona **Ingeniería de código | Sobrescribir componente de UModel con el código de programa**.	• Esto actualiza el directorio correspondiente (o archivo de proyecto) solamente. • Se identifican los archivos nuevos del directorio (archivo de proyecto) y se añaden al proyecto. • En la ventana *Mensajes* aparece la entrada `Recopilando archivos fuente en...`.

Nota: durante la sincronización de código puede recibir un mensaje solicitando que actualice el proyecto de UModel antes de iniciar la sincronización. Esto ocurre cuando se abren proyectos de UModel creados con una versión anterior a la versión más reciente de UModel. Haga clic en **Sí** para actualizar el proyecto y guardarlo en el formato más reciente. El mensaje de notificación ya no aparecerá más.

6.5.1 Consejos prácticos

Cambiar el nombre de los clasificadores y aplicar ingeniería inversa

El proceso descrito más abajo tiene lugar durante la ingeniería inversa y la sincronización automática, tanto en la versión independiente de UModel como en los complementos de UModel para Visual Studio y Eclipse.

Si cambia el nombre de un clasificador en la aplicación de programación, el clasificador se elimina o se vuelve a insertar en la ventana *Estructura del modelo* de UModel como clasificador nuevo.

El clasificador nuevo solo se vuelve a insertar en los diagramas de modelado que se crean automáticamente durante el proceso de ingeniería inversa o cuando se genera un diagrama con el comando **Mostrar en un diagrama nuevo de | Contenido**. El clasificador nuevo se inserta en una posición predeterminada en el diagrama que probablemente no coincida con su ubicación anterior.

Para más información consulte el apartado Refactorizar código y sincronización.

Generación automática de RealizacionesDeComponente

UModel puede generar `RealizacionesDeComponente` automáticamente durante el proceso de ingeniería de código. Las `RealizacionesDeComponente` solo se generan cuando está totalmente claro a qué componente se debe asignar una clase, es decir:

- Cuando solo existe un archivo de proyecto de Visual Studio en el archivo .ump.
- Cuando existen varios proyectos de Visual Studio pero sus clases están totalmente separadas en el modelo.

Para habilitar la generación automática de RealizacionesDeComponente:

1. Seleccione el comando de menú **Herramientas | Opciones**.
2. Haga clic en la pestaña *Ingeniería de código* y marque la casilla *Generar las realizacionesDeComponente que falten*.

Las `RealizacionesDeComponente` automáticas se crean para un clasificador al que solo se le puede asignar un único componente. Es decir:

- un clasificador que no tenga ninguna `RealizaciónDeComponente` o
- un clasificador que esté dentro del espacio de nombres de un lenguaje de código

Hay varias maneras de buscar componentes, dependiendo del tipo de componente.

Si se trata de componentes que representan un archivo de proyecto de código (cuando tiene definida la propiedad projectfile):

- se busca si hay UN componente que tiene/realiza clasificadores en el paquete que lo contiene.
- se busca si hay UN componente que tiene/realiza clasificadores en un subpaquete del paquete que lo contiene (de arriba a abajo).
- se busca si hay UN componente que tiene/realiza clasificadores en uno de los paquetes primarios (de abajo a arriba).
- se busca si hay UN componente que tiene realiza clasificadores en un subpaquete de uno de los paquetes primarios (de arriba a abajo).

Si se trata de componentes que representan un directorio (cuando tiene definida la propiedad directory):

- se busca si hay UN componente que tiene/realiza clasificadores en el paquete que lo contiene
- se busca si hay UN componente que tiene/realiza clasificadores en uno de los paquetes primarios (de abajo a arriba)

Notas:

- es necesario activar la opción *Generar las realizacionesDeComponente que falten* (de la pestaña *Ingeniería de código* del cuadro de diálogo "Opciones locales").
- en cuanto UModel encuentra UN componente viable durante los pasos descritos más arriba, el componente se utiliza y se omiten los pasos siguientes.

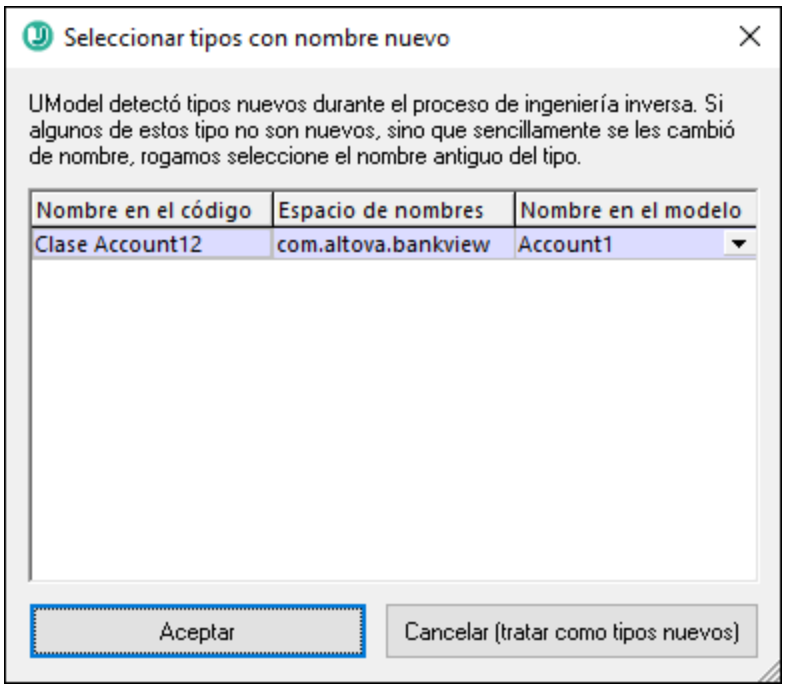
Errores/advertencias:

- si no se encuentra ningún componente viable, se genera una advertencia.

- si se encuentran varios componentes viables, se genera un error.

6.5.2 Refactorización de código y sincronización

Cuando se refactoriza código, a menudo se modifican los nombres de clase. En UModel 2009 o superior, si se detecta que durante la fase de ingeniería inversa se añadieron tipos nuevos o se cambió el nombre de algunos tipos, aparece el cuadro de diálogo "Seleccionar tipos con nombre nuevo" (*imagen siguiente*). La columna *Nombre en el código* enumera los tipos nuevos, mientras que el nombre original de cada tipo aparece en la columna *Nombre en el modelo*. UModel trata de averiguar cuál era el nombre original del tipo a partir del espacio de nombres, el contenido de la clase, las clases bases y otros datos.



Si se cambió el nombre de una clase, seleccione el nombre antiguo de la clase en la lista desplegable de la columna *Nombre en el modelo* (p. ej. C1). Esto permite conservar todos los datos relacionados y que el proceso de ingeniería de código funcione con precisión.

Cambiar el nombre de las clases en el modelo y volver a generar código

Tras crear un modelo y generar código a partir de él, si quiere puede volver a realizar cambios en el modelo antes de iniciar el proceso de sincronización.

Por ejemplo, imagine que quiere cambiar el nombre de las clases antes de generar código por segunda vez. Como previamente asignó un nombre de archivo a cada clase, en el campo nombre del archivo de código, la clase nueva y el nombre de archivo no coinciden.

Cuando inicie el proceso de sincronización, UModel le pregunta si quiere que el nombre del archivo de código coincida con el nombre de la clase nueva. Recuerde que también tiene la opción de cambiar los constructores de clase.

Ingeniería de ida y vuelta y relaciones entre elementos de modelado

Cuando se actualiza el modelo con el código, las asociaciones entre elementos de modelado aparecen en pantalla automáticamente si se marcó la opción *Crear asociaciones automáticamente* en la pestaña *Edición de diagramas* del cuadro de diálogo "Opciones" (**Herramientas | Opciones**). UModel muestra las asociaciones de los elementos que tengan configurado el tipo de los atributos y en cuyo mismo diagrama esté el elemento de modelado `type`.

Las `realizacionesDeInterfaz` y las generalizaciones aparecen automáticamente en el diagrama cuando se actualiza el modelo con el código.

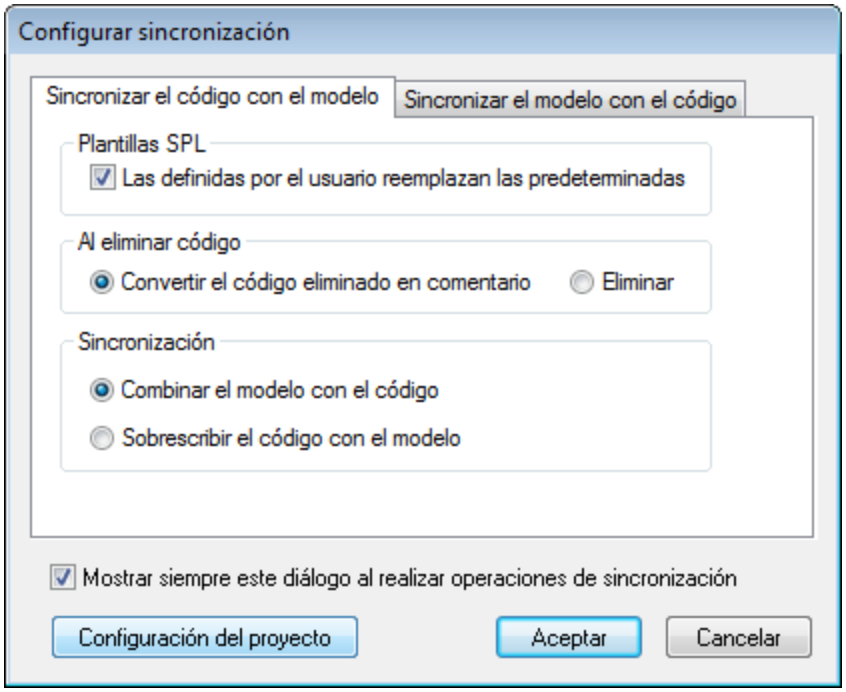
6.5.3 Configurar la sincronización del código

Las opciones de configuración para la sincronización del código son relevantes en estos casos:

- Cuando se genera código de programa a partir del modelo (es decir, cuando se ejecuta el comando **Proyecto | Combinar el código de programa con el proyecto de UModel** o el comando **Proyecto | Sobrescribir el código de programa con el proyecto de UModel**).
- Cuando se importa código fuente en el modelo (es decir, cuando se ejecuta el comando **Proyecto | Combinar el proyecto de UModel con el código de programa** o el comando **Proyecto | Sobrescribir el proyecto de UModel con el código de programa**).
- Cuando tiene lugar una sincronización automática en cualquier sentido (cuando se usa UModel Enterprise o Professional Edition como complemento de Visual Studio o Eclipse).

Para cambiar las opciones de sincronización del código:

- En el menú **Proyecto** haga clic en el comando **Configurar sincronización**.



Cuadro de diálogo "Configurar sincronización"

El cuadro de diálogo "Configurar sincronización" se abre automáticamente cada vez que se ejecuta un comando de sincronización de código. Para deshabilitar este comportamiento predeterminado desactive la casilla *Mostrar siempre este diálogo al realizar operaciones de sincronización*.

Las opciones de configuración se organizan en dos pestañas:

- *Sincronizar el código con el modelo* (las opciones de esta pestaña se aplican cuando se genera código de programa a partir del modelo)
- *Sincronizar el modelo con el código* (las opciones de esta pestaña se aplican cuando se importa código de programa en el modelo).

Opción	Descripción
Plantillas SPL	Esta opción solo se aplica cuando se genera código de programa. Marque la casilla *Las definidas por el usuario realizan las predeterminadas* si creó plantillas SPL (Spy Programming Language) personales y quiere usarlas en vez de las que vienen con UModel (*véase* Plantillas SPL).
Al eliminar código	Esta opción solo se aplica cuando se genera código de programa. Elija si al sincronizar el código el código eliminado debe eliminarse por completo o convertirse en comentarios.
Sincronización	Esta opción se aplica tanto si se genera como si se importa código de programa. Elija si los cambios deben combinarse o sobrescribirse.

Opción	Descripción
	Suponiendo que el código se generó una vez a partir de un modelo y que desde entonces se han realizado cambios tanto en el modelo como el código, por ejemplo: • en el modelo se añadió una clase nueva llamada X • en el código externo se añadió una clase nueva llamada Y. Si elije la opción ***Combinar el modelo con el código***: • la clase Y añadida en el código externo se conserva y • la clase X añadida en el modelo se añade al código. Si elije la opción ***Sobrescribir el código con el modelo***: • la clase Y añadida en el código externo se elimina (o se elimina y convierte en comentario, dependiendo de la configuración) y • la clase X añadida en el modelo se añade al código. Si elije la opción ***Combinar el código con el modelo***: • la clase X añadida en el modelo se conserva y • la clase Y añadida en el código externo se añade al modelo. Si elije la opción ***Sobrescribir el modelo con el código***: • la clase X añadida en el modelo se elimina (o se elimina y convierte en comentario, dependiendo de la configuración) y • la clase Y añadida en el código externo se añade al modelo.
Configuración del proyecto	Este botón abre el cuadro de diálogo "Configuración del proyecto", donde puede modificar la configuración de la función de ingeniería de código para cada tipo de lenguaje. Para más información consulte los apartados Opciones de importación de código y Opciones de generación de código. El cuadro de diálogo "Configuración del proyecto" también se puede abrir con el comando de menú **Proyecto | Configuración del proyecto**. Recuerde que las opciones de configuración elegidas en este cuadro de diálogo son globales (se guardan con el proyecto y se aplican sea cual sea el equipo donde se abra el proyecto de UModel), mientras que las opciones definidas con **Herramientas | Opciones** son locales (solamente afectan a la instalación actual de UModel).

6.6 Correspondencias con elementos de UModel

Esta sección muestra las correspondencias entre elementos de UModel y elementos (construcciones) de los demás lenguajes de programación (C++, C#, Java, VB.NET) y de bases de datos y esquemas XML. Se dedica un apartado a cada lenguaje de programación y las correspondencias deben tenerse en cuenta a la hora de importar código al modelo o de generar código a partir del modelo.

- Correspondencias con C++
- Correspondencias con C#
- Correspondencias con VB.NET
- Correspondencias con Java
- Correspondencias con XML Schema
- Correspondencias con elementos de BD

6.6.1 Correspondencias con C#

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de código C# cuando se genera código a partir del modelo.
- Elementos de código C# y elementos de UModel cuando se actualiza el modelo con el código.

C# Project

C#		UModel	
Proyect	projectfile	projectfile	Component
	directory	directory	

C# Namespace

C#		UModel	
Namespace	name	name	Package <<namespace>>

C# Class

C#			UModel		
Class	name		name		Class
	modifiers	internal	visibility	package	
		protected internal		protected <<internal>>	
		public		public	
		protected		protected	

C#				UModel			
		private			private		
		sealed		leaf			
		abstract		abstract			
		static		<<static>>			
		unsafe		<<unsafe>>			
		partial		<<partial>>			
		new		<<new >>			
	filename			code file name			
	associated projectfile/directory			ComponentRealization			
	base types			Generalization, InterfaceRealization(s)			
	attribute sections			<<attributes>>			
	doc comments			Comment(->Documentation)			
	Field	name		name		Property	
		modifiers	internal	visibility	package		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			static	static			
			readonly	readonly			
			volatile	<<volatile>>			
			unsafe	<<unsafe>>			
			new	<<new >>			
		type		type			
		type dimensions		multiplicity			
		type pointer		type modifier			
		nullable		<<nullable>>			
		default value		default			
		attribute sections		<<attributes>>			
		doc comments		Comment(->Documentation)			

C#				UModel			
	Constant	name		name		Property <<const>>	
		modifiers	internal	visibility	package		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			new	<<new >>			
		type		type			
		type dimensions		multiplicity			
		type pointer		type modifier			
		nullable		<<nullable>>			
		default value		default			
		attribute sections		<<attributes>>			
		doc comments		Comment(->Documentation)			
	Method	name		name		Operation	
		modifiers	internal	visibility	package		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			static	static			
			abstract	abstract			
			sealed	leaf			
			override	<<override>>			
			partial	<<partial>>			
			virtual	<<virtual>>			
			new	<<new >>			
			unsafe	<<unsafe>>			
		attribute sections		<<attributes>>			
		doc comments		Comment(->Documentation)			

C#					UModel				
		implemented interfaces			implements				
		type			direction	return	Parameter		
		Parameter	name		name				
			modifiers	ref	direction	inout			
				out		out			
				params	varArgList				
			type		type				
			type dimensions		multiplicity				
			type pointer		type modifier				
			this		<<this>>				
			nullable		<<nullable>>				
		Type Parameter	name		name		Template Parameter		
			constraint		constraining classifier				
			predefined constraint	struct	<<ValueTypeConstraint>>				
				class	<<ReferenceTypeConstraint>>				
				new ()	<<ConstructorConstraint>>				
			attribute sections		<<attributes>>				
	Constructor	name			name			Operation <<constructor>>	
		modifiers	internal		visibility	package			
			protected internal			protected <<internal>>			
			public			public			
			protected			protected			
			private			private			
			static		static				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		Parameter	name		name		Parameter		

C#					UModel				
			modifiers	ref	direction	inout			
				out		out			
				params	varArgList				
			type		type				
			type dimensions		multiplicity				
			type pointer		type modifier				
			nullable		<<nullable>>				
	Destructor	name			name			Operation <<destructor>>	
		modifiers	private		visibility	private			
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
	Property	name			name			Operation <<property>>	
		modifiers	internal		visibility	package			
			protected internal			protected <<internal>>			
			public			public			
			protected			protected			
			private			private			
			static		static				
			abstract		abstract				
			sealed		leaf				
			override		<<override>>				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		type dimensions			multiplicity				
		nullable			<<nullable>>				

C#					UModel				
		Get Accessor	modifiers	internal	visibility	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
		Set Accessor	modifiers	internal	visibility	internal	<<SetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
	Operator	name			name			Operation <<operator>>	
		modifiers	public		visibility	public			
			static		static				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		Parameter	name		name				
			modifier	params	varArgList				
			type		type				
			type dimensions		multiplicity				
			type pointer		type modifier				
			nullable		<<nullable>>				
	Indexer	name (="this")			name (="this")			Operation <<indexer>>	
		modifiers	internal		visibility	package			
			protected internal			protected <<internal>>			
			public			public			
			protected			protected			
			private			private			
			static		static				
			abstract		abstract				

C#					UModel				
			sealed		leaf				
			override		<<override>>				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		Parameter	name		name				
			modifier	params	varArgList				
			type		type				
			type dimensions		multiplicity				
			type pointer		type modifier				
			nullable		<<nullable>>				
		Get Accessor	modifiers	internal	visibility	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
		Set Accessor	modifiers	internal	visibility	internal	<<SetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
	Event	name			name			Operation <<event>>	
		modifiers	internal		visibility	package			
			protected internal			protected <<internal>>			
			public			public			
			protected			protected			
			private			private			
			static		static				

C#				UModel				
			abstract	abstract				
			sealed	leaf				
			override	<<override>>				
			virtual	<<virtual>>				
			new	<<new >>				
			unsafe	<<unsafe>>				
		attribute sections		<<attributes>>				
		doc comments		Comment(->Documentation)				
		type		direction	return	Parameter		
		type dimensions		multiplicity				
		nullable		<<nullable>>				
		Add Accessor		<<AddRemoveAccessor>>				
		Remove Accessor						
	Type Parameter	name		name			Template Parameter	
		constraint		constraining classifier				
		predefined constraint	struct	<<ValueTypeConstraint>>				
			class	<<ReferenceTypeConstraint>>				
			new ()	<<ConstructorConstraint>>				
		attribute sections		<<attributes>>				

C# Struct

C#			UModel		
Struct	name		name		Class <<struct>>
	modifiers	internal	visibility	package	
		protected internal		protected <<internal>>	
		public		public	
		protected		protected	
		private		private	
		unsafe	<<unsafe>>		
		partial	<<partial>>		

C#				UModel			
	new			<<new >>			
filename				code file name			
associated projectfile/directory				ComponentRealization			
base types				InterfaceRealization(s)			
attribute sections				<<attributes>>			
doc comments				Comment(->Documentation)			
Field	name			name		Property	
	modifiers	internal		visibility	package		
		protected internal			protected <<internal>>		
		public			public		
		protected			protected		
		private			private		
		static		static			
		readonly		readonly			
		volatile		<<volatile>>			
		unsafe		<<unsafe>>			
		new		<<new >>			
	type			type			
	type dimensions			multiplicity			
	type pointer			type modifier			
	nullable			<<nullable>>			
	default value			default			
	attribute sections			<<attributes>>			
	doc comments			Comment(->Documentation)			
Constant	name			name		Property <<const>>	
	modifiers	internal		visibility	package		
		protected internal			protected <<internal>>		
		public			public		
		protected			protected		
		private			private		

C#				UModel			
			new	<<new >>			
		type		type			
		type dimensions		multiplicity			
		type pointer		type modifier			
		nullable		<<nullable>>			
		default value		default			
		attribute sections		<<attributes>>			
		doc comments		Comment(->Documentation)			
	Fixedsize Buffer	name		name		Property <<fixed>>	
		modifiers	internal	visibility	package		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			unsafe	<<unsafe>>			
			new	<<new >>			
		type		type			
		type pointer		type modifier			
		nullable		<<nullable>>			
		buffer size		default			
		attribute sections		<<attributes>>			
		doc comments		Comment(->Documentation)			
	Method	name		name		Operation	
		modifiers	internal	visibility	package		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			static	static			
			abstract	abstract			

<table>
<tr><th colspan="5">C#</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="20"></td><td rowspan="18"></td><td rowspan="8"></td><td colspan="2">sealed</td><td colspan="3">leaf</td><td rowspan="18"></td><td rowspan="18"></td></tr>
<tr><td colspan="2">override</td><td colspan="3"><<override>></td></tr>
<tr><td colspan="2">partial</td><td colspan="3"><<partial>></td></tr>
<tr><td colspan="2">virtual</td><td colspan="3"><<virtual>></td></tr>
<tr><td colspan="2">new</td><td colspan="3"><<new >></td></tr>
<tr><td colspan="2">unsafe</td><td colspan="3"><<unsafe>></td></tr>
<tr><td colspan="3">attribute sections</td><td colspan="3"><<attributes>></td></tr>
<tr><td colspan="3">doc comments</td><td colspan="3">Comment(->Documentation)</td></tr>
<tr><td colspan="3">implemented interfaces</td><td colspan="3">implements</td></tr>
<tr><td colspan="3">type</td><td>direction</td><td>return</td><td>Parameter</td></tr>
<tr><td rowspan="8">Parameter</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="8"></td></tr>
<tr><td rowspan="3">modifiers</td><td>ref</td><td rowspan="2">direction</td><td>inout</td></tr>
<tr><td>out</td><td>out</td></tr>
<tr><td>params</td><td colspan="2">varArgList</td></tr>
<tr><td colspan="2">type</td><td colspan="2">type</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">multiplicity</td></tr>
<tr><td colspan="2">type pointer</td><td colspan="2">type modifier</td></tr>
<tr><td colspan="2">this</td><td colspan="2"><<this>></td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<nullable>></td></tr>
<tr><td rowspan="6">Type Parameter</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="6">Template Parameter</td></tr>
<tr><td colspan="2">constraint</td><td colspan="2">constraining classifier</td></tr>
<tr><td rowspan="3">predefined constraint</td><td>struct</td><td colspan="2"><<ValueTypeConstraint>></td></tr>
<tr><td>class</td><td colspan="2"><<ReferenceTypeConstraint>></td></tr>
<tr><td>new ()</td><td colspan="2"><<ConstructorConstraint>></td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="2"><<attributes>></td></tr>
<tr><td rowspan="3">Constructor</td><td colspan="3">name</td><td colspan="3">name</td><td rowspan="3">Operation <<constructor>></td><td rowspan="3"></td></tr>
<tr><td rowspan="2">modifiers</td><td colspan="2">internal</td><td rowspan="2">visibility</td><td>package</td></tr>
<tr><td colspan="2">protected internal</td><td>protected <<internal>></td></tr>
</table>

C#					UModel			
			public			public		
			protected			protected		
			private			private		
			static		static			
			unsafe		<<unsafe>>			
		attribute sections			<<attributes>>			
		doc comments			Comment(->Documentation)			
		Parameter	name		name		Parameter	
			modifiers	ref	direction	inout		
				out		out		
				params	varArgList			
			type		type			
			type dimensions		multiplicity			
			type pointer		type modifier			
			nullable		<<nullable>>			
	Destructor	name			name		Operation <<destructor>>	
		modifiers	private		visibility	private		
			unsafe		<<unsafe>>			
		attribute sections			<<attributes>>			
		doc comments			Comment(->Documentation)			
	Property	name			name		Operation <<property>>	
		modifiers	internal		visibility	package		
			protected internal			protected <<internal>>		
			public			public		
			protected			protected		
			private			private		
			static		static			
			abstract		abstract			
			sealed		leaf			
			override		<<override>>			

C#					UModel				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		type dimensions			multiplicity				
		nullable			<<nullable>>				
		Get Accessor	modifiers	internal	visibility	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
		Set Accessor	modifiers	internal	visibility	internal	<<SetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
	Operator	name			name			Operation <<operator>>	
		modifiers	public		visibility	public			
			static		static				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		Parameter	name		name				
			modifier	params	varArgList				
			type		type				
			type dimensions		multiplicity				
			type pointer		type modifier				
			nullable		<<nullable>>				

<table>
<tr><th colspan="5">C#</th><th colspan="5">UModel</th></tr>
<tr><td rowspan="20"></td><td rowspan="20">Indexer</td><td colspan="2">name (="this")</td><td></td><td colspan="2">name (="this")</td><td rowspan="20">Operation
<<indexer>></td><td rowspan="20"></td></tr>
<tr><td rowspan="5">modifiers</td><td>internal</td><td></td><td rowspan="5">visibility</td><td>package</td></tr>
<tr><td>protected internal</td><td></td><td>protected <<internal>></td></tr>
<tr><td>public</td><td></td><td>public</td></tr>
<tr><td>protected</td><td></td><td>protected</td></tr>
<tr><td>private</td><td></td><td>private</td></tr>
<tr><td colspan="2">static</td><td></td><td colspan="2">static</td></tr>
<tr><td colspan="2">abstract</td><td></td><td colspan="2">abstract</td></tr>
<tr><td colspan="2">sealed</td><td></td><td colspan="2">leaf</td></tr>
<tr><td colspan="2">override</td><td></td><td colspan="2"><<override>></td></tr>
<tr><td colspan="2">virtual</td><td></td><td colspan="2"><<virtual>></td></tr>
<tr><td colspan="2">new</td><td></td><td colspan="2"><<new >></td></tr>
<tr><td colspan="2">unsafe</td><td></td><td colspan="2"><<unsafe>></td></tr>
<tr><td colspan="2">attribute sections</td><td></td><td colspan="2"><<attributes>></td></tr>
<tr><td colspan="2">doc comments</td><td></td><td colspan="2">Comment(->Documentation)</td></tr>
<tr><td colspan="2">type</td><td></td><td>direction</td><td>return</td><td rowspan="6">Parameter</td></tr>
<tr><td rowspan="5">Parameter</td><td colspan="2">name</td><td></td><td colspan="2">name</td></tr>
<tr><td>modifier</td><td>params</td><td colspan="2">varArgList</td></tr>
<tr><td colspan="2">type</td><td colspan="2">type</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">multiplicity</td></tr>
<tr><td colspan="2">type pointer</td><td colspan="2">type modifier</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<nullable>></td></tr>
<tr><td rowspan="4">Get Accessor</td><td rowspan="4">modifiers</td><td>internal</td><td rowspan="4">visibility</td><td>internal</td><td rowspan="4"><<GetAccessor>></td></tr>
<tr><td>protected internal</td><td>protected internal</td></tr>
<tr><td>protected</td><td>protected</td></tr>
<tr><td>private</td><td>private</td></tr>
<tr><td rowspan="4">Set Accessor</td><td rowspan="4">modifiers</td><td>internal</td><td rowspan="4">visibility</td><td>internal</td><td rowspan="4"><<SetAccessor>></td></tr>
<tr><td>protected internal</td><td>protected internal</td></tr>
<tr><td>protected</td><td>protected</td></tr>
</table>

<table>
<tr><th colspan="5">C#</th><th colspan="5">UModel</th></tr>
<tr><td></td><td></td><td></td><td></td><td>private</td><td></td><td>private</td><td></td><td></td><td></td></tr>
<tr><td></td><td rowspan="16">Event</td><td colspan="2">name</td><td></td><td colspan="2">name</td><td></td><td rowspan="16">Operation
<<event>></td><td rowspan="16"></td></tr>
<tr><td rowspan="8">modifiers</td><td>internal</td><td></td><td rowspan="8">visibility</td><td>package</td><td></td></tr>
<tr><td>protected internal</td><td></td><td>protected <<internal>></td><td></td></tr>
<tr><td>public</td><td></td><td>public</td><td></td></tr>
<tr><td>protected</td><td></td><td>protected</td><td></td></tr>
<tr><td>private</td><td></td><td>private</td><td></td></tr>
<tr><td>static</td><td></td><td colspan="2">static</td><td></td></tr>
<tr><td>abstract</td><td></td><td colspan="2">abstract</td><td></td></tr>
<tr><td>sealed</td><td></td><td colspan="2">leaf</td><td></td></tr>
<tr><td>override</td><td></td><td colspan="2"><<override>></td><td></td></tr>
<tr><td>virtual</td><td></td><td colspan="2"><<virtual>></td><td></td></tr>
<tr><td>new</td><td></td><td colspan="2"><<new >></td><td></td></tr>
<tr><td>unsafe</td><td></td><td colspan="2"><<unsafe>></td><td></td></tr>
<tr><td colspan="2">attribute sections</td><td></td><td colspan="2"><<attributes>></td><td></td></tr>
<tr><td colspan="2">doc comments</td><td></td><td colspan="2">Comment(->Documentation)</td><td></td></tr>
<tr><td colspan="2">type</td><td></td><td>direction</td><td>return</td><td rowspan="3">Parameter</td><td></td></tr>
<tr><td colspan="2">type dimensions</td><td></td><td colspan="2">multiplicity</td><td></td></tr>
<tr><td colspan="2">nullable</td><td></td><td colspan="2"><<nullable>></td><td></td></tr>
<tr><td colspan="2">Add Accessor</td><td></td><td colspan="3" rowspan="2"><<AddRemoveAccessor>></td><td></td></tr>
<tr><td colspan="2">Remove Accessor</td><td></td><td></td></tr>
<tr><td rowspan="6">Type Parameter</td><td colspan="2">name</td><td></td><td colspan="2">name</td><td></td><td rowspan="6">Template Parameter</td><td rowspan="6"></td></tr>
<tr><td colspan="2">constraint</td><td></td><td colspan="2">constraining classifier</td><td></td></tr>
<tr><td rowspan="3">predefined constraint</td><td>struct</td><td></td><td colspan="2"><<ValueTypeConstraint>></td><td></td></tr>
<tr><td>class</td><td></td><td colspan="2"><<ReferenceTypeConstraint>></td><td></td></tr>
<tr><td>new ()</td><td></td><td colspan="2"><<ConstructorConstraint>></td><td></td></tr>
<tr><td>attribute sections</td><td></td><td></td><td colspan="2"><<attributes>></td><td></td></tr>
</table>

C# Interface

<table>
<tr><th colspan="5">C#</th><th colspan="5">UModel</th></tr>
<tr><td rowspan="19">Interface</td><td colspan="3">name</td><td colspan="3">name</td><td rowspan="19">Interface</td></tr>
<tr><td rowspan="6">modifiers</td><td colspan="2">internal</td><td rowspan="6">visibility</td><td colspan="2">package</td></tr>
<tr><td colspan="2">protected internal</td><td colspan="2">protected <<internal>></td></tr>
<tr><td colspan="2">public</td><td colspan="2">public</td></tr>
<tr><td colspan="2">protected</td><td colspan="2">protected</td></tr>
<tr><td colspan="2">private</td><td colspan="2">private</td></tr>
<tr><td colspan="2">unsafe</td><td colspan="3"><<unsafe>></td></tr>
<tr><td colspan="2">partial</td><td colspan="3"><<partial>></td></tr>
<tr><td colspan="2">new</td><td colspan="3"><<new >></td></tr>
<tr><td colspan="3">filename</td><td colspan="3">code file name</td></tr>
<tr><td colspan="3">associated projectfile/directory</td><td colspan="3">ComponentRealization</td></tr>
<tr><td colspan="3">base types</td><td colspan="3">Generalization(s)</td></tr>
<tr><td colspan="3">attribute sections</td><td colspan="3"><<attributes>></td></tr>
<tr><td colspan="3">doc comments</td><td colspan="3">Comment(->Documentation)</td></tr>
<tr><td rowspan="9">Method</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="9">Operation</td></tr>
<tr><td rowspan="3">modifiers</td><td>public</td><td>visibility</td><td>public</td></tr>
<tr><td>new</td><td colspan="2"><<new >></td></tr>
<tr><td>unsafe</td><td colspan="2"><<unsafe>></td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="2"><<attributes>></td></tr>
<tr><td colspan="2">doc comments</td><td colspan="2">Comment(->Documentation)</td></tr>
<tr><td colspan="2">type</td><td>direction</td><td>return</td><td rowspan="8">Parameter</td></tr>
<tr><td rowspan="7">Parameter</td><td colspan="2">name</td><td colspan="2">name</td></tr>
<tr><td rowspan="3">modifiers</td><td>ref</td><td rowspan="3">direction</td><td>inout</td></tr>
<tr><td>out</td><td>out</td></tr>
<tr><td>params</td><td colspan="2">varArgList</td></tr>
<tr><td colspan="2">type</td><td colspan="2">type</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">multiplicity</td></tr>
<tr><td colspan="2">type pointer</td><td colspan="2">type modifier</td></tr>
</table>

C#					UModel				
			this		<<this>>				
			nullable		<<nullable>>				
		Type Parameter	name		name		Template Parameter		
			constraint		constraining classifier				
			predefined constraint	struct	<<ValueTypeConstraint>>				
				class	<<ReferenceTypeConstraint>>				
				new ()	<<ConstructorConstraint>>				
			attribute sections		<<attributes>>				
	Property	name			name			Operation <<property>>	
		modifiers	public		visibility	public			
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		type dimensions			multiplicity				
		nullable			<<nullable>>				
		Get Accessor	modifiers	internal	visibility	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
		Set Accessor	modifiers	internal	visibility	internal	<<SetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
	Indexer	name (="this")			name (="this")			Operation <<indexer>>	
		modifiers	public		visibility	public			

C#					UModel				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		Parameter	name		name				
			modifier	params	varArgList				
			type		type				
			type dimensions		multiplicity				
			type pointer		type modifier				
			nullable		<<nullable>>				
		Get Accessor	modifiers	internal	visibility	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
		Set Accessor	modifiers	internal	visibility	internal	<<SetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
	Event	name			name			Operation <<event>>	
		modifiers	public		visibility	public			
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		type dimensions			multiplicity				
		nullable			<<nullable>>				
		Add Accessor			<<AddRemoveAccessor>>				

C#				UModel		
		Remove Accessor				
	Type Parameter	name		name	Template Parameter	
		constraint		constraining classifier		
		predefined constraint	struct	<<ValueTypeConstraint>>		
			class	<<ReferenceTypeConstraint>>		
			new ()	<<ConstructorConstraint>>		
		attribute sections		<<attributes>>		

C# Delegate

C#				UModel				
Delegate	name			name				Class <<delegate>>
	modifiers	internal		visibility	package			
		protected internal			protected <<internal>>			
		public			public			
		protected			protected			
		private			private			
		unsafe		<<unsafe>>				
		new		<<new >>				
	filename			code file name				
	associated projectfile/directory			ComponentRealization				
	attribute sections			<<attributes>>				
	doc comments			Comment(->Documentation)				
	type			direction	return	Parameter	Operation	
	Parameter	name		name				
		modifiers	ref	direction	inout			
			out		out			
			params	varArgList				
		type		type				
		type dimensions		multiplicity				
		type pointer		type modifier				

C#					UModel				
		nullable			<<nullable>>				
	Type Parameter	name			name			Template Parameter	
		constraint			constraining classifier				
		predefined constraint	struct		<<ValueTypeConstraint>>				
			class		<<ReferenceTypeConstraint>>				
			new ()		<<ConstructorConstraint>>				
		attribute sections			<<attributes>>				

C# Enum

C#			UModel		
Enum	name		name		Enumeration
	modifiers	internal	visibility	package	
		protected internal		protected <<internal>>	
		public		public	
		protected		protected	
		private		private	
		new	<<new >>		
	filename		code file name		
	associated projectfile/directory		ComponentRealization		
	base type		type	<<BaseType>>	
	attribute sections		<<attributes>>		
	doc comments		Comment(->Documentation)		
	Enum Constant	name	name	Enumeration Literal	
		default value	default		
		attribute sections	<<attributes>>		

C#			UModel		
		doc comments	Comment(->Documentation)		

C# Parameterized Type

C#	UModel
Parameterized Type	Anonymous Bound Element

6.6.2 Correspondencias con VB.NET

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de código VB.NET cuando se genera código a partir del modelo.
- Elementos de código VB.NET y elementos de UModel cuando se actualiza el modelo con el código.

VB.NET			UModel		
Project	projectfile		projectfile		Component
	directory		directory		
Namespace	name		name		Package <<namespace>>
Class	name		name		Class
	modifiers	Friend	visibility	package	
		Protected Friend		protected <<Friend>>	
		Public		public	
		Protected		protected	
		Private		private	
		NotInheritable	leaf		
		MustInherit	abstract		
		Partial	<<Partial>>		
		Shadows	<<Shadows>>		
	filename		code file name		
	associated projectfile/directory		ComponentRealization		
	base types		Generalization, InterfaceRealization(s)		

<table>
<tr><th colspan="4">VB.NET</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="20"></td><td colspan="3">attribute sections</td><td colspan="3"><<Attributes>></td><td rowspan="20"></td></tr>
<tr><td colspan="3">doc comments</td><td colspan="3">Comment(->Documentation)</td></tr>
<tr><td rowspan="12">Field</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="12">Property</td></tr>
<tr><td rowspan="6">modifiers</td><td>Friend</td><td rowspan="5">visibility</td><td>package</td></tr>
<tr><td>Protected Friend</td><td>protected <<Friend>></td></tr>
<tr><td>Public</td><td>public</td></tr>
<tr><td>Protected</td><td>protected</td></tr>
<tr><td>Private</td><td>private</td></tr>
<tr><td>Shared</td><td colspan="2">static</td></tr>
<tr><td>ReadOnly</td><td colspan="2">readonly</td></tr>
<tr><td>Shadows</td><td colspan="2"><<Shadows>></td></tr>
<tr><td colspan="2">type</td><td colspan="2">type</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">multiplicity</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<Nullable>></td></tr>
<tr><td colspan="2">default value</td><td colspan="2">default</td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="2"><<Attributes>></td></tr>
<tr><td colspan="2">doc comments</td><td colspan="2">Comment(->Documentation)</td></tr>
<tr><td rowspan="12">Constant</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="12">Property
<<Const>></td></tr>
<tr><td rowspan="6">modifiers</td><td>Friend</td><td rowspan="5">visibility</td><td>package</td></tr>
<tr><td>Protected Friend</td><td>protected <<Friend>></td></tr>
<tr><td>Public</td><td>public</td></tr>
<tr><td>Protected</td><td>protected</td></tr>
<tr><td>Private</td><td>private</td></tr>
<tr><td>Shadows</td><td colspan="2"><<Shadows>></td></tr>
<tr><td colspan="2">type</td><td colspan="2">type</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">multiplicity</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<Nullable>></td></tr>
<tr><td colspan="2">default value</td><td colspan="2">default</td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="2"><<Attributes>></td></tr>
<tr><td colspan="2">doc comments</td><td colspan="2">Comment(->Documentation)</td></tr>
</table>

VB.NET					UModel				
	Method	name			name			Operation	
		modifiers	Friend		visibility	package			
			Protected Friend			protected <<Friend>>			
			Public			public			
			Protected			protected			
			Private			private			
			Shared		static				
			MustOverride		abstract				
			NotOverriddable		leaf				
			Overrides		<<Overrides>>				
			Overridable		<<Overridable>>				
			Partial		<<Partial>>				
			Shadows		<<Shadows>>				
			Overloads		<<Overloads>>				
		attribute sections			<<Attributes>>				
		doc comments			Comment(->Documentation)				
		implemented interfaces			implements				
		type (function)			direction	return	Parameter		
		Parameter	name		name				
			modifiers	ByRef	direction	inout			
				ByVal		in			
				ParamArray	varArgList				
				Optional	default				
			type		type				
			type dimensions		multiplicity				
			nullable		<<Nullable>>				
		Type Parameter	name		name		Template Parameter		
			constraint		constraining classifier				
			predefined	Structure	<<ValueTypeConstraint>>				

VB.NET					UModel			
			constraint	Class	<<ReferenceTypeConstraint>>			
				New	<<ConstructorConstraint>>			
			attribute sections		<<Attributes>>			
	Constructor	name			name			Operation <<Constructor>>
		modifiers	Friend		visibility	package		
			Protected Friend			protected <<Friend>>		
			Public			public		
			Protected			protected		
			Private			private		
			Shared		static			
		attribute sections			<<Attributes>>			
		doc comments			Comment(->Documentation)			
		Parameter	name		name		Parameter	
			modifiers	ByRef	direction	inout		
				ByVal		in		
				ParamArray	varArgList			
				Optional	default			
			type		type			
			type dimensions		multiplicity			
			nullable		<<Nullable>>			
	Property	name			name			Operation <<Property>>
		modifiers	Friend		visibility	package		
			Protected Friend			protected <<Friend>>		
			Public			public		
			Protected			protected		
			Private			private		
			Default		<<Property>> (Default <= IsDefault)			
			Shared		static			

VB.NET					UModel					
			MustOverride		abstract					
			NotOverriddable		leaf					
			Overrides		<<Overrides>>					
			Overridable		<<Overridable>>					
			Shadows		<<Shadows>>					
			Overloads		<<Overloads>>					
			ReadOnly		<<GetAccessor>> (without <<SetAccessor>>)					
			WriteOnly		<<SetAccessor>> (without <<GetAccessor>>)					
		attribute sections			<<Attributes>>					
		doc comments			Comment(->Documentation)					
		type			direction	return	Parameter			
		type dimensions			multiplicity					
		nullable			<<Nullable>>					
		Get Accessor	modifiers	Friend	visibility	Friend	<<GetAccessor>>			
				Protected Friend		Protected Friend				
				Protected		Protected				
				Private		Private				
		Set Accessor	modifiers	Friend	visibility	Friend	<<SetAccessor>>			
				Protected Friend		Protected Friend				
				Protected		Protected				
				Private		Private				
	Operator	name			name			Operation <<Operator>>		
		modifiers	Public		visibility	Public				
			Shared		static					
			Narrowing		name <= Narrowing					
			Widening		name <= Widening					
		attribute sections			<<Attributes>>					
		doc comments			Comment(->Documentation)					

<table>
<tr><th colspan="4">VB.NET</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="11"></td><td rowspan="6"></td><td colspan="2">type</td><td>direction</td><td>return</td><td>Parameter</td><td rowspan="6"></td><td rowspan="11"></td></tr>
<tr><td rowspan="5">Parameter</td><td colspan="2">name</td><td colspan="3">name</td></tr>
<tr><td>modifier</td><td>ByVal</td><td>direction</td><td>in</td></tr>
<tr><td colspan="2">type</td><td colspan="3">type</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="3">multiplicity</td></tr>
<tr><td colspan="2">nullable</td><td colspan="3"><<Nullable>></td></tr>
<tr><td rowspan="16">Event</td><td colspan="2">name</td><td colspan="3">name</td><td rowspan="16">Operation
<<Event>></td></tr>
<tr><td rowspan="5">modifiers</td><td>Friend</td><td rowspan="5">visibility</td><td>package</td></tr>
<tr><td>Protected Friend</td><td>protected <<Friend>></td></tr>
<tr><td>Public</td><td>public</td></tr>
<tr><td>Protected</td><td>protected</td></tr>
<tr><td>Private</td><td>private</td></tr>
<tr><td>Shared</td><td colspan="3">static</td></tr>
<tr><td>MustOverride</td><td colspan="3">abstract</td></tr>
<tr><td>NotOverriddable</td><td colspan="3">leaf</td></tr>
<tr><td>Overrides</td><td colspan="3"><<Overrides>></td></tr>
<tr><td>Overridable</td><td colspan="3"><<Overridable>></td></tr>
<tr><td>Shadows</td><td colspan="3"><<Shadows>></td></tr>
<tr><td>Overloads</td><td colspan="3"><<Overloads>></td></tr>
<tr><td rowspan="3">kind</td><td>without specifying a delegate type</td><td colspan="3"><<Event>> (Type <= Simple)</td></tr>
<tr><td>with specifying a delegate type</td><td colspan="3"><<Event>> (Type <= Regular)</td></tr>
<tr><td>with custom accessors</td><td colspan="3"><<Event>> (Type <= Custom)</td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="3"><<Attributes>></td></tr>
<tr><td colspan="2">doc comments</td><td colspan="3">Comment(->Documentation)</td></tr>
<tr><td colspan="2">type</td><td>direction</td><td>return</td><td rowspan="3">Parameter</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">multiplicity</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<Nullable>></td></tr>
<tr><td rowspan="2">Type Parameter</td><td colspan="2">name</td><td colspan="3">name</td><td rowspan="2">Template Parameter</td></tr>
<tr><td colspan="2">constraint</td><td colspan="3">constraining classifier</td></tr>
</table>

VB.NET				UModel			
		predefined constraint	Structure	<<ValueTypeConstraint>>			
			Class	<<ReferenceTypeConstraint>>			
			New	<<ConstructorConstraint>>			
		attribute sections		<<Attributes>>			
Structure	name			name			Class <<Structure>>
	modifiers	Friend		visibility	package		
		Protected Friend			protected <<Friend>>		
		Public			public		
		Protected			protected		
		Private			private		
		Partial		<<Partial>>			
		Shadows		<<Shadows>>			
	filename			code file name			
	associated projectfile/directory			ComponentRealization			
	base types			InterfaceRealization(s)			
	attribute sections			<<Attributes>>			
	doc comments			Comment(->Documentation)			
	Field	name		name		Property	
		modifiers	Friend	visibility	package		
			Public		public		
			Private		private		
			Shared	static			
			ReadOnly	readonly			
			Shadows	<<Shadows>>			
		type		type			
		type dimensions		multiplicity			
		nullable		<<Nullable>>			
		default value		default			
		attribute sections		<<Attributes>>			
		doc comments		Comment(->Documentation)			

VB.NET				UModel			
	Constant	name		name		Property <<Const>>	
		modifiers	Friend	visibility	package		
			Public		public		
			Private		private		
			Shadows	<<Shadows>>			
		type		type			
		type dimensions		multiplicity			
		nullable		<<Nullable>>			
		default value		default			
		attribute sections		<<Attributes>>			
		doc comments		Comment(->Documentation)			
	Method	name		name		Operation	
		modifiers	Friend	visibility	package		
			Public		public		
			Private		private		
			Shared	static			
			MustOverride	abstract			
			NotOverriddable	leaf			
			Overrides	<<Overrides>>			
			Overridable	<<Overridable>>			
			Partial	<<Partial>>			
			Shadows	<<Shadows>>			
			Overloads	<<Overloads>>			
		attribute sections		<<Attributes>>			
		doc comments		Comment(->Documentation)			
		implemented interfaces		implements			
		type (function)		direction	return	Parameter	
		Parameter	name	name			
			modifiers ByRef	direction	inout		
			modifiers ByVal		in		

VB.NET					UModel			
				ParamArray	varArgList			
				Optional	default			
			type		type			
			type dimensions		multiplicity			
			nullable		<<Nullable>>			
		Type Parameter	name		name		Template Parameter	
			constraint		constraining classifier			
			predefined constraint	Structure	<<ValueTypeConstraint>>			
				Class	<<ReferenceTypeConstraint>>			
				New	<<ConstructorConstraint>>			
			attribute sections		<<Attributes>>			
	Constructor	name			name			Operation <<Constructor>>
		modifiers	Friend		visibility	package		
			Public			public		
			Private			private		
			Shared		static			
		attribute sections			<<Attributes>>			
		doc comments			Comment(->Documentation)			
		Parameter	name		name		Parameter	
			modifiers	ByRef	direction	inout		
				ByVal		in		
				ParamArray	varArgList			
				Optional	default			
			type		type			
			type dimensions		multiplicity			
			nullable		<<Nullable>>			
	Property	name			name			Operation <<Propert

VB.NET					UModel			
		modifiers	Friend		visibility	package	y>>	
			Public			public		
			Private			private		
			Shared		static			
			Default		<<Property>> (Default <= IsDefault)			
			MustOverride		abstract			
			NotOverriddable		leaf			
			Overrides		<<Overrides>>			
			Overridable		<<Overridable>>			
			Shadows		<<Shadows>>			
			Overloads		<<Overloads>>			
			ReadOnly		<<GetAccessor>> (without <<SetAccessor>>)			
			WriteOnly		<<SetAccessor>> (without <<GetAccessor>>)			
		attribute sections			<<Attributes>>			
		doc comments			Comment(->Documentation)			
		type			direction	return	Parameter	
		type dimensions			multiplicity			
		nullable			<<Nullable>>			
		Get Accessor	modifiers	Friend	visibility	Friend	<<GetAccessor>>	
				Private		Private		
		Set Accessor	modifiers	Friend	visibility	Friend	<<SetAccessor>>	
				Private		Private		
	Operator	name			name		Operation <<Operator>>	
		modifiers	Public		visibility	Public		
			Shared		static			
			Narrowing		name <= Narrowing			
			Widening		name <= Widening			
		attribute sections			<<Attributes>>			
		doc comments			Comment(->Documentation)			

VB.NET				UModel				
		type		direction	return	Parameter		
		Parameter	name	name				
			modifier ByVal	direction	in			
			type	type				
			type dimensions	multiplicity				
			nullable	<<Nullable>>				
	Event	name		name			Operation <<Event>>	
		modifiers	Friend	visibility	package			
			Public		public			
			Private		private			
			Shared	static				
			MustOverride	abstract				
			NotOverriddable	leaf				
			Overrides	<<Overrides>>				
			Overridable	<<Overridable>>				
			Shadows	<<Shadows>>				
			Overloads	<<Overloads>>				
		kind	without specifying a delegate type	<<Event>> (Type <= Simple)				
			with specifying a delegate type	<<Event>> (Type <= Regular)				
			with custom accessors	<<Event>> (Type <= Custom)				
		attribute sections		<<Attributes>>				
		doc comments		Comment(->Documentation)				
		type		direction	return	Parameter		
		type dimensions		multiplicity				
		nullable		<<Nullable>>				
	Type Parameter	name		name			Template Parameter	
		constraint		constraining classifier				
		predefined constraint	Structure	<<ValueTypeConstraint>>				
			Class	<<ReferenceTypeConstraint>>				

VB.NET				UModel		
			New	<<ConstructorConstraint>>		
		attribute sections		<<Attributes>>		
Interface	name			name		Interface
	modifiers	Friend		visibility	package	
		Protected Friend			protected <<Friend>>	
		Public			public	
		Protected			protected	
		Private			private	
		Shadows		<<Shadows>>		
	filename			code file name		
	associated projectfile/directory			ComponentRealization		
	base types			Generalization(s)		
	attribute sections			<<Attributes>>		
	doc comments			Comment(->Documentation)		
	Method	name		name		Operation
		modifiers	Public	visibility	public	
			Shadows	<<Shadows>>		
		attribute sections		<<Attributes>>		
		doc comments		Comment(->Documentation)		
		type (function)		direction	return	Parameter
		Parameter	name	name		
			modifiers ByRef	direction	inout	
			modifiers ByVal		in	
			modifiers ParamArray	varArgList		
			modifiers Optional	default		
			type	type		
			type dimensions	multiplicity		
			nullable	<<Nullable>>		
		Type Parameter	name	name		Template Parameter

VB.NET					UModel				
			constraint		constraining classifier				
			predefined constraint	Structure	<<ValueTypeConstraint>>				
				Class	<<ReferenceTypeConstraint>>				
				New	<<ConstructorConstraint>>				
			attribute sections		<<Attributes>>				
	Property	name			name			Operation <<Property>>	
		modifiers	Public		visibility	public			
			Default		<<Property>> (Default <= IsDefault)				
			Shadows		<<Shadows>>				
			ReadOnly		<<GetAccessor>> (without <<SetAccessor>>)				
			WriteOnly		<<SetAccessor>> (without <<GetAccessor>>)				
		attribute sections			<<Attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		type dimensions			multiplicity				
		nullable			<<Nullable>>				
	Event	name			name			Operation <<Event>>	
		modifiers	Public		visibility	public			
			Shadows		<<Shadows>>				
		kind	without specifying a delegate type		<<Event>> (Type <= Simple)				
			with specifying a delegate type		<<Event>> (Type <= Regular)				
		attribute sections			<<Attributes>>				
		doc comments			Comment(->Documentation)				
		type			direction	return	Parameter		
		type dimensions			multiplicity				
		nullable			<<Nullable>>				

<table>
<tr><th colspan="4">VB.NET</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="6"></td><td rowspan="6">Type Parameter</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="6">Template Parameter</td><td rowspan="6"></td></tr>
<tr><td colspan="2">constraint</td><td colspan="2">constraining classifier</td></tr>
<tr><td rowspan="3">predefined constraint</td><td>Structure</td><td colspan="2"><<ValueTypeConstraint>></td></tr>
<tr><td>Class</td><td colspan="2"><<ReferenceTypeConstraint>></td></tr>
<tr><td>New</td><td colspan="2"><<ConstructorConstraint>></td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="2"><<Attributes>></td></tr>
<tr><td rowspan="30">Delegate</td><td colspan="2">name</td><td colspan="4">name</td><td rowspan="30">Class <<Delegate>></td></tr>
<tr><td rowspan="6">modifiers</td><td colspan="2">Friend</td><td rowspan="6">visibility</td><td colspan="2">package</td></tr>
<tr><td colspan="2">Protected Friend</td><td colspan="2">protected <<Friend>></td></tr>
<tr><td colspan="2">Public</td><td colspan="2">public</td></tr>
<tr><td colspan="2">Protected</td><td colspan="2">protected</td></tr>
<tr><td colspan="2">Private</td><td colspan="2">private</td></tr>
<tr><td colspan="2">Shadows</td><td colspan="4"><<Shadows>></td></tr>
<tr><td colspan="2">filename</td><td colspan="4">code file name</td></tr>
<tr><td colspan="2">associated projectfile/directory</td><td colspan="4">ComponentRealization</td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="4"><<Attributes>></td></tr>
<tr><td colspan="2">doc comments</td><td colspan="4">Comment(->Documentation)</td></tr>
<tr><td colspan="2">type</td><td>direction</td><td>return</td><td rowspan="6">Parameter</td><td rowspan="6">Operation</td></tr>
<tr><td rowspan="6">Parameter</td><td colspan="2">name</td><td colspan="2">name</td></tr>
<tr><td rowspan="2">modifiers</td><td>ByRef</td><td rowspan="2">direction</td><td>inout</td></tr>
<tr><td>ByVal</td><td>in</td></tr>
<tr><td colspan="2">type</td><td colspan="2">type</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">multiplicity</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<Nullable>></td></tr>
<tr><td rowspan="5">Type Parameter</td><td colspan="2">name</td><td colspan="3">name</td><td rowspan="5">Template Parameter</td></tr>
<tr><td colspan="2">constraint</td><td colspan="3">constraining classifier</td></tr>
<tr><td rowspan="3">predefined constraint</td><td>struct</td><td colspan="3"><<ValueTypeConstraint>></td></tr>
<tr><td>class</td><td colspan="3"><<ReferenceTypeConstraint>></td></tr>
<tr><td>new ()</td><td colspan="3"><<ConstructorConstraint>></td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="3"><<Attributes>></td></tr>
</table>

VB.NET			UModel		
Enum	name		name		Enumeration
	modifiers	Friend	visibility	package	
		Protected Friend		protected <<Friend>>	
		Public		public	
		Protected		protected	
		Private		private	
		Shadows	<<Shadows>>		
	filename		code file name		
	associated projectfile/directory		ComponentRealization		
	base type		type	<<BaseType>>	
	attribute sections		<<Attributes>>		
	doc comments		Comment(->Documentation)		
	Enum Constant	name	name	Enumeration Literal	
		default value	default		
		attribute sections doc comments	<<Attributes>>		
			Comment(->Documentation)		
Parameterized Type			Anonymous Bound Element		

6.6.3 Correspondencias con Java

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de código Java cuando se genera código a partir del modelo.
- Elementos de código Java y elementos de UModel cuando se actualiza el modelo con el código.

Java		UModel	
Project	projectfile	projectfile	Component
	directory	directory	
Package	name	name	Package <<namespace>>
Class	name	name	Class

Java				UModel			
	modifiers	package		visibility	package		
		public			public		
		protected			protected		
		private			private		
		abstract		abstract			
		strictfp		<<strictfp>>			
		final		<<final>>			
	filename			code file name			
	associated projectfile/directory			ComponentRealization			
	extends clause			Generalization			
	implements clause			InterfaceRealization(s)			
	java docs			Comment(->Documentation)			
	Field	name		name		Property	
		modifiers	package	visibility	package		
			public		public		
			protected		protected		
			private		private		
			static	static			
			transient	<<transient>>			
			volatile	<<volatile>>			
			final	<<final>>			
		type		type			
		type dimensions		multiplicity			
		default value		default			
		java docs		Comment(->Documentation)			
	Method	name		name		Operation	
		modifiers	package	visibility	package		
			public		public		
			protected		protected		
			private		private		

Java					UModel				
			static		static				
			abstract		abstract				
			final		<<final>>				
			native		<<native>>				
			strictfp		<<strictfp>>				
			synchronized		<<synchronized>>				
		throws clause			raised exceptions				
		java docs			Comment(->Documentation)				
		type			direction	return	Parameter		
		Parameter	name		name				
			modifier	final	<<final>>				
			...		varArgList				
			type		type				
			type dimensions		multiplicity				
		Type Parameter	name		name		Template Parameter		
			bound		constraining classifier				
	Constructor	name			name			Operation <<constructor>>	
		modifiers	public		visibility	public			
			protected			protected			
			private			private			
		throws clause			raised exceptions				
		java docs			Comment(->Documentation)				
		Parameter	name		name		Parameter		
			modifier	final	<<final>>				
			...		varArgList				
			type		type				
			type dimensions		multiplicity				
		Type Parameter	name		name		Template Parameter		
			bound		constraining classifier				

Java					UModel				
	Type Parameter	name			name			Template Parameter	
		bound			constraining classifier				
Interface	name				name				Interface
	modifiers	package			visibility	package			
		public				public			
		protected				protected			
		private				private			
		abstract			abstract				
		strictfp			<<strictfp>>				
	filename				code file name				
	associated projectfile/directory				ComponentRealization				
	extends clause				Generalization(s)				
	java docs				Comment(->Documentation)				
	Field	name			name			Property	
		modifiers	public		visibility	public			
			static		static				
			final		<<final>>				
		type			type				
		type dimensions			multiplicity				
		default value			default				
		java docs			Comment(->Documentation)				
	Method	name			name			Operation	
		modifiers	public		visibility	public			
			abstract		abstract				
		throws clause			raised exceptions				
		java docs			Comment(->Documentation)				
		type			direction	return	Parameter		
		Parameter	name		name				
			modifier	final	<<final>>				
			...		varArgList				

Java				UModel			
			type	type			
			type dimensions	multiplicity			
		Type Parameter	name	name	Template Parameter		
			bound	constraining classifier			
	Type Parameter	name		name			Template Parameter
		bound		constraining classifier			
Enum	name			name			Enumeration
	modifiers	package		visibility	package		
		public			public		
		protected			protected		
		private			private		
	filename			code file name			
	associated projectfile/directory			ComponentRealization			
	java docs			Comment(->Documentation)			
	Enum Constant	name		name		Enumeration Literal	
	Field	name		name		Property	
		modifiers	package	visibility	package		
			public		public		
			protected		protected		
			private		private		
			static	static			
			transient	<<transient>>			
			volatile	<<volatile>>			
			final	<<final>>			
		type		type			
		type dimensions		multiplicity			
		default value		default			
		java docs		Comment(->Documentation)			
	Method	name		name		Operation	

Java					UModel				
		modifiers	package		visibility	package			
			public			public			
			protected			protected			
			private			private			
			static		static				
			abstract		abstract				
			final		<<final>>				
			native		<<native>>				
			strictfp		<<strictfp>>				
			synchronized		<<synchronized>>				
		throws clause			raised exceptions				
		java docs			Comment(->Documentation)				
		type			direction	return	Parameter		
		Parameter	name		name				
			modifier	final	<<final>>				
			...		varArgList				
			type		type				
			type dimensions		multiplicity				
		Type Parameter	name		name		Template Parameter		
			bound		constraining classifier				
	Constructor	name			name			Operation <<constructor>>	
		modifiers	public		visibility	public			
			protected			protected			
			private			private			
		throws clause			raised exceptions				
		java docs			Comment(->Documentation)				
		Parameter	name		name		Parameter		
			modifier	final	<<final>>				
			...		varArgList				
			type		type				

Java				UModel			
			type dimensions	multiplicity			
		Type Parameter	name	name	Template Parameter		
			bound	constraining classifier			
Parameterized Type				Anonymous Bound Element			
Annotation				<<annotations> modifiers			

6.6.4 Correspondencias con XML Schema

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de XML Schema cuando se genera código a partir del modelo.
- Elementos de XML Schema y elementos de UModel cuando se actualiza el modelo con el código.

Leyenda:

Elemento XSD/UML

Propiedad de estereotipo (=valor etiquetado)

XSD			UModel	
file path			projectfile	Component
schema	target namespace		name	Package <<namespace>>
	attributeFormDefault		attributeFormDefault	Class <<schema>>
	blockDefault		blockDefault	
	elementFormDefault		elementFormDefault	
	finalDefault		finalDefault	
	version		version	
	xml:lang		xml:lang	
	xmlns		xmlns	
	annotation	source	source	
		appinfo		Comment <<appinfo

XSD					UModel			
								>>
		documentation	xml:lang		xml:lang			Comment <<documentation>>
	attributeGroup	name			name			Class <<attributeGroup>>
		annotation	appinfo				Comment <<appinfo>>	
			documentation				Comment <<documentation>>	
		attribute	name		name		Property <<attribute>>	
			form		form			
			use		use			
			ref		type			
			type					
			default		default			
			fixed			fixed		
		attributeGroup	ref		type		Property <<attributeGroup>>	
		anyAttribute	namespace		namespace		Property <<anyAttribute>>	
			processContents		processContents			
	attribute	name			name			Class <<attribute>>
		form			form			
		use			use			
		type			type		Property	
		default			default			
		fixed				fixed		
		annotation	appinfo				Comment <<appinfo>>	
			documentation				Comment <<documentation>>	

XSD					UModel			
		simpleType			name (= name of Class + "_anonymousType[n]")	DataType <<simpleType>>		
	element	name			name			Class <<element>>
		abstract			abstract			
		block			block			
		final			final			
		form			form			
		nillable			nillable			
		type			type			Property
		default			default			
		fixed				fixed		
		substitutionGroup			general			Generalization <<substitution>>
		annotation	appinfo					Comment <<appinfo>>
			documentation					Comment <<documentation>>
		simpleType			name (= name of Class + "_anonymousType[n]")	DataType <<simpleType>>		
		complexType			name (= name of Class + "_anonymousType[n]")	Class <<complexType>>		
	group	name			name			Class <<group>>
		annotation	appinfo					Comment <<appinfo>>
			documentation					Comment <<documentation>>
		all			name (= "_all")			Property
					name (= "mg"_ + "all")			Class

XSD					UModel				
			annotation	appinfo		Comment <<appinfo>>			
				documentation		Comment <<documentation>>			
			element	name	name	Property <<element>>			
				ref	type				
				type					
		choice			name (= "_choice")		Property		
					name (= "mg"_ + "choice")		Class <<choice>>		
			annotation	appinfo		Comment <<appinfo>>			
				documentation		Comment <<documentation>>			
			element	name	name	Property <<element>>			
				ref	type				
				type					
			group			Property <<group>>			
			any	namespace	namespace	Property <<any>>			
				processContents	processContents				
			choice			Property			
						Class <<choice>>			
			sequence			Property			
						Class <<sequence>>			
		sequence			name (= "_sequence")		Property		

XSD					UModel			
					name (= "mg"_ + "sequence")		Class <<sequence>>	
			annotation	appinfo		Comment <<appinfo>>		
				documentation		Comment <<documentation>>		
			element	name	name	Property <<element>>		
				ref	type			
				type				
			group			Property <<group>>		
			any	namespace	namespace	Property <<any>>		
				processContents	processContents			
			choice			Property		
						Class <<choice>>		
			sequence			Property		
						Class <<sequence>>		
	notation	name			name			DataType <<notation>>
		system			system			
		public			public			
		annotation	appinfo				Comment <<appinfo>>	
			documentation				Comment <<documentation>>	
	complexType	name			name			Class <<complexType>>
		abstract			abstract			
		block			block			

XSD					UModel			
		final			final			
		mixed			mixed			
		annotation	source		source			
			appinfo			Comment <<appinfo>>		
			documentation	xml:lang	xml:lang	Comment <<documentation>>		
		group			name (= "_ref[n]")	Property <<group>>		
			maxOccurs		multiplicity			
			minOccurs					
			ref		type			
		all			name (= "mg"_ + "all")	Class <<all>>		
					name (= "_all")	Property		
			maxOccurs		multiplicity			
			minOccurs					
		choice			name (= "mg"_ + "choice[n]")	Class <<choice>>		
					name (= "_choice[n]")	Property		
			maxOccurs		multiplicity			
			minOccurs					
		sequence			name (= "mg"_ + "sequence[n]")	Class <<sequence>>		
					name (= "_sequence[n]")	Property		
			maxOccurs		multiplicity			
			minOccurs					
		attribute	name		name	Property <<attribute>>		
			ref		type			

XSD					UModel			
			type					
		attributeGroup	ref		type	Property <<attributeGroup>>		
		anyAttribute	namespace		namespace	Property <<anyAttribute>>		
			processContents		processContents			
		complexContent	restriction	base	general	Generalization <<restriction>>		
			extension			Generalization <<extension>>		
	simpleType	name			name		DataType <<simpleType>> Enumeration <<simpleType>>	
		final			final			
		annotation	source		source			
			appinfo				Comment <<appinfo>>	
			documentation	xml:lang	xml:lang		Comment <<documentation>>	
		list	itemType		name (= "_itemType")	Property <<itemType>>	<<list>>	
			simpleType		DataType <<simpleType>>			
		union	memberTypes		name (= "memberType[n]")	Property <<memberType>>	<<union>>	
			simpleType		DataType <<simpleType>>			
		minExclusive	value		value		<<minExclusive>>	
			fixed		fixed			
		minInclusive	value		value		<<minInclusive>>	
			fixed		fixed			
		maxExclu	value		value		<<maxExc	

XSD					UModel			
			fixed		fixed			
		maxInclusive	value		value	<<maxInclusive>>		
			fixed		fixed			
		totalDigits	value		value	<<totalDigits>>		
			fixed		fixed			
		fractionDigits	value		value	<<fractionDigits>>		
			fixed		fixed			
		length	value		value	<<length>>		
			fixed		fixed			
		minLength	value		value	<<minLength>>		
			fixed		fixed			
		maxLength	value		value	<<maxLength>>		
			fixed		fixed			
		whitespace	value		value	<<whitespace>>		
			fixed		fixed			
		pattern	value		value	<<whitespace>>		
		enumeration	value		name	EnumerationLiteral		
		simpleType				DataType <<simpleType>>		
		restriction	base		general	Generalization <<restriction>>		
	complexType simpleContent	name			name		DataType <<complexType>> <<simpleContent>>	
		annotation	source		source			
			appinfo			Comment <<appinfo>>		
			documentation	xml:lang	xml:lang	Comment <<documentation>>		

XSD					UModel				
		minExclusive	value		value		<<minExclusive>>		
			fixed		fixed				
		minInclusive	value		value		<<minInclusive>>		
			fixed		fixed				
		maxExclusive	value		value		<<maxExclusive>>		
			fixed		fixed				
		maxInclusive	value		value		<<maxInclusive>>		
			fixed		fixed				
		totalDigits	value		value		<<totalDigits>>		
			fixed		fixed				
		fractionDigits	value		value		<<fractionDigits>>		
			fixed		fixed				
		length	value		value		<<length>>		
			fixed		fixed				
		minLength	value		value		<<minLength>>		
			fixed		fixed				
		maxLength	value		value		<<maxLength>>		
			fixed		fixed				
		whitespace	value		value		<<whitespace>>		
			fixed		fixed				
		pattern	value		value		<<whitespace>>		
		attribute	name		name		Property <<attribute>>		
			ref		type				
			type						
		attributeGroup	ref		type		Property <<attributeGroup>>		
		anyAttribute	namespace		namespace		Property <<anyAttribute>>		

XSD					UModel				
			processContents		processContents				
		simpleType					DataType <<simpleType>>		
		restriction	base		general		Generalization <<restriction>>		
		extension	base		general		Generalization <<extension>>		
	import	schemaLocation			schemaLocation			ElementImport <<import>>	
		namespace			namespace				
	include	schemaLocation			schemaLocation			ElementImport <<include>>	
	redefine	schemaLocation			schemaLocation			ElementImport <<redefine>>	
		simpleType			<<redefine>>			DataType <<simpleType>>	
		complexType						Class <<complexType>>	
		attributeGroup						Class <<attributeGroup>>	
		group						Class <<group>>	

6.6.5 Correspondencias con elementos de BD

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de BD cuando se genera código a partir del modelo.
- Elementos de BD y elementos de UModel cuando se actualiza el modelo con el código.

<table>
<tr><th colspan="11">Database</th><th colspan="5">UModel</th></tr>
<tr><td>Database</td><td colspan="10">connection</td><td colspan="4">connection</td><td>Component</td></tr>
<tr><td rowspan="25">Database</td><td colspan="10">name</td><td colspan="4">name</td><td rowspan="25">Package <<name space>> <<Database>></td></tr>
<tr><td rowspan="24">Schema</td><td colspan="9">name</td><td colspan="4">name</td><td rowspan="24">Package <<name space>> <<Schema>></td></tr>
<tr><td rowspan="23">Table</td><td colspan="8">name</td><td colspan="3">name</td><td rowspan="23">Class <<Table>></td></tr>
<tr><td rowspan="11">Column</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="11">Property</td></tr>
<tr><td colspan="2">Data Type</td><td colspan="2">type</td></tr>
<tr><td colspan="2">Not Null</td><td colspan="2"><<not_null>></td></tr>
<tr><td colspan="2">Null</td><td colspan="2"><<nullable>></td></tr>
<tr><td colspan="2">Length</td><td colspan="2" rowspan="3">Multiplicity</td></tr>
<tr><td colspan="2">Precision</td></tr>
<tr><td colspan="2">Scale</td></tr>
<tr><td colspan="2">Default</td><td colspan="2">default</td></tr>
<tr><td colspan="2">Autoincrement</td><td colspan="2"><<autoincrement>></td></tr>
<tr><td colspan="2">Part of Primary Key</td><td colspan="2"><<PK>></td></tr>
<tr><td colspan="2">Part of Foreign Key</td><td colspan="2"><<FK>></td></tr>
<tr><td colspan="2">Part of Unique Key</td><td colspan="2"><<unique>></td></tr>
<tr><td rowspan="2">Primary Key</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="2">Class <<PrimaryKey>></td></tr>
<tr><td>Column</td><td>name</td><td>name</td><td>Property</td></tr>
<tr><td rowspan="3">Foreign Key</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="3">Class <<ForeignKey>></td></tr>
<tr><td>Column</td><td>name</td><td>name</td><td>Property</td></tr>
<tr><td rowspan="2">Foreign Column</td><td>name</td><td>name</td><td rowspan="2">Property</td></tr>
<tr><td>foreign table</td><td>type</td></tr>
<tr><td rowspan="2">Unique Key</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="2">Class <<UniqueKey>></td></tr>
<tr><td>Column</td><td>name</td><td>name</td><td>Property</td></tr>
<tr><td rowspan="3">Index</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="3">Class <<Index>></td></tr>
<tr><td rowspan="2">Column</td><td>name</td><td></td><td rowspan="2">Property</td></tr>
<tr><td>order: ascending</td><td><<ascending>></td></tr>
</table>

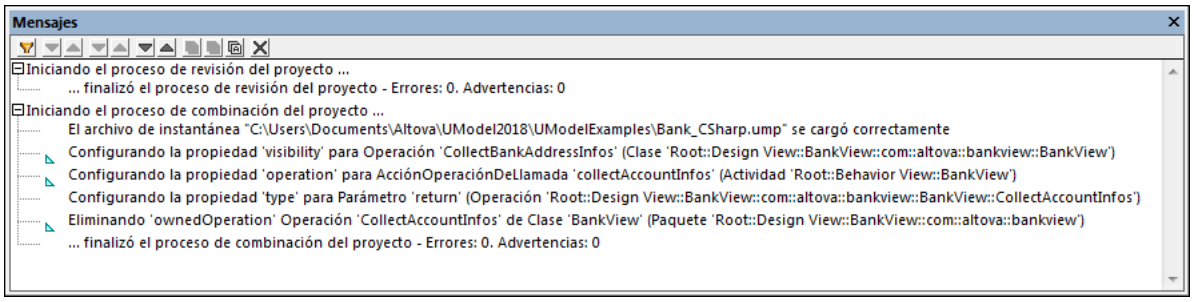
Database						UModel					
					order: ascending	<<descending>>					
			CheckConstraint	name		name		Class <<CheckConstraint>>			
				definition		definition					
		View	name			name			Class <<View>>		
			definition			definition					
			Column	name		name		Property			
				Data Type		type					
				Not Null		<<not_null>>					
				Null		<<nullable>>					
				Length		Multiplicity					
				Precision							
				Scale							
				Default		default					
				Autoincrement		<<autoincrement>>					
		Stored Procedure	name			name		Operation <<StoredProcedure>>	Class <<StoredProcedures>>		
			definition			definition					
			Parameter	name		name	Parameter				
				direction mode		direction					
				data type		type					
		Function	name			name		Operation <<Function>>	Class <<Functions>>		
			definition			definition					
			Parameter	name		name	Parameter				
				direction mode		direction					
				data type		type					
		Trigger	name			name			Class <<Trigger>>		
			definition			definition					

6.7 Combinar proyectos de UModel

En UModel puede realizar fusiones a 2 y 3 bandas para combinar varios proyectos de UModel distintos en un solo modelo *.ump. Esta función es de gran utilidad si en el mismo proyecto trabajan varias personas a la vez o si quiere reunir todo su trabajo en un solo modelo.

Para combinar dos proyectos UML:

1. Abra el archivo UML que debe hacer de archivo de destino (es decir, el archivo con el que se debe combinar el otro modelo).
2. Seleccione el comando **Proyecto | Combinar el proyecto**.
3. Seleccione el proyecto UML que debe combinarse con el primer proyecto. En la ventana *Mensajes* puede comprobar cómo se desarrolla el proceso de combinación.



Nota: si hace clic en una entrada de la ventana *Mensajes* el elemento de modelado correspondiente aparece resaltado en la *Estructura del modelo*.

Estas son las consecuencias de combinar dos proyectos:

- Elementos de modelado nuevos: los elementos que no existían en el proyecto principal se añaden al proyecto.
- Diferencias entre los mismos elementos de modelado: los elementos del segundo modelo tienen prioridad. Por ejemplo, solo puede haber un valor predeterminado para un atributo, así que se usa el valor predeterminado del segundo archivo.
- Diferencias entre diagramas: UModel comprueba si hay diferencias entre los diagramas de los dos modelos.

  - Si hay diferencias, los diagramas nuevos/distintos se añaden al modelo principal (con un sufijo numérico tipo actividad1, etc.) y el diagrama original se conserva.
  - Si no hay diferencias, no se realiza ningún cambio y se ignoran los diagramas que sean idénticos. Después puede eliminar los diagramas que quiera.

- Se puede deshacer el proceso de combinación paso a paso con el botón **Deshacer** de la barra de herramientas (o con **Ctrl+Z**).
- Al hacer clic en una entrada de la ventana *Mensajes* aparece resaltado el elemento de modelado correspondiente en la *Estructura del modelo*
- El nombre del archivo combinado es el del primer archivo que abrió.

6.7.1 Fusión de proyectos a tres bandas

UModel también ofrece una función para combinar varios proyectos de UModel editados por programadores diferentes. Esta función se conoce como fusión de proyectos a tres bandas.

La fusión de proyectos a tres bandas sirve para combinar proyectos de UModel de nivel superior, es decir, proyectos principales que pueden contener subproyectos. La fusión a tres bandas no funciona con archivos independientes si estos tienen referencias sin resolver a otros archivos.

Cuando se combinan proyectos principales, los subproyectos editables que contienen también se combinan automáticamente. Esto significa que no hace falta combinar los subproyectos por separado. Para ver un ejemplo consulte el apartado Ejemplo de fusión manual a tres bandas. Debe tener en cuenta algunos aspectos:

- el proceso de combinación se puede deshacer entero paso a paso con el botón **Deshacer** de la barra de herramientas o con **Ctrl+Z**.
- si hace clic en una entrada de la ventana *Mensajes*, el elemento de modelado correspondiente aparece resaltado en la *Estructura del modelo*.
- el proyecto principal combinado conserva el mismo **nombre de archivo** que el del archivo que eligió como destino de la combinación.

¿Qué efectos tiene la fusión de proyectos a tres bandas?

Es importante tener en cuenta que cuando decimos "proyecto original", nos referimos al archivo de proyecto que eligió como destino de la combinación y que abrió primero al principio del proceso.

Elementos de modelado nuevos:

- si el segundo proyecto tiene elementos que no existen en el proyecto original, estos elementos se añaden al proyecto de destino.
- si el proyecto original tiene elementos que no existen en el segundo proyecto, estos elementos se conservan en el proyecto de destino.

Elementos de modelado eliminados:

- si en el segundo proyecto se han eliminado elementos que todavía existen en el proyecto original, estos elementos se eliminan en el proyecto de destino.
- si en el proyecto original se han eliminado elementos que todavía existen en el segundo proyecto, estos elementos siguen estando eliminados en el proyecto de destino.

Diferencias entre los mismos elementos de modelado:

- si una propiedad (p. ej. el nivel de acceso de una clase) se modifica en el proyecto original o en el segundo proyecto, el modelo de destino incluye el valor más reciente.
- si una propiedad (p. ej. el nivel de acceso de una clase) se modifica tanto en el archivo original como en el segundo archivo, el modelo de destino toma el valor del segundo archivo (y aparece una advertencia en la ventana *Mensajes*).

Elementos con una posición distinta:

- si un elemento se mueve en el proyecto original o en el segundo proyecto, en el modelo de destino también cambia la posición de ese elemento.
- si un elemento se mueve (a un elemento primario distinto) tanto en el proyecto original como en el segundo proyecto, aparece un aviso para que seleccione manualmente el elemento primario en que se debe insertar el elemento en el modelo de destino.

Diferencias entre diagramas:

UModel primero comprueba si hay diferencias entre los diagramas de los dos modelos.

- si hay diferencias, el diagrama nuevo o distinto se añade al modelo de destino (con un sufijo numérico tipo actividad1, etc.) y el diagrama original se conserva.
- si no hay diferencias, no se realizan cambios y se ignoran los diagramas que sean idénticos. Después puede eliminar los diagramas que quiera.

Sistemas de control de versiones para fusiones a tres bandas

Si trabaja en un sistema en el que se reservan y liberan archivos (*check-in*/*check-out*), UModel genera automáticamente archivos de tipo "ancestro común" (también llamados copias instantáneas de volumen o instantáneas) que después se usan para la fusión a tres bandas. Con ellos se consigue una combinación mucho más precisa que con el método de fusión a dos bandas.

La fusión a tres bandas **automática** de UModel no funciona con todos los sistemas de control de versiones, pero en esos casos siempre puede usar la fusión a tres bandas **manual**.

- los sistemas de control de versiones que combinan archivos automáticamente sin la intervención del usuario no suelen ser compatibles con la fusión a tres bandas automática de UModel.
- los sistemas de control de versiones que piden que el usuario elija entre **Reemplazar** o **Combinar** cuando cambia un archivo del proyecto suelen ser compatibles con la fusión a 3 bandas automática de UModel. Después de que el control de versiones reemplace el archivo, seleccione el comando **Reemplazar** para activar el aviso de UModel que permite realizar una fusión a 3 bandas. Para reservar/liberar archivos es necesario usar UModel.
- puede poner bajo control de versiones tanto el proyecto principal como sus subproyectos. Si cambia datos en un subproyecto un aviso automático le pedirá que libere el subproyecto.
- cada acción de reservar/liberar crea un *archivo antecesor común* o instantánea que se utiliza después durante el proceso de fusión a 3 bandas.

Nota: los archivos de instantánea solo se crean y emplean de forma automática con la versión independiente de UModel (es decir, esta función no está disponible en el complemento de UModel para Eclipse o Visual Studio).

Ejemplo

Imagine que Usuario A edita un archivo de proyecto de UModel y cambia el nombre de una clase en el diagrama BankView Main. Ahora imagine que Usuario B abre el mismo archivo de proyecto y cambia el nivel de acceso de esa misma clase.

Como UModel genera una instantánea para cada usuario, el historial de edición de instantáneas permite combinar los diferentes cambios ocurridos en el proyecto. En este caso, en el proyecto de destino la clase tendrá el nuevo nombre como el nuevo nivel de acceso.

6.7.2 Ejemplo de fusión manual a tres bandas

Este ejemplo muestra una fusión simple de proyecto a tres bandas. Imaginemos que dos usuarios distintos (Daniel y Alicia) crearon cada uno una copia de un proyecto de UModel y realizaron modificaciones en ellas. Ahora existen tres versiones distintas del mismo proyecto: la original, la copia de Daniel y la copia de Alicia. En el contexto de una fusión a tres bandas, el proyecto original representa el *archivo antecesor común*.

Ahora imaginemos que el archivo antecesor común es el proyecto **Bank_CSharp.ump**, que encontrará en la carpeta **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**. Las copias de Daniel y Alicia deben crearse de forma manual, por lo que primero crearemos dos copias del proyecto **Bank_Csharp.ump** en carpetas secundarias dentro de la carpeta **...\UModelExamples**. Estas carpetas secundarias recibirán el nombre de **C#_Alicia** y **C#_Daniel**. No es necesario modificar el nombre del archivo de proyecto.

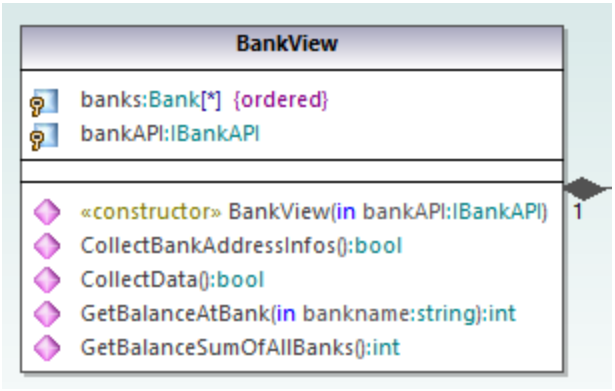
> Use el comando **Proyecto | Guardar como** para crear las copias de Daniel y Alicia. Cuando la aplicación pregunte si desea ajustar las rutas de acceso relativas, haga clic en **Sí**. Así evitará errores de sintaxis en las copias del proyecto.

El objetivo de este ejemplo es que Alicia consiga un proyecto final que combine no solo los cambios del archivo **Bank_CSharp.ump** original, sino también los de la copia de Daniel, mediante una fusión a tres bandas.

Paso nº1: preparar el proyecto de Daniel

Daniel abre el archivo de proyecto **Bank_CSharp.ump** de la carpeta **C#_Daniel**, abre el diagrama "BankView Main" y realiza cambios en la clase `BankView`.

1. Daniel elimina la operación `CollectAccountInfos():bool` de la clase `BankView`.
2. Daniel cambia el nivel de acceso de la operación `CollectBankAddressInfos():bool` de "protected" a "public".

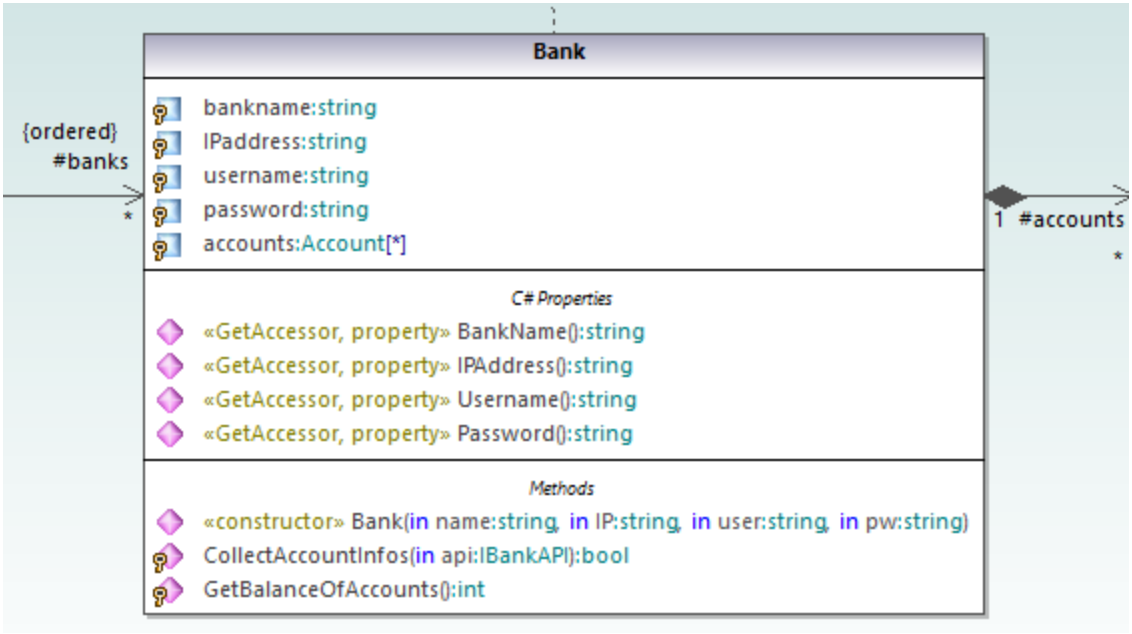


3. Después guarda el proyecto.

Paso nº2: preparar el proyecto de Alicia

Alicia abre el archivo de proyecto **Bank_CSharp.ump** de la carpeta **C#_Alicia**, abre el diagrama "BankView Main" y realiza cambios en la clase `Bank`.

1. Alicia cambia el nivel de acceso de las operaciones `CollectAccountInfos` y `GetBalanceOfAccounts` de "public" a "protected".



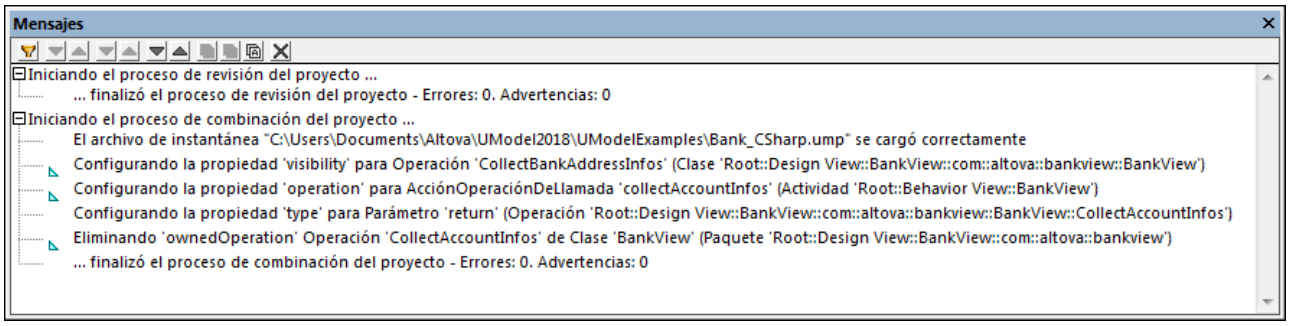
2. Después guarda el proyecto.

Paso nº3: realizar la fusión a tres bandas

Alicia comienza la fusión a tres bandas:

1. Alicia abre su proyecto, que está en la carpeta **C#_Alicia**.
2. En el menú **Proyecto** hace clic en el comando **Combinar el proyecto (fusión a tres bandas)** y selecciona el archivo de proyecto de Daniel, que está en la carpeta **C#_Daniel**.
3. La aplicación le solicita que abra el archivo antecesor común. Alicia selecciona el archivo de proyecto original **Bank_CSharp.ump** de la carpeta **...\UModelExamples**.

El proceso de fusión a tres bandas se inicia y la aplicación vuelve al archivo de proyecto desde el cual se inició la fusión a tres bandas (es decir, al archivo de proyecto de la carpeta **C#_Alicia**). En la ventana *Mensajes* aparecen los detalles del proceso de combinación.



Como resultado de la fusión a tres bandas:

- los cambios que realizó Daniel en el proyecto se reproducen en el proyecto de Alicia.
- los cambios que realizó Alicia en el proyecto se conservan en el archivo de proyecto.

Nota: a partir de ahora se debe usar el archivo de proyecto de la carpeta C#_Alicia como archivo antecesor común para futuras fusiones a tres bandas de los archivos de proyecto de las carpetas C#_Daniel y C#_Alicia.

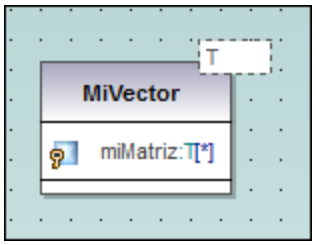
6.8 Plantillas UML

En UModel puede usar plantillas UML y crear asignaciones entre ellas y genéricos Java, C# y Visual Basic.

- Las plantillas son elementos de modelado potenciales con parámetros formales sin enlazar.
- Estos elementos de modelado parametrizados describen un grupo de elementos de modelado de un tipo concreto: clasificadores u operaciones.
- Las plantillas no se pueden usar como tipos directamente, sus parámetros deben estar enlazados.
- *Generar instancias* significa enlazar los parámetros de la plantilla con valores reales.
- Los valores reales de los parámetros son expresiones.
- El enlace que existe entre una plantilla y un elemento de modelado produce un elemento de modelado nuevo (elemento enlazado) basado en la plantilla.
- Si en C# existen varios clasificadores de restricción, los parámetros de la plantilla se pueden editar directamente en la ventana *Propiedades* cuando se selecciona el parámetro de la plantilla.

Presentación de **firmas** de plantilla en UModel:

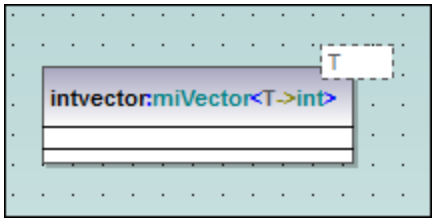
- En la imagen que aparece a continuación puede verse la plantilla de clase **`MiVector`**, cuyo parámetro de plantilla formal (T) aparece en la esquina superior derecha en un rectángulo discontinuo.
- Los parámetros formales sin información de tipo (T) son clasificadores implícitos: clase, tipo de datos, enumeración, tipo primitivo e interfaz. Los demás tipos de parámetro deben mostrarse explícitamente (p. ej. los enteros).
- La propiedad **`miMatriz`** tiene un número ilimitado de elementos de tipo T.



Cuando se hace clic con el botón derecho en la plantilla, aparece un menú contextual con el comando **Mostrar | Elementos enlazados**. Este comando muestra los elementos enlazados propiamente dichos.

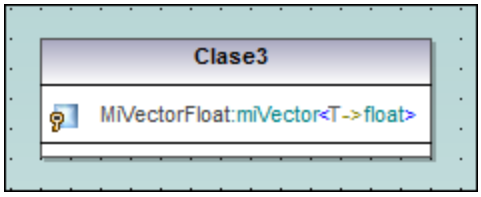
Presentación de **enlaces** de plantilla en UModel:

- En la imagen que aparece a continuación puede verse la plantilla con nombre enlazada **`intvector`**.
- Se trata de una plantilla de tipo **`miVector`**.
- El parámetro T se sustituye con int.
- Los caracteres -> significan *sustituido con*.



Uso de plantillas en operaciones/propiedades:

- En la imagen anterior puede ver un enlace de plantilla anónimo.
- Tiene la propiedad **`MiVectorFloat`** de tipo **`MiVector<T->float>`**.



También puede definir plantillas cuando defina propiedades u operaciones. La función de finalización automática le ayudará a utilizar la sintaxis correcta.

- La operación Operación1 devuelve un vector de tipo float.

6.8.1 Firmas de plantilla

Una *firma de plantilla* es una cadena de texto que especifica los parámetros de plantilla formales. Por su parte, una *plantilla* es un elemento parametrizado que se utiliza para generar elementos de modelado nuevos mediante la sustitución o el enlace de parámetros formales con parámetros reales (valores).

Parámetro de plantilla formal
T
Plantilla con un solo parámetro formal sin tipo
(almacena elementos de tipo T)

Varios parámetros de plantilla formales
KeyType:DateType, ValueType

Sustitución de parámetros
T>unaClaseBase
La sustitución de parámetros debe ser de tipo unaClaseBase o derivarse de ese tipo.

Valores predeterminados para parámetros de plantilla
T=unValorPredeterminado

Clasificadores de sustitución
T>{contract}unaClaseBase

allowsSubstitutable es true
El parámetro debe ser un clasificador que puede ser sustituido con el clasificador designado por el nombre de clasificador.

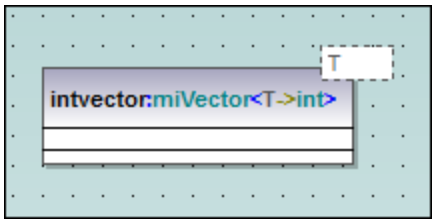
Parámetros de plantilla de restricción
T:Interface>unaInterfaz
Cuando la restricción limite a un elemento que no sea una clase (una interfaz, un tipo de datos), la restricción aparece después del carácter ":". Por ejemplo, T está restringido a una interfaz (T:Interfaz), que debe ser de tipo "unaInterfaz" (>unaInterfaz).

Usar comodines en firmas de plantilla
```
T>vector<T->?<unaClaseBase>
```
El parámetro de plantilla T debe ser de tipo "vector" que contiene objetos que son un supratipo de unaClaseBase.

Parámetros de plantilla de extensión
```
T>Comparable<T->T>
```

6.8.2 Enlace de plantilla

Un *enlace de plantilla* es el resultado de sustituir los parámetros formales con los valores reales (es decir, se crea una instancia de la plantilla). UModel genera automáticamente clases enlazadas anónimamente cuando se produce este enlace. Los enlaces se pueden definir en el campo del nombre de la clase, como puede ver en la imagen siguiente.



Parámetros formales de sustitución/enlace
```
vector <T->int>
```

Crear enlaces usando el nombre de la clase
```
a_float_vector:vector<T->float>
```

Enlazar varias plantillas simultáneamente
```
Clase5:vector<T->int, map<KeyType->int, ValueType<T->int>
```

Usar comodines ? como parámetros (Java 5.0)
```
vector<T->?>
```

Restringir comodines - límites superiores (extensión de UModel)
```
vector<T->?>unaClaseBase>
```

Restringir comodines - límites inferiores (extensión de UModel)

```
vector<T->?<unaClaseDerivada>
```

6.8.3 Usar plantillas en operaciones y propiedades

Operación que devuelve una plantilla enlazada

```
Clase1
Operación1():vector<T->int>
```

El parámetro `T` está enlazado con `int`. La `Operación1` devuelve un vector de tipos `int`.

Clase que contiene una operación de plantilla

```
Clase1
Operación1<T>(in T):T
```

Usar comodines

```
Clase1
Propiedad1:vector<T->?>
```

Esta clase contiene un vector genérico cuyo tipo no se ha especificado (? es el comodín).

Para ver las propiedades con tipo como asociaciones:

- haga clic con el botón derecho en una propiedad y seleccione **Mostrar | PropiedadX como asociación** o
- arrastre una propiedad hasta el diagrama.

7 Transformar modelos UML

Puede transformar cualquier paquete de UML de un lenguaje de modelado en otro. Esta transformación convierte todos los elementos relevantes del lenguaje de origen al lenguaje de llegada: clases, interfaces, atributos, operaciones, generalizaciones, etc. Los lenguajes de origen y de destino pueden ser cualesquiera de entre los que son compatibles con UModel (C++, C#, Java, VB.NET, UML, además de bases de datos y esquemas XML).

Una transformación conlleva un modelo "fuente" (es decir, el paquete que quiere transformar) y un modelo "meta" (paquete de destino). Como puede que el paquete de destino ya contenga elementos, puede realizar la transformación de un modelo de dos maneras:

1. Sobrescribir los cambios del modelo de origen en el de destino
2. Combinar los cambios del modelo de origen en el de destino

(Si el destino es un paquete nuevo, entonces no importa si los cambios se sobrescriben o se combinan al transformar el modelo por primera vez.)

Si el modelo de origen contiene diagramas de clases, existe la opción de transformarlos en el modelo de destino (esto se aplica a C#, Java, VB.NET y UML). Los diagramas que existen en el modelo de destino se actualizan conforme a la configuración de la transformación. Es decir, los elementos de los diagramas del modelo de origen sobrescribirán o se combinarán con los del modelo de destino.

Durante la transformación un cuadro de diálogo asistente le permitirá escoger si quiere asignar cada uno de los tipos de datos del lenguaje de origen con un tipo de datos en el lenguaje de destino. Si omite este paso, UModel usará asignaciones predeterminadas. Las asignaciones de tipos de datos también pueden cambiarse más tarde, pero en ese caso deberá volver a efectuar la transformación para que se reflejen los cambios en el modelo de destino.

Al realizar transformaciones de modelos, UModel hará los siguientes cambios automáticamente:

- si en el modelo de origen se ha aplicado el estereotipo UML «create» a una operación de clase, en el modelo de destino (C++, C#, Java, VB.NET) el estereotipo aplicado será «constructor» y viceversa: si una operación tiene es estereotipo «constructor» en C++, C#, Java o VB.NET, la misma operación tendrá el estereotipo «create» en el modelo UML de destino.
- si el modelo de destino es una BD, una propiedad "id" del modelo de origen se convertirá en una clave principal o foránea del tipo de datos correspondiente en el modelo de destino.

UModel permite actualizar continuamente los modelos transformados. Esto significa que puede trabajar en el modelo de origen y efectuar la transformación tantas veces como sea necesario para mantener el modelo de destino al día con el modelo de origen. La transformación de modelos también se puede configurar para que se ejecute de forma automática (véase Configuración de la transformación).

Para transformar un modelo:

1. Abra el proyecto de UModel que contiene el paquete que hará de modelo de origen.
2. En el menú Proyecto haga clic en **Transformación de modelos**.
3. Seleccione el paquete de origen (el que quiere transformar a un lenguaje distinto) y haga clic en **Siguiente**.

4. Seleccione un paquete de destino y haga clic en siguiente (para colocar todos los elementos en un paquete de destino nuevo, marque la casilla **Transformar en un paquete nuevo**).
5. Escoja el tipo de transformación (por ejemplo, Java a **C#**). El resto de ajustes se explican en Configuración de la transformación.
6. Ahora tiene dos opciones:
   a. Para llevar a cabo la transformación con las asignaciones predeterminadas haga clic en **Finalizar**.
   b. Para revisar las asignaciones antes de la transformación haga clic en Aceptar, cambie las asignaciones de datos como desee y haga clic en Finalizar.

Cuando una transformación se efectúa con éxito se genera automáticamente un nuevo diagrama de paquetes llamado "Transformación de modelos desde <paquete de origen> a <paquete de destino>". El diagrama se genera en el paquete de *destino*. La imagen siguiente ilustra el paquete de origen, el paquete de destino, la relación de dependencia entre ambos y una lista de valores etiquetados.

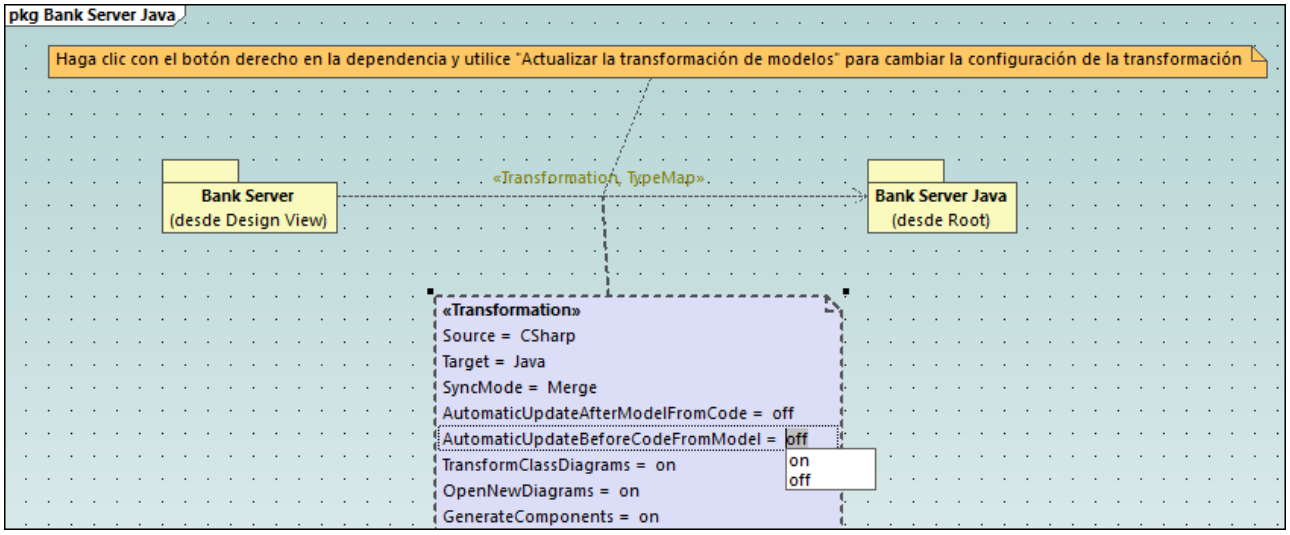


Diagrama de ejemplo "Transformación de modelo"

Aparte de ilustrar la transformación de un modelo, este diagrama también permite modificar la configuración de la transformación del modelo:

1. Haga clic en la relación de dependencia del diagrama (o en la ventana *Estructura del modelo* que encontrará en *Relaciones*)
2. Cambie las opciones necesarias desde la ventana Propiedades.

También puede hacer doble clic directamente en un valor etiquetado del diagrama para cambiar su valor.

Una vez haya terminado de cambiar la configuración, vuelva a efectuar la transformación para actualizar el modelo de destino. Para ello:

- Haga clic con el botón derecho en la relación de dependencia del diagrama y seleccione **Actualizar la transformación de modelos** del menú contextual o
- Haga clic con el botón derecho en el paquete de *origen* de la ventana *Estructura del modelo* y seleccione **Actualizar la transformación de modelos** del menú contextual.

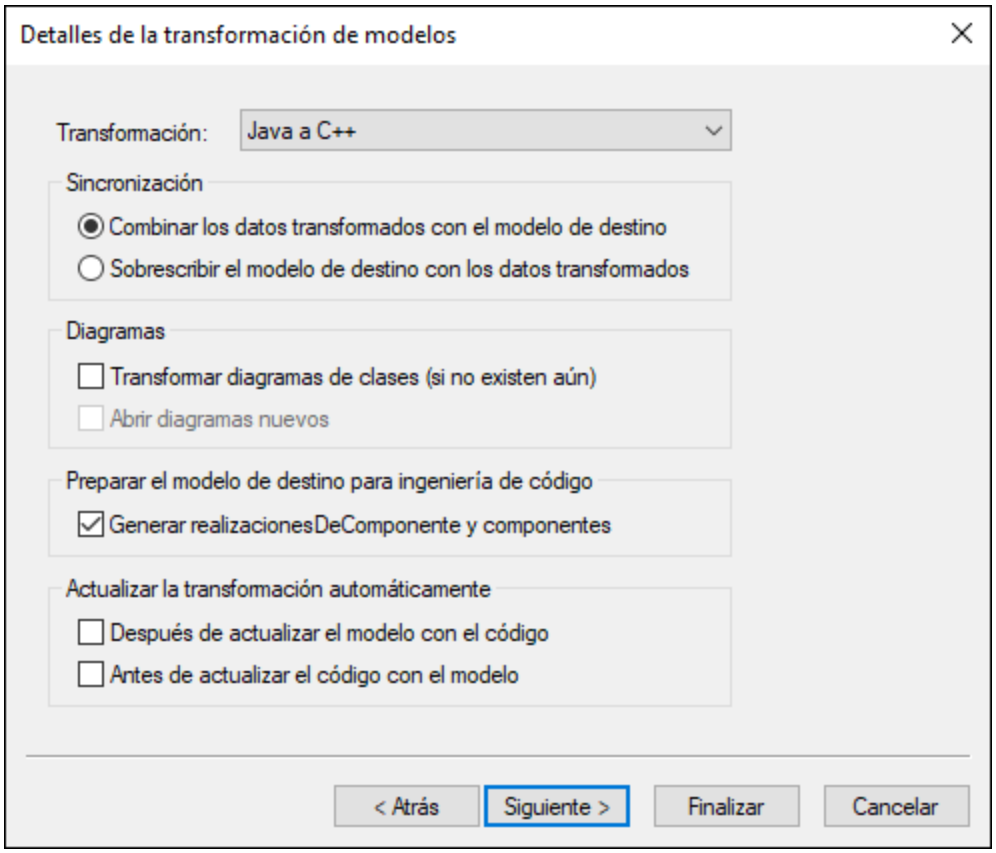
Aquí tiene algunos ejemplos de transformaciones paso a paso:

- Ejemplo: transformar un modelo Java en un modelo C++
- Transformar un modelo C# en un modelo Java
- Ejemplo: convertir la estructura de una BD Access en SQLite

7.1 Configuración de la transformación

La transformación de modelos puede configurarse en el cuadro de diálogo "Detalles de la transformación de modelos", que aparece al realizar una una nueva transformación de un modelo o al actualizar una que ya existe. Para más detalles consulte Transformar modelos UML.

Nota: al transformar un modelo en C# existe la opción de transformar campos en propiedades C# autoimplementadas. Esta opción está disponible como casilla de verificación en el cuadro de diálogo **Asignación de tipos**. Para acceder a este cuadro de diálogo haga clic en Siguiente en el cuadro de diálogo **Detalles de la transformación de modelos**.



A continuación exponemos las opciones disponibles.

Transformación

Este cuadro combinado permite seleccionar el lenguaje de origen y destino de la transformación. Las opciones del cuadro combinado dependen del lenguaje del paquete de origen elegido para la transformación. Esta opción está deshabilitada si vuelve a efectuar una transformación que ya existe.

Sincronización

Aquí tiene dos opciones con las que puede definir si los datos transformados se combinan con los datos de destino o si, por el contrario, los datos transformados sobrescriben los datos de destino. Por ejemplo, asumamos que una clase de los datos de origen contiene `OperationA`, mientras que en los datos de destino

la misma clase contiene `OperationA` y `OperationB`. Si combina los datos ambas operaciones seguirán existiendo en el modelo de destino. Si los sobrescribe, se borrará `OperationB` del modelo de destino.

Diagramas

La opción **Transformar diagramas de clases (si no existen aún)** genera diagramas de clases nuevos si estos no existen en el modelo de destino. Para abrir todos los diagramas una vez haya finalizado la transformación, marque la casilla **Abrir diagramas nuevos**.

Preparar el modelo de destino para ingeniería de código

Seleccione la opción Generar **realizacionesDeComponentes y componentes** si quiere habilitar la ingeniería inversa en el paquete de destino. Al seleccionar esta casilla, UModel crea automáticamente relaciones RealizaciónDeComponente y componentes de ingeniería de código en el modelo de destino. Antes de generar código a partir del modelo de destino, asegúrese de indicar también una carpeta para la generación de código:

1. En el paquete "Vista Componente" haga clic en el componente generado automáticamente por UModel.
2. Introduzca la ruta en la propiedad directorio de la ventana Propiedades.

Para obtener más información, consulte Generar código de programa.

Actualizar la transformación automáticamente

Esta opción permite mantener el modelo de origen sincronizado con el de destino y es útil en caso de que su modelo de origen esté configurado para generar código (o para ser actualizado con código). Si modifica frecuentemente el modelo de origen (o su código fuente) después de haberlo transformado en el modelo de destino puede propagar automáticamente todos los cambios al modelo de destino:

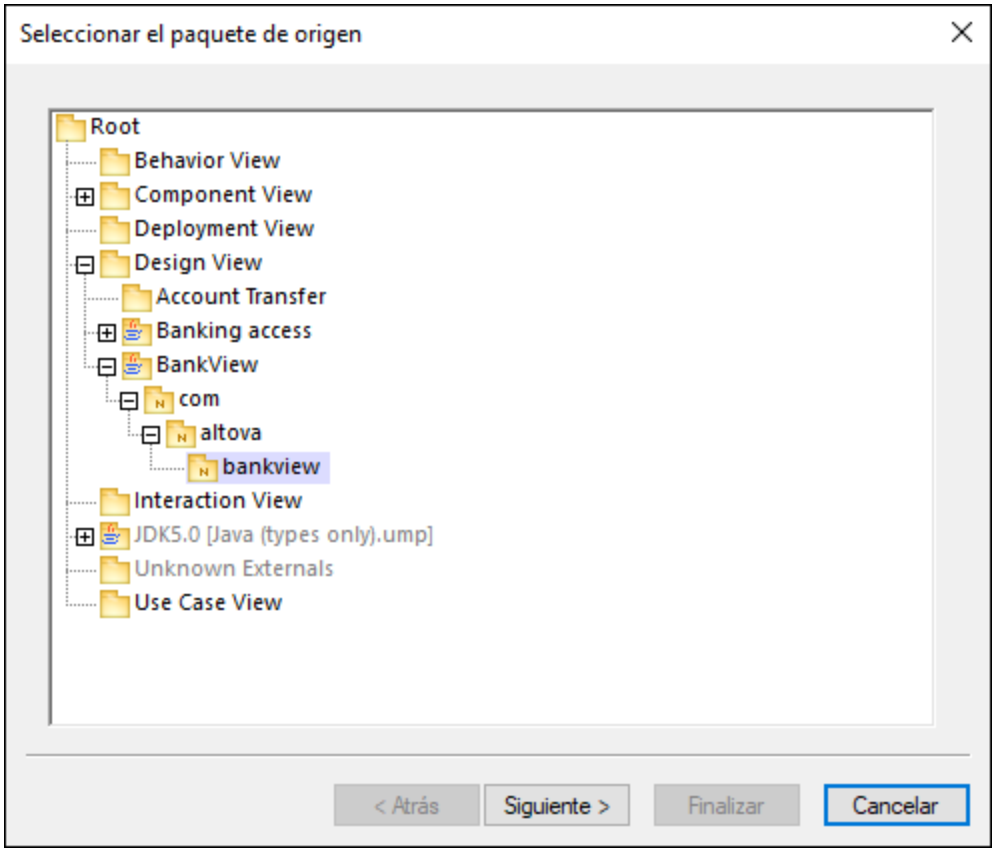
a) cada vez que actualice el modelo de origen desde el código de programa de origen
b) siempre antes de generar código de programa a partir del modelo de origen
c) en los dos casos anteriores.

Por ejemplo, asumamos que su proyecto contiene un paquete creado originalmente para ingeniería de código C#. Este paquete se transformó luego en un paquete Java usando el comando de menú **Proyecto | Transformación de modelos**. Después de la transformación, su proyecto tiene dos paquetes: el paquete C# de origen y el paquete Java de destino. Si habilita la opción a) la transformación de C# en Java se llevará a cabo automáticamente cada vez que modifique algo en el código C# y actualice el modelo con los cambios. Asimismo, si habilita la opción b) y ha modificado algo en el modelo en C#, la transformación de C# en Java se llevará a cabo automáticamente cada vez que genere el código de programa en C#. Para ver un ejemplo más detallado, consulte Transformar un modelo C# en un modelo Java.

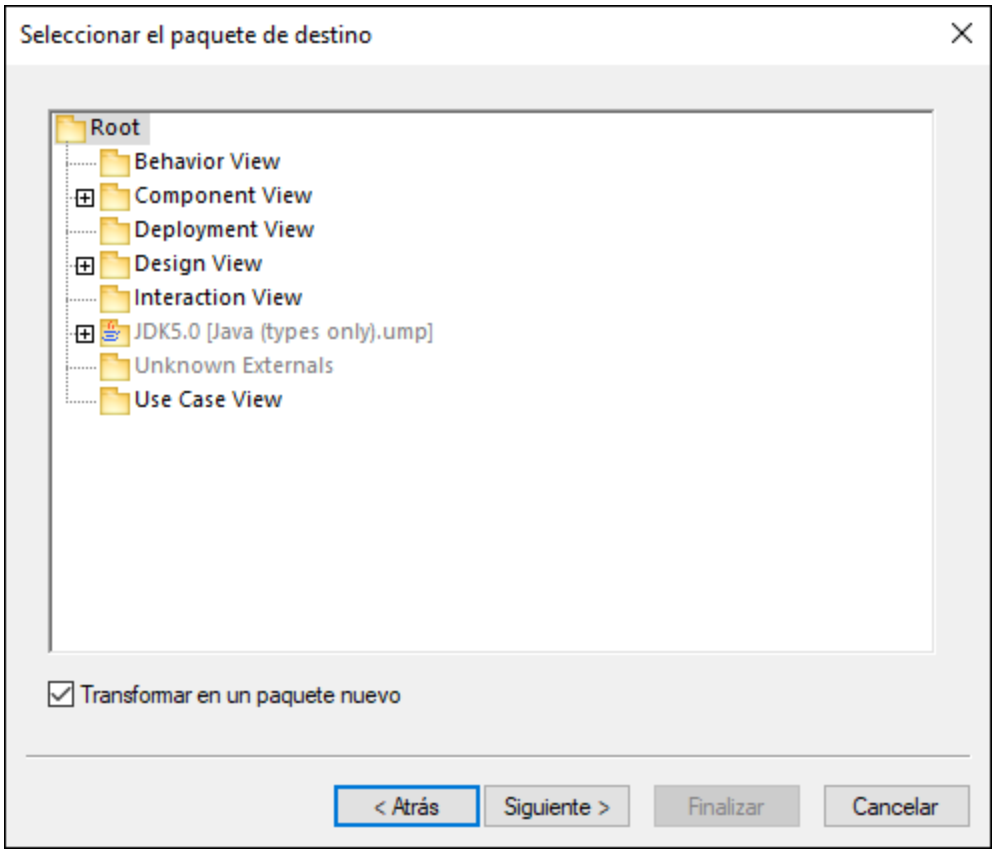
7.2 Ejemplo: transformar un modelo Java en un modelo C++

En este ejemplo veremos cómo transformar un modelo Java en un modelo C++. También aprenderemos a generar código C++ a partir del modelo generado (de destino).

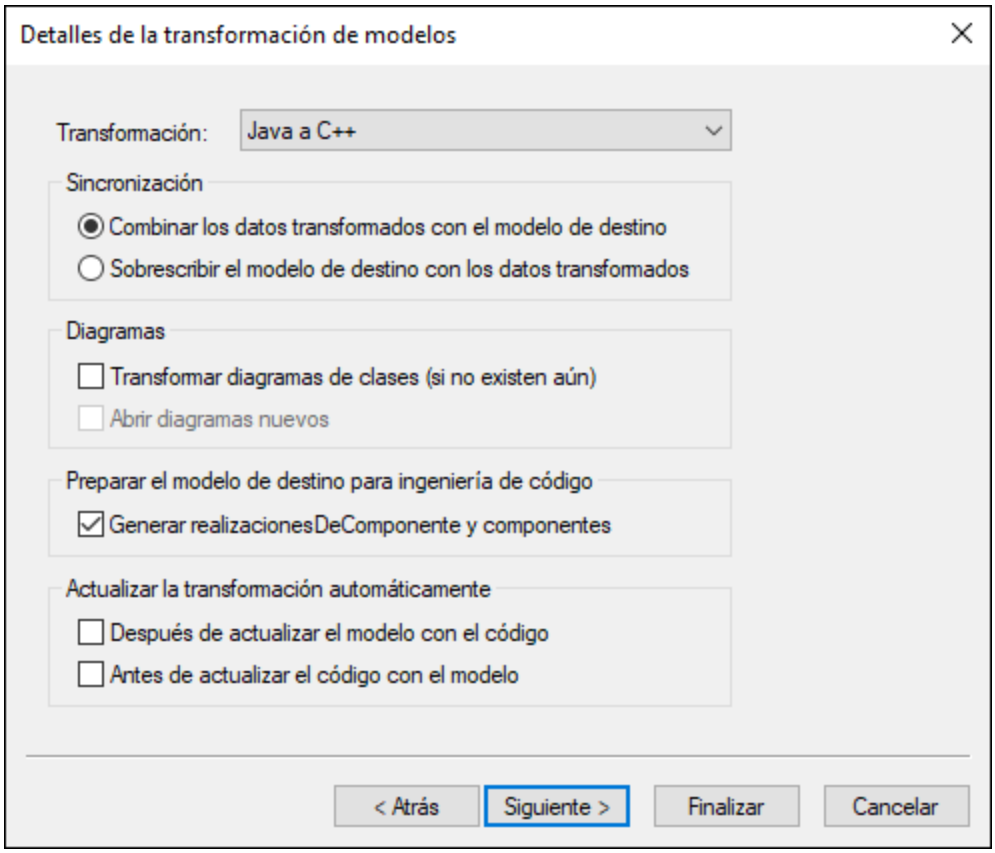
1. Abra el archivo de ejemplo `Bank_Java.ump`, que encontrará en la carpeta **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**.
2. En el menú Proyecto haga clic en **Transformación de modelos**.
3. Seleccione como paquete de origen el espacio de nombres "bankview". La ruta completa a este paquete en la Estructura del modelo es **Root \ DesignView \ BankView \ com \ altova \ bankview**.



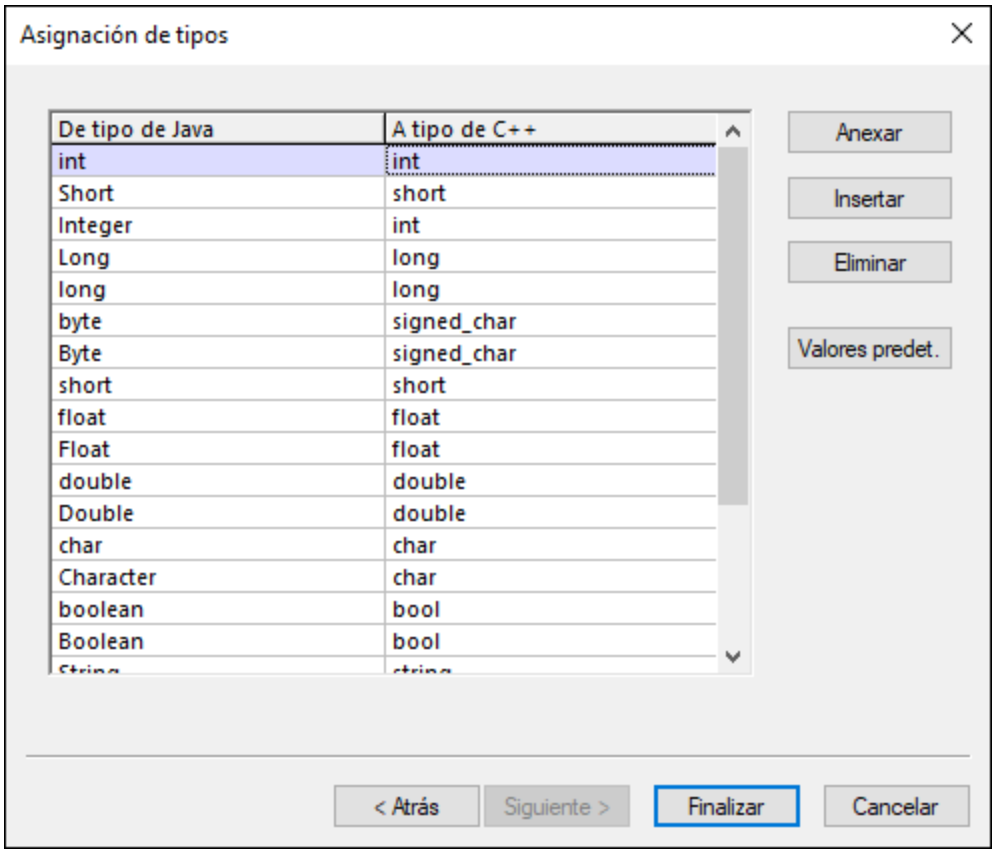
4. Haga clic en **Siguiente**. Seleccione la casilla **Transformar en un paquete nuevo**.



5. Haga clic en Siguiente. En el cuadro de diálogo que aparece, seleccione el tipo de transformación **Java a C++**. Para consultar el resto de opciones de configuración, consulte Configuración de la transformación.



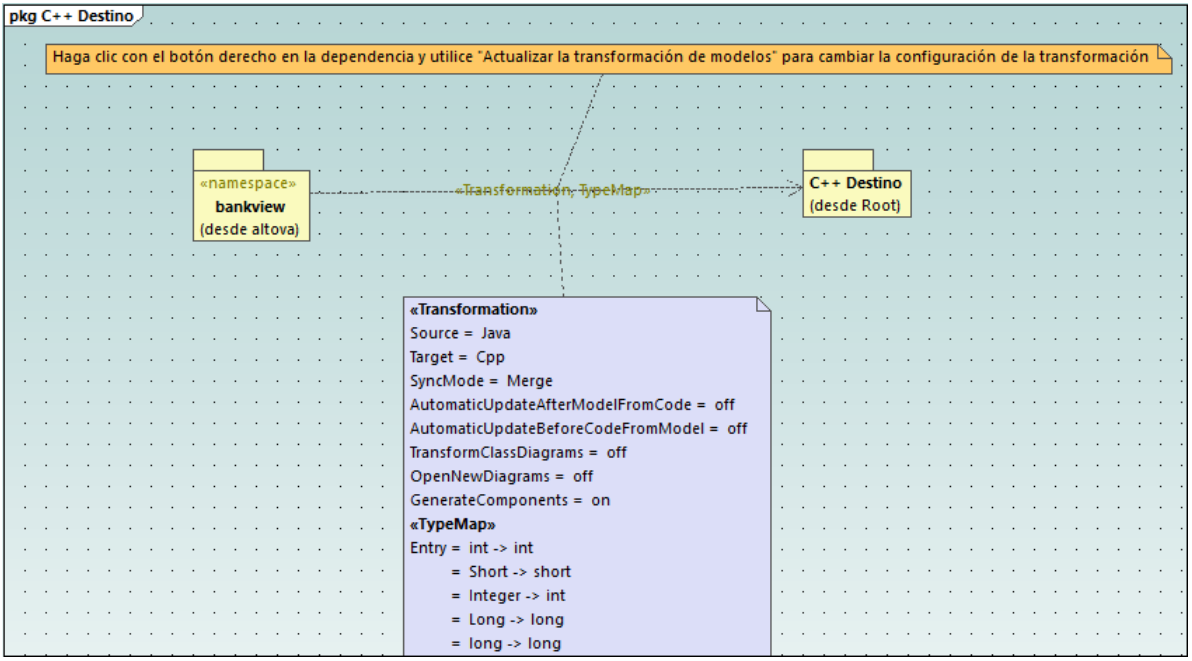
6. Haga clic en **Siguiente**. Se abrirá el cuadro de diálogo "Asignación de tipos", en el que puede definir el tipo de asignación entre Java y C#. Haga clic en Finalizar para usar la configuración predeterminada.



7. Cuando UModel le pregunte si quiere incluir el perfil *Model Transformation Profile de UModel* haga clic en **Aceptar**.

Cuando termine la transformación observará los siguientes cambios en el proyecto:

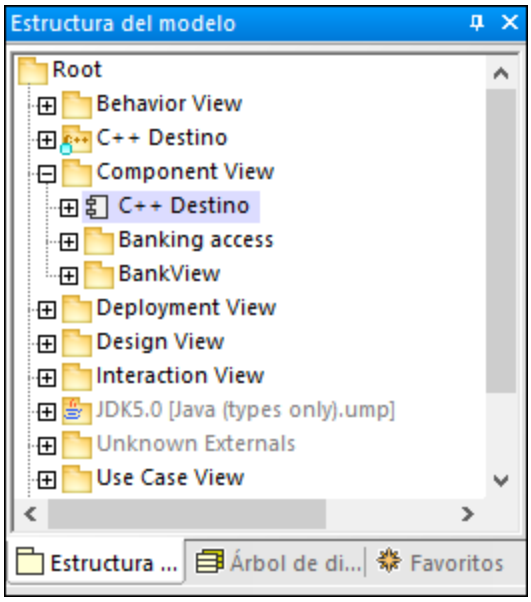
- Se genera un diagrama de paquetes llamado "Transformación de modelos desde bankview a C++ Destino" en el modelo de destino. Este paquete se abre automáticamente e ilustra la transformación que acaba de tener lugar. Si necesita cambiar las opciones previamente definidas, consulte el apartado Transformar modelos UML.



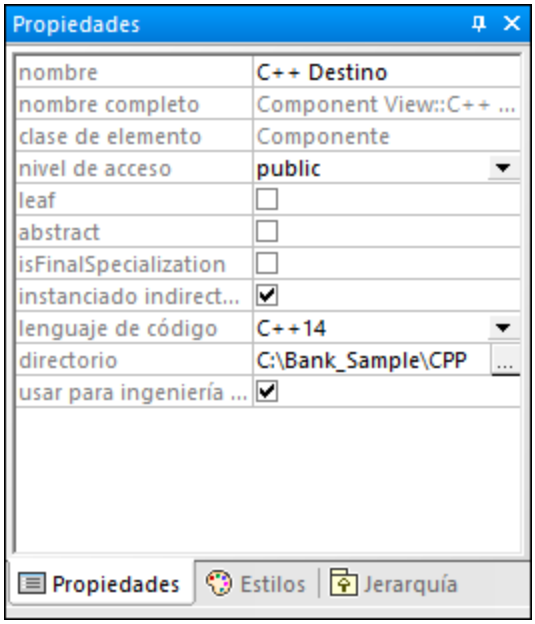
- La Estructura del modelo ahora incluya un paquete C++ Target. Este paquete contiene todos los elementos transformados desde el modelo de origen Java y adaptados a C++. Por ejemplo, si abre el diagrama "BankView Main" notará que contiene el tipo `bool`, cuando en Java el tipo es `boolean`.
- El paquete "Component View" de la ventana *Estructura del modelo* incluye un nuevo componente, "C++ Target", que se generó automáticamente porque la opción **realizacionesDeComponentes y componentes** estaba habilitada. El nuevo componente define las opciones de ingeniería de código para el modelo de destino (en este caso, C++).

Ahora puede generar código C++ desde el modelo de destino:

1. Haga clic en el componente "C++ Destino" del paquete "Component View".

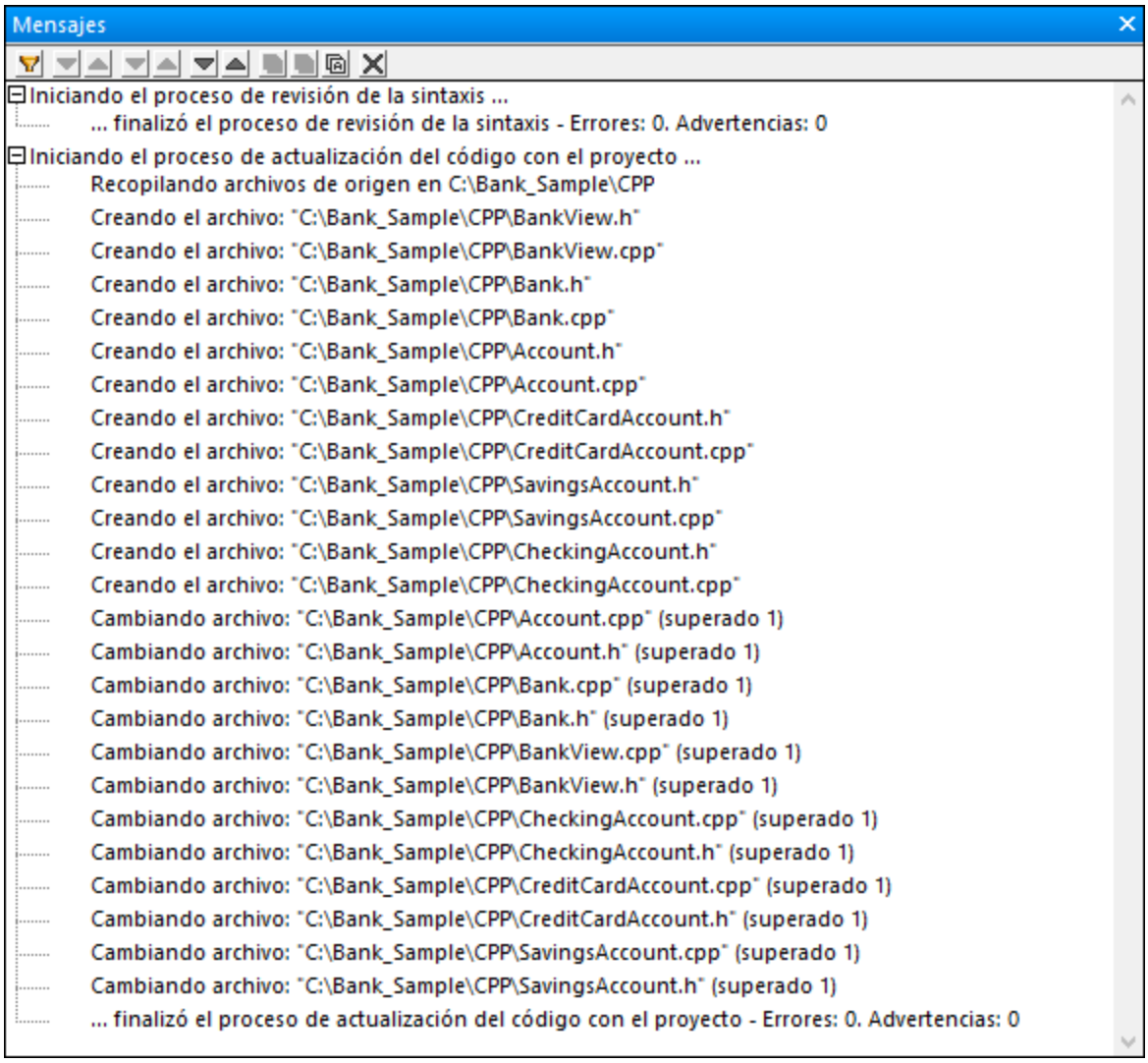


2. En la propiedad **directorio** de la ventana Propiedades, introduzca la carpeta en la que se debe generar el código C++ (por ejemplo, **C:\Bank_Sample\CPP**, siempre que esta carpeta exista).



3. Haga clic con el botón derecho en el paquete C++ Destino y seleccione **Ingeniería de código | Combinar código de programa con Paquete de Umodel**.

La ventana Mensajes muestra el resultado de la generación de código C++.



Para más información sobre generar código desde un proyecto de UModel, consulte Generar código de programa.

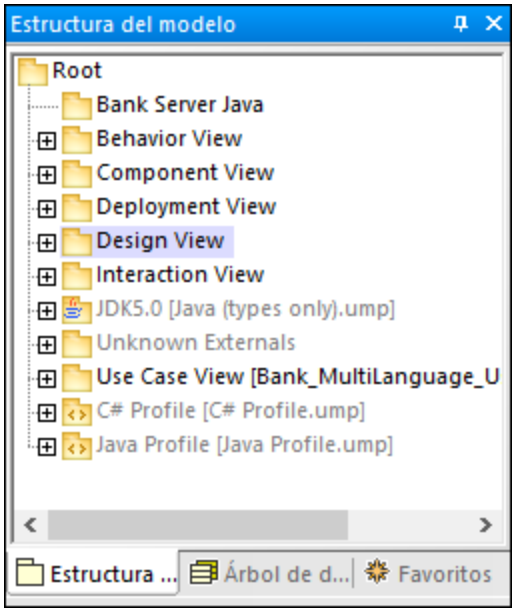
7.3 Transformar un modelo C# en un modelo Java

Este ejemplo muestra cómo transformar un modelo C# en un modelo Java. También ilustra cómo mantener sincronizados los modelos de origen y de destino, manual o automáticamente.

El proyecto de UModel que usaremos para este ejemplo se encuentra en **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Bank_multiLanguage.ump**. Si abre el paquete "Design View" verá que el modelo contiene dos paquetes escritos en Java y uno escrito en C#. El ejemplo asume que ahora los requisitos han cambiado y que el tercer paquete también debe implementarse en Java.

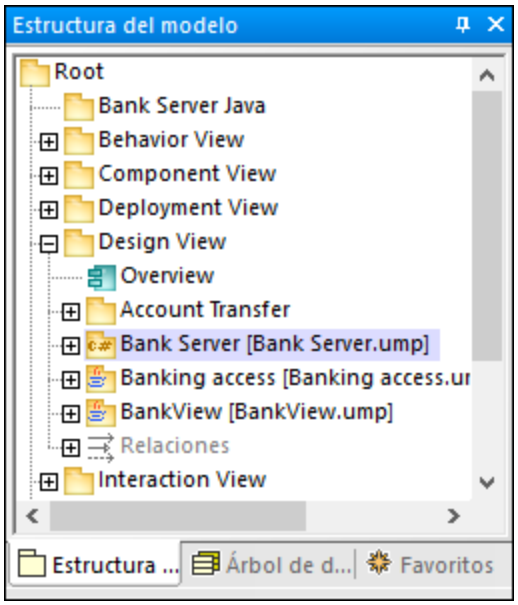
Empecemos por crear el paquete que almacenará todos los elementos del nuevo modelo de destino en Java.

1. Haga clic con el botón derecho en el paquete raíz `Root`, seleccione **Elemento nuevo | Paquete** del menú contextual.
2. Cambie el nombre del paquete a `Bank Server Java`.

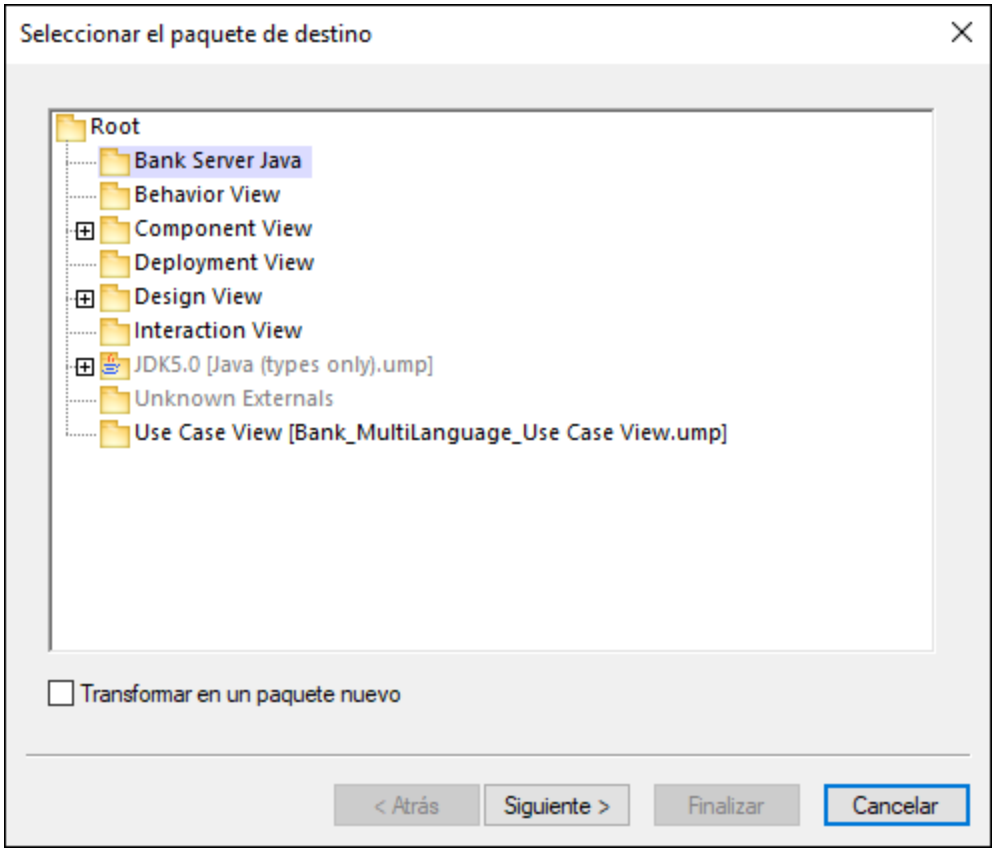


Ahora puede ejecutar la transformación de C# en Java:

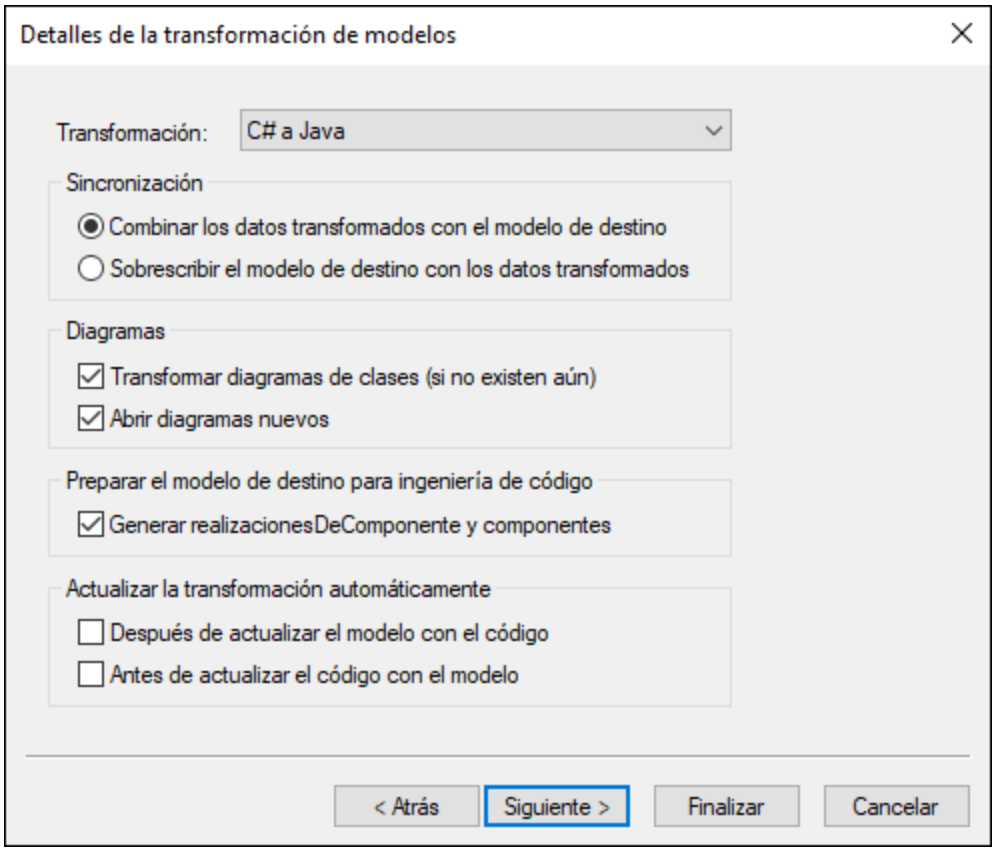
1. Haga clic con el botón derecho en el paquete de origen "Bank Server" y seleccione **Transformaciones de modelos** del menú contextual.



2. Cuando UModel le pida que seleccione un paquete de destino, seleccione el paquete "Bank Server Java" que creó antes y haga clic en **Siguiente**.



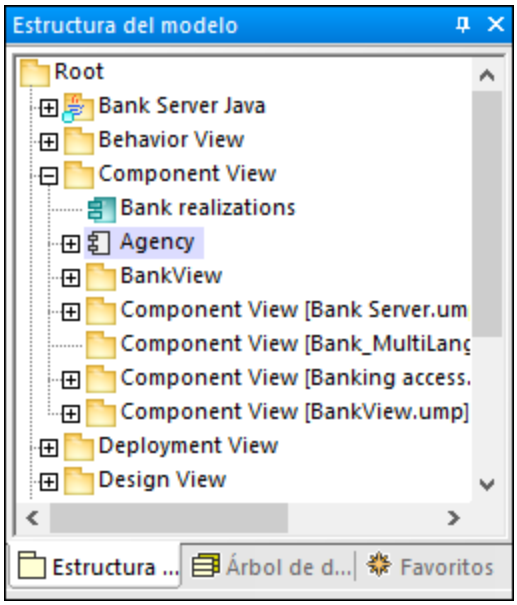
3. Seleccione **C# a Java** como tipo de transformación. No cambie la configuración por ahora.



4. Haga clic en **Finalizar**. Cuando UModel le pregunte si quiere incluir el perfil *Model Transformation Profile de UModel* haga clic en **Aceptar**.

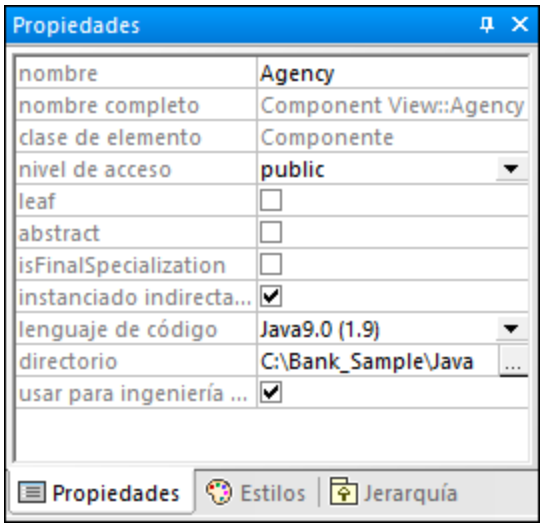
Cuando termine la transformación observará los siguientes cambios en el proyecto:

- Se genera un paquete de diagramas llamado "Model transformation from Bank Server to Bank Server Java" en el paquete de destino y se abre automáticamente. Este diagrama ilustra la transformación que acaba de tener lugar y también le permite cambiar la configuración que definió antes, como se muestra más abajo.
- El paquete de destino "Bank Server Java" incluye todos los elementos contiene todos los elementos transformados desde el modelo de origen C# y adaptados a Java. Por ejemplo, si abre el diagrama "Bank Server" notará que contiene el tipo `boolean`, cuando en C# el tipo es `bool`.
- En la ventana *Estructura del modelo* el paquete "Component View" incluye un nuevo componente: "Agency". Este componente se generó automáticamente porque la opción **realizacionesDeComponentes y componentes** estaba habilitada y el paquete de origen contiene el espacio de nombres `Agency`. El nuevo componente define las opciones de ingeniería de código para el modelo de destino (en este caso, Java).



Configuremos ahora el modelo de destino en Java para ingeniería de código.

1. Haga clic en el componente "Agency" del paquete "Component View".
2. En la propiedad **directorio** de la ventana Propiedades, introduzca la carpeta en la que se debe generar el código (por ejemplo, **C:\Bank_Sample\Java**, siempre que esta carpeta exista).

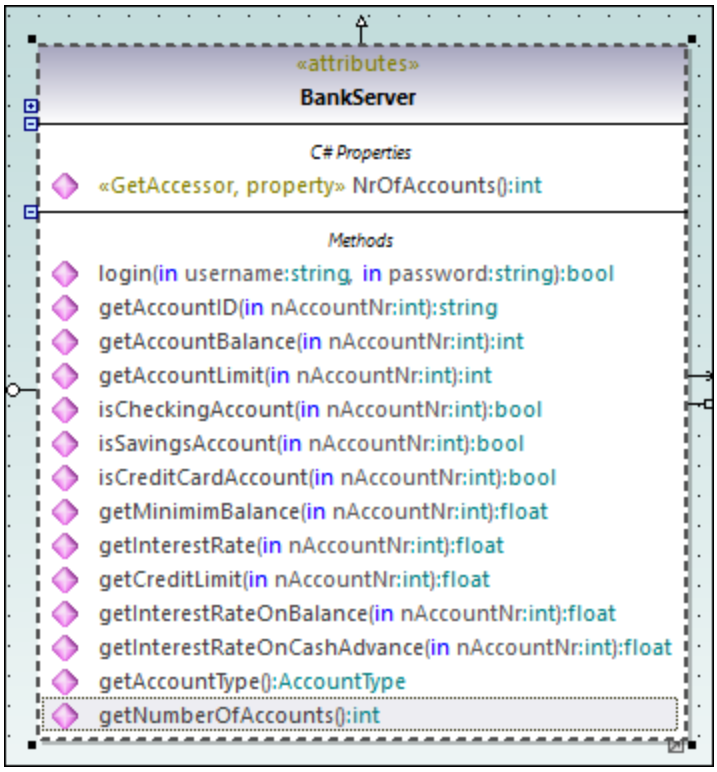


A continuación generaremos el código Java a partir del modelo de destino:

1. Haga clic con el botón derecho en el paquete "Bank Server Java" y seleccione **Ingeniería de código | Combinar código de programa con Paquete de Umodel**.
2. Haga clic en Aceptar para confirmar la configuración de sincronización.

En este momento su proyecto de UModel contiene tanto el modelo de origen en C# Bank Server como el modelo de destino en Java (y ambos están configurados para que generen código). A partir de ahora se pueden mantener ambos proyectos sincronizados (manual o automáticamente) incluso aunque siga trabajado en el

modelo de origen en C#. Para ilustrar esto, abra el diagrama "Bank Server" que encontrará en el paquete de origen en C# y añada a la clase `BankServer` una nueva operación llamada `getNumberOfAccounts` que devolverá un valor `int`.



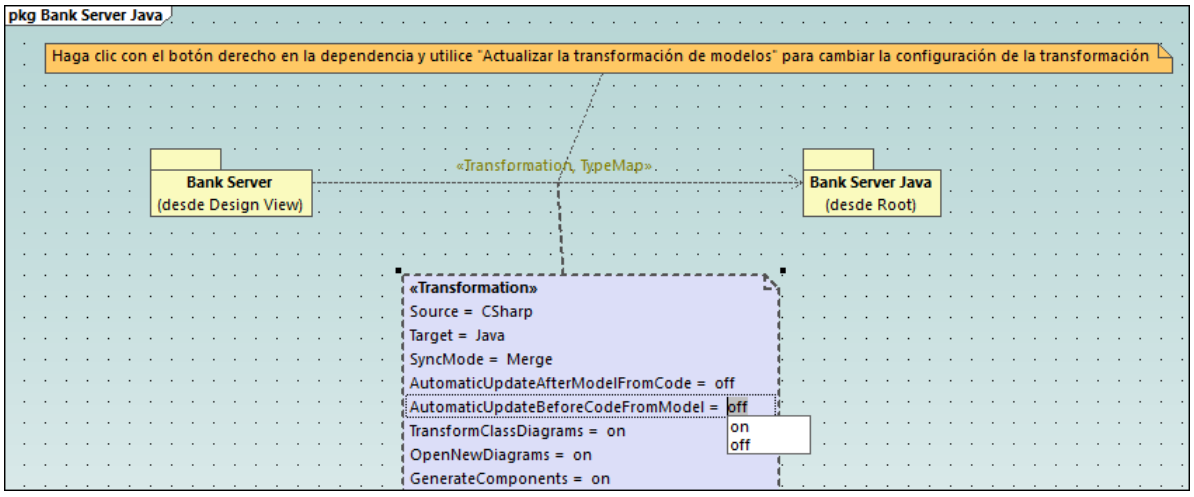
Esta modificación se puede propagar manualmente al modelo de destino:

1. Haga clic con el botón derecho en el paquete de origen "Bank Server" y seleccione **Actualizar la transformación de modelos | Desde "BankServer" hasta "BankServer Java**..."..
2. Haga clic en **Finalizar**.

Ahora la operación `getNumberOfAccounts` que añadimos antes desde el modelo en C# se ha combinado con el modelo de destino en Java.

Por último configuraremos la transformación para que haya actualizaciones automáticas de C# a Java cada vez que modifique el modelo en C# o cuando importe el código fuente en C# al modelo en C#.

1. Abra el diagrama de paquetes "Model transformation from Bank Server to Bank Server Java".
2. Haga doble clic en el valor etiquetado `AutomaticaUpdateAfterModelFromCode` y establézcalo en "on".
3. Repita el paso anterior para el valor etiquetado `AutomaticUpdateBeforeCodeFromModel`.



Para activar las actualizaciones automáticas:

1. Regrese a la clase BankServer en el modelo de origen en C# y elimine la operación getNumberOfAccounts.
2. Haga clic con el botón derecho en el paquete Bank Server C# y ejecute el comando **Combinar código de programa con Paquete de Umodel** o **Combinar el proyecto de UModel con el código de programa**.

Al estar habilitadas las actualizaciones automáticas, en la clase de destino BankServer habrá ocurrido el mismo cambio automáticamente.

7.4 Ejemplo: convertir la estructura de una BD Access en SQLite

Este ejemplo explica cómo convertir un modelo de base de datos de un tipo en otro. En concreto, explica cómo leer la estructura de una base de datos Microsoft Access y pasarla a un modelo UML para luego combinarla con una base de datos SQLite que ya existe. Al final del ejemplo la estructura de la base de datos Access de origen se replicará en la base de datos SQLite de destino. Tenga en cuenta que aunque en este ejemplo se usen las bases de datos Microsoft Access y SQLite, el mismo mecanismo que explicamos en este ejemplo se puede aplicar al convertir otros tipos de datos compatibles con UModel (*véase Trabajar con bases de datos en UModel*).

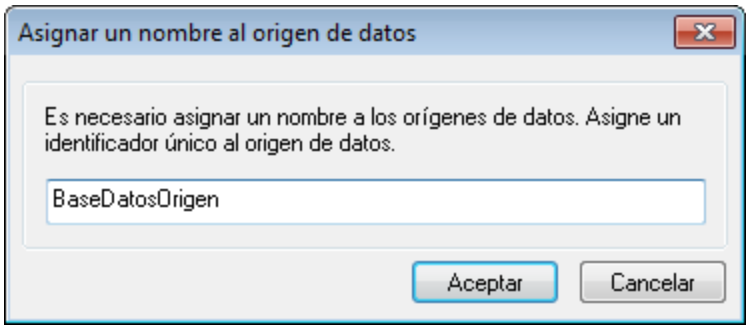
Para el ejemplo necesitará estos archivos de la carpeta **C:\Usuarios\... \Documentos\Altova\UModel2023\UModelExamples\Tutorial**:

- **Nanonull.mdb:** la base de datos Microsoft Access de origen
- **Nanonull.sqlite:** la base de datos SQLite de destino

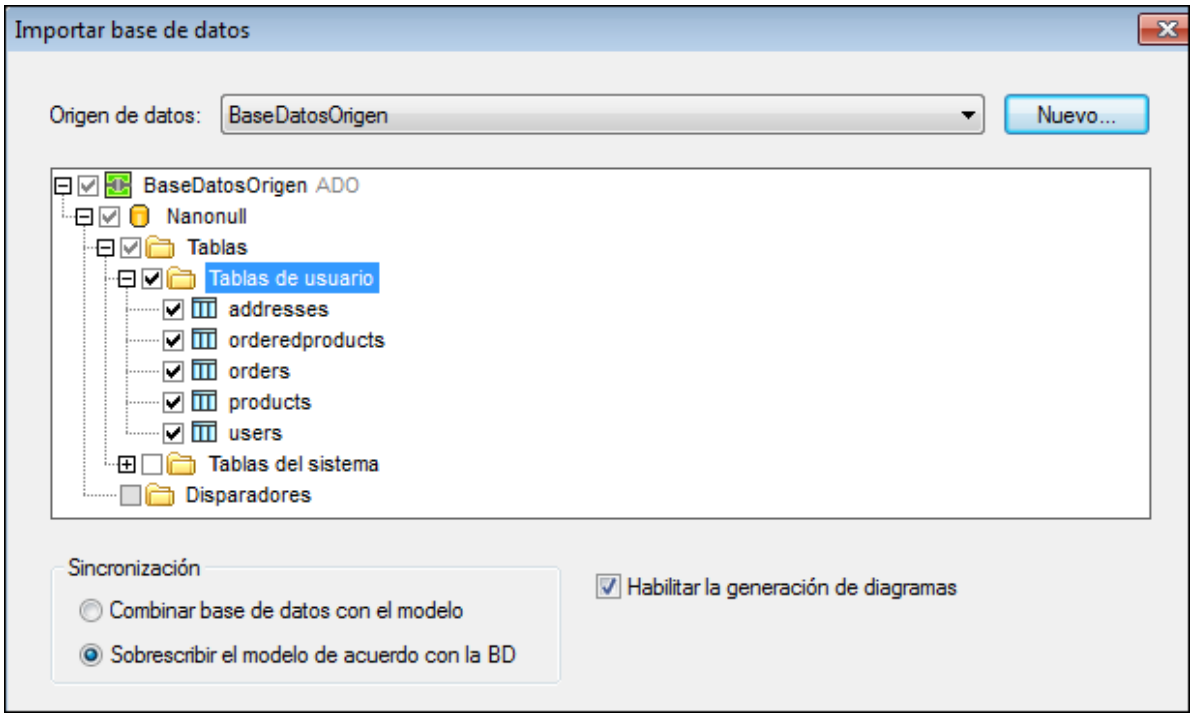
Nota: antes de comenzar con el ejemplo recomendamos que cree una copia de seguridad del archivo de base de datos de muestra Nanonull.sqlite porque su contenido se habrá modificado una vez haya seguido las instrucciones del ejemplo.

Paso nº1: importar la base de datos de origen en UModel

1. Haga clic en el comando de menú **Proyecto | Importar base de datos SQL** y siga los pasos del asistente para conectarse a la base de datos Microsoft Access de origen (**Nanonull.mdb**). Consulte el apartado Conectarse a una base de datos para más información.
2. Cuando la aplicación solicite un nombre para el origen de los datos, indique un nombre descriptivo, como `BaseDatosOrigen`.



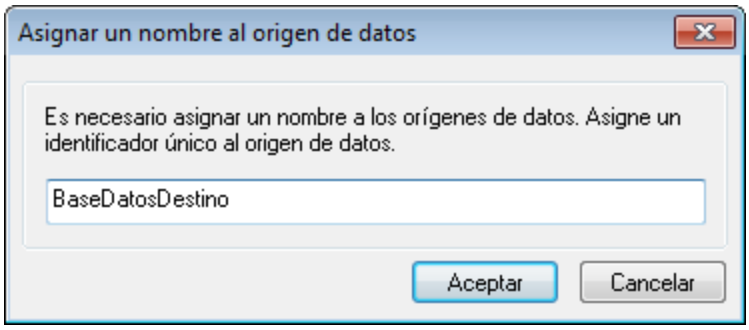
3. Seleccione los objetos de la base de datos que desea importar al modelo y haga clic en **Finalizar**.



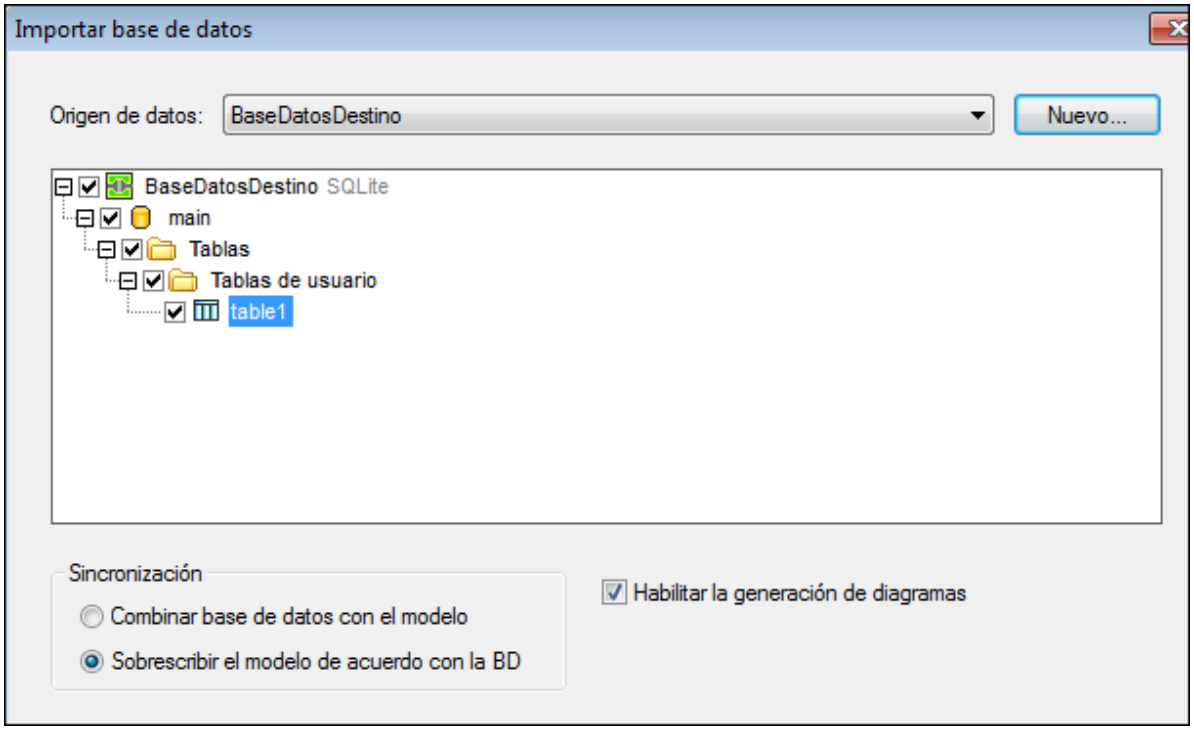
Observe que ahora hay un paquete llamado `BaseDatosOrigen` en la ventana *Estructura del modelo*, dentro del paquete raíz **Root**.

Paso nº2: importar la base de datos de destino en UModel

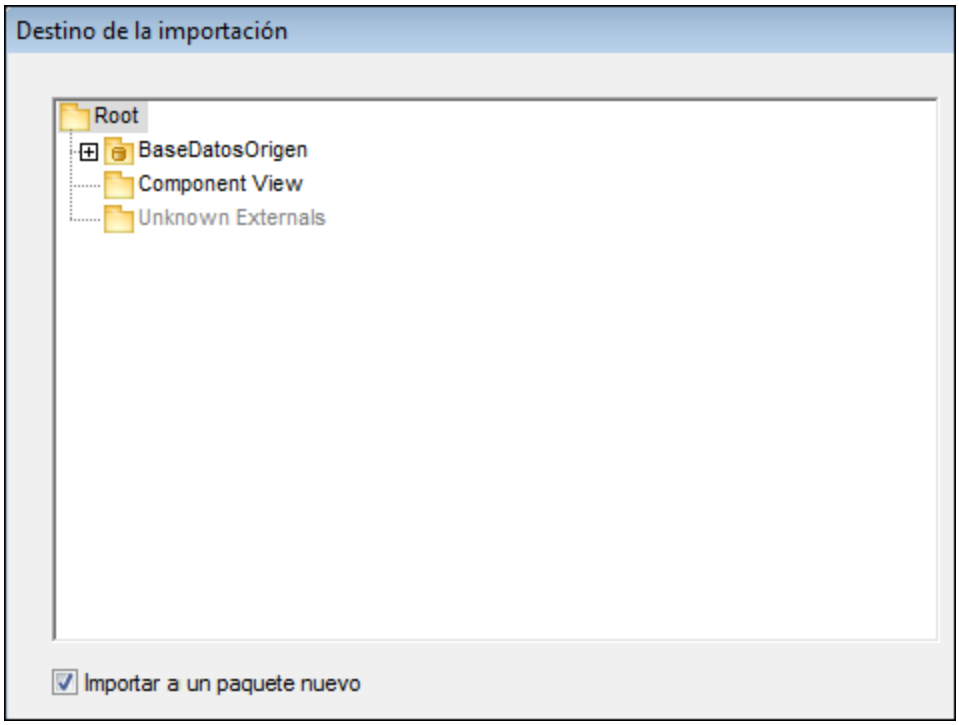
1. Haga clic en el comando de menú **Proyecto | Importar base de datos SQL** y siga los pasos del asistente para conectarse a la base de datos SQLite de destino (**Nanonull.sqlite**).
2. Cuando la aplicación solicite un nombre para el destino de los datos, indique un nombre descriptivo, como `BaseDatosDestino`.



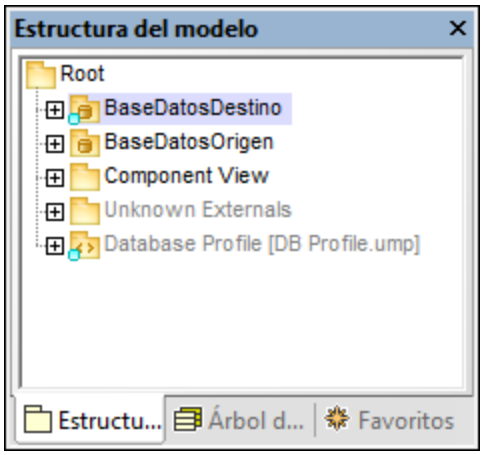
3. Seleccione los objetos de la base de datos que desea importar al modelo y haga clic en **Finalizar**.



4. Cuando la aplicación solicite un paquete de destino, marque la casilla *Importar a un paquete nuevo* y haga clic en **Finalizar**.

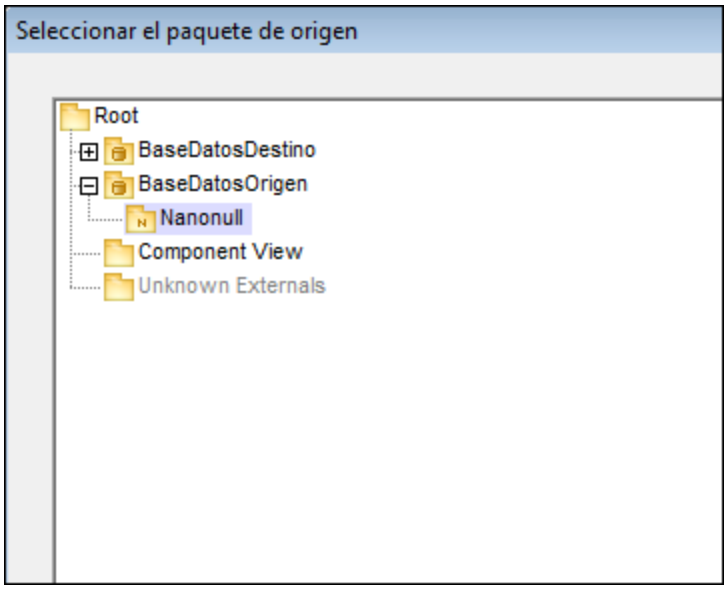


Llegados a este punto el nuevo paquete `BaseDatosDestino` aparece en la ventana *Estructura del modelo*, dentro del paquete raíz **Root**.

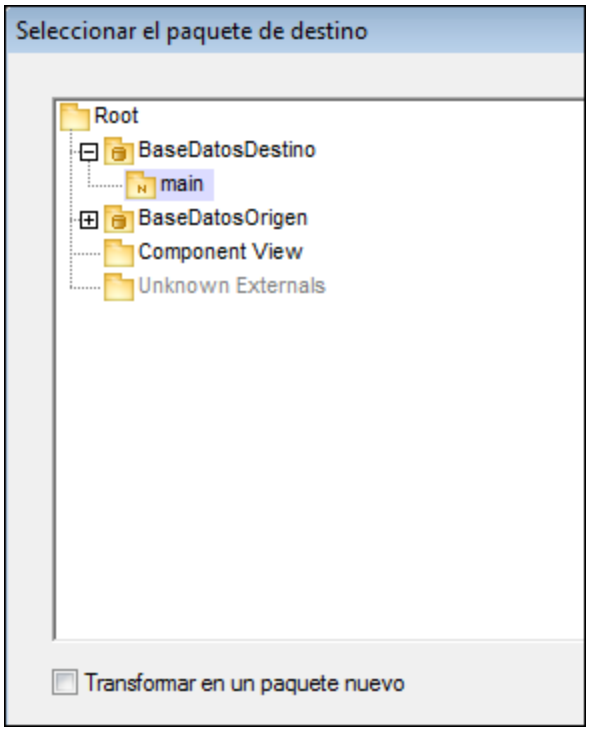


Paso nº3: ejecutar la transformación del modelo desde la base de datos de origen a la de destino

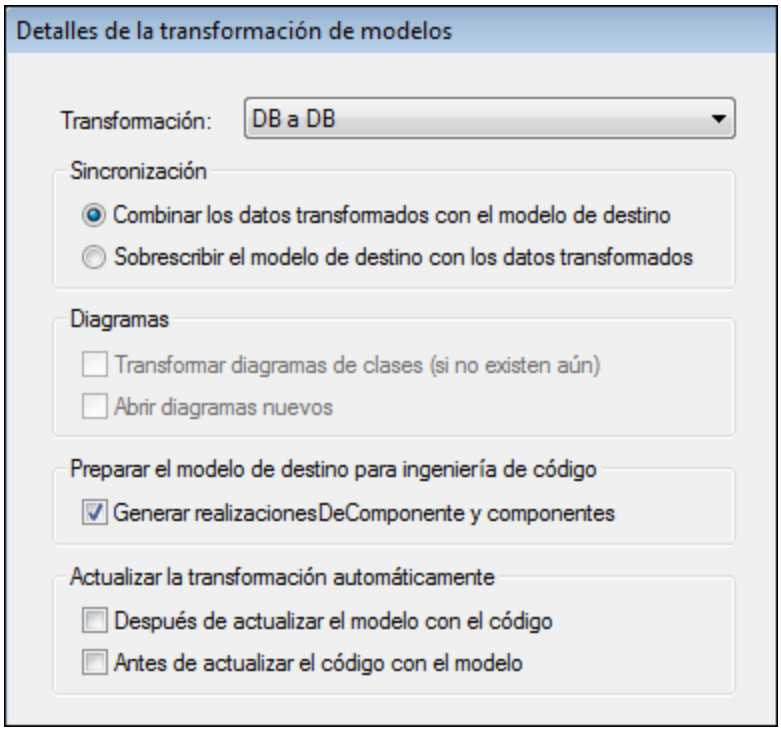
1. Haga clic en el comando de menú **Proyecto | Transformación de modelos**.
2. En el cuadro de diálogo "Seleccionar el paquete de origen" haga clic en el paquete **BaseDatosOrigen / Nanonull** y después en **Siguiente**.



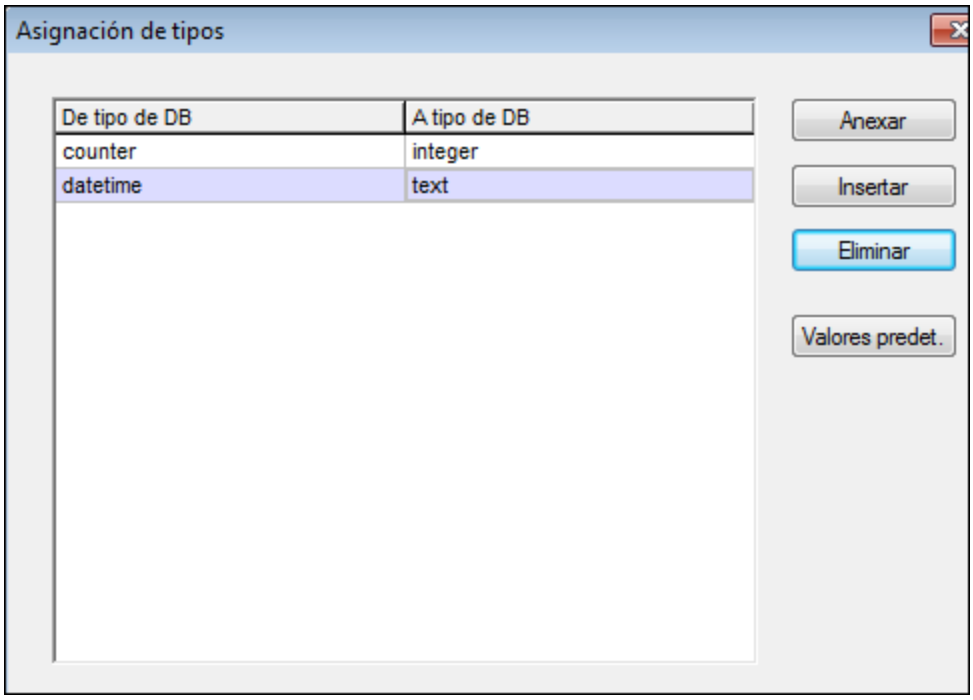
3. En el cuadro de diálogo "Seleccionar el paquete de destino" haga clic en el paquete **BaseDatosDestino / main** y después en **Siguiente**.



4. En el cuadro de diálogo "Detalles de la transformación de modelos" seleccione el tipo de transformación **BD a BD** y haga clic en **Siguiente**.

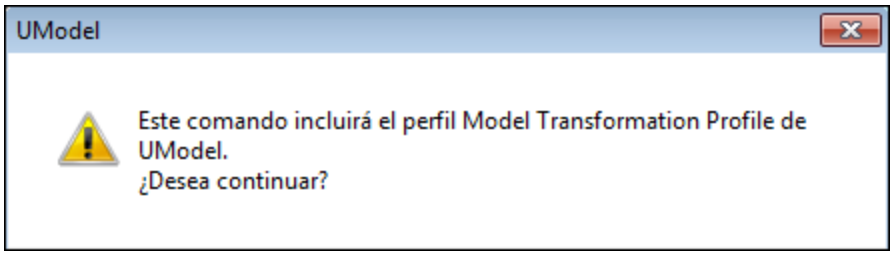


5. En el cuadro de diálogo "Asignación de tipos" repase los tipos de datos y realice los cambios que sean necesarios. Para este ejemplo asignaremos solamente los tipos de datos propios de Microsoft Access que no existen en SQLite, como puede ver aquí:

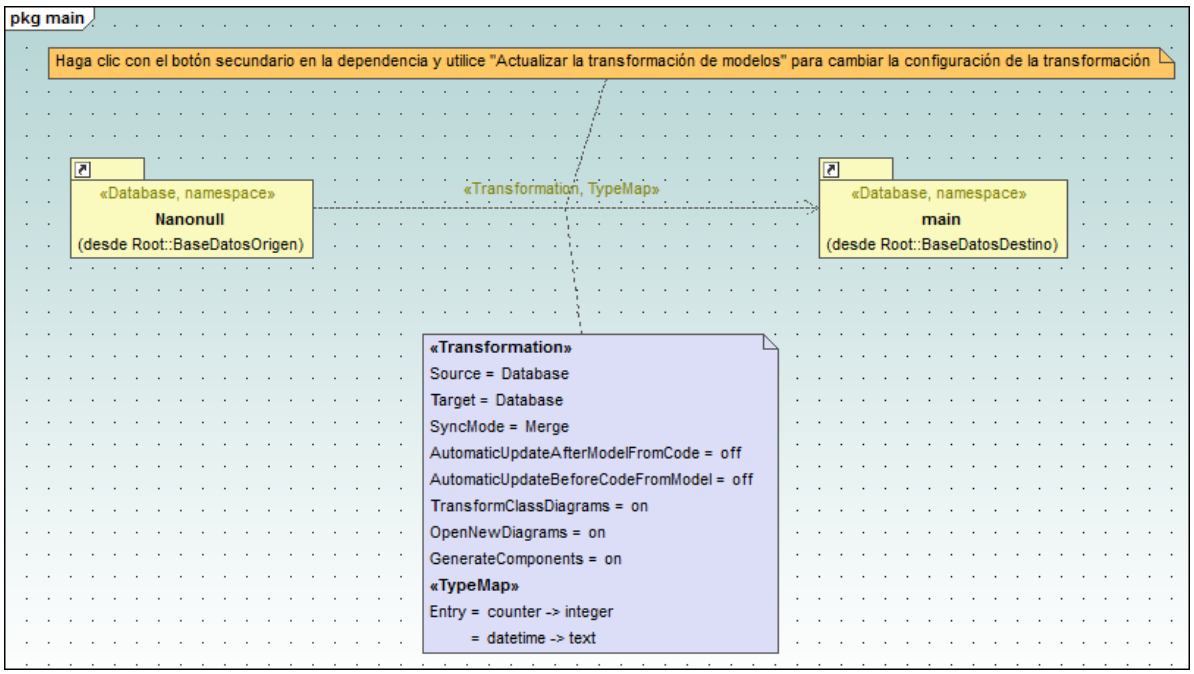


Por lo general, asegúrese de que en la columna de la izquierda aparecen tipos de datos compatibles con la base de datos de origen y en la derecha tipos de datos compatibles con la base de datos de destino. Para agregar o eliminar asignaciones de tipos utilice los botones **Anexar**, **Insertar** y **Eliminar** del cuadro de diálogo.

6. Haga clic en **Finalizar**. Después aparece un mensaje (*imagen siguiente*). Haga clic en **Aceptar** para terminar.

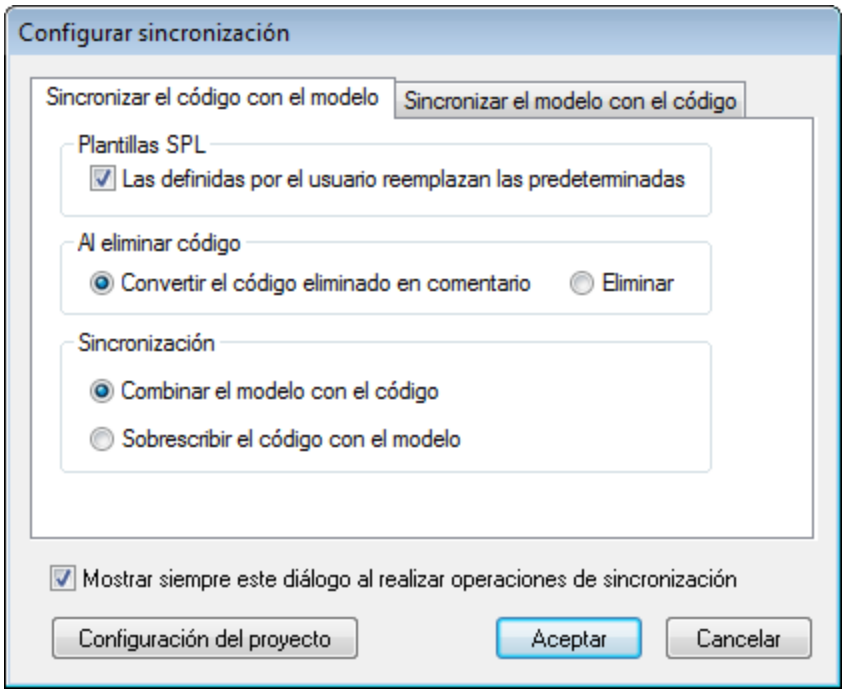


UModel genera un diagrama de dependencias donde puede repasar (y modificar) todas las opciones definidas hasta ahora, incluidas las asignaciones de tipos de datos. Para este ejemplo dejamos las opciones tal y como están.

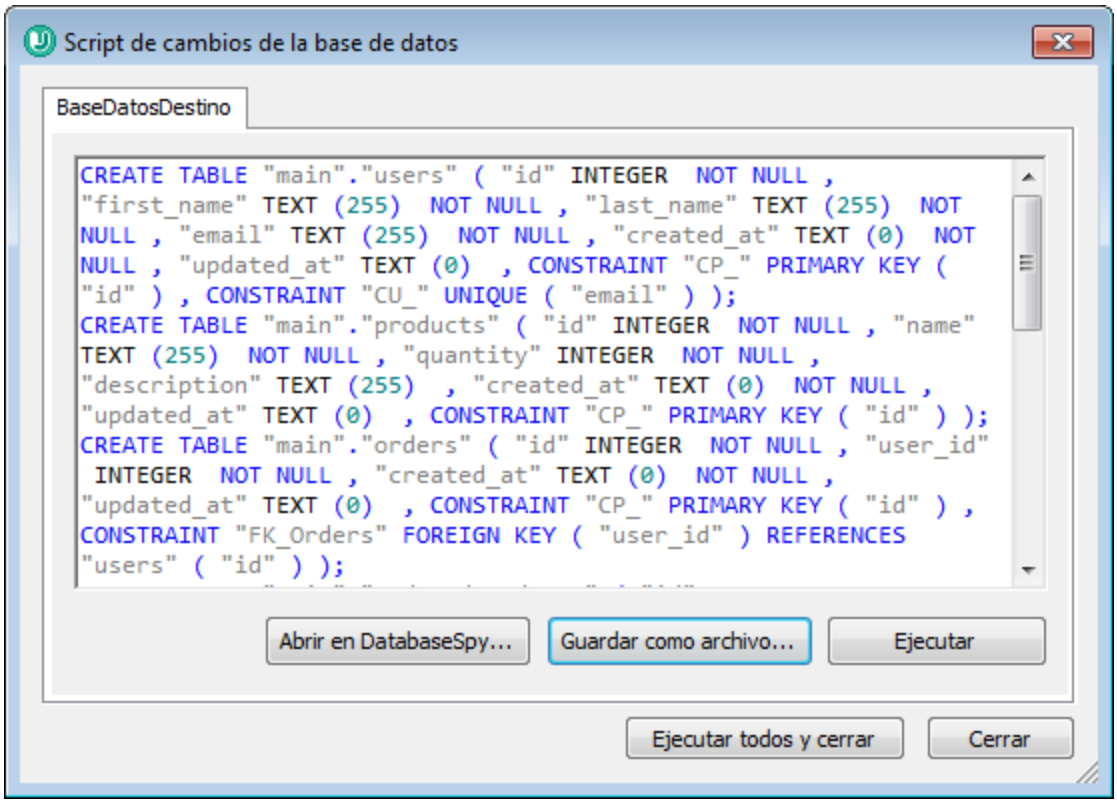


Paso nº4: combinar el código de programa con el proyecto de UModel

1. Haga clic en el comando de menú **Proyecto | Combinar el código de programa con el proyecto de UModel**.
2. No modifique las opciones de configuración predeterminadas y haga clic en **Aceptar**.



UModel genera un script de actualización de base de datos y muestra una vista previa en un cuadro de diálogo (*imagen siguiente*). Ahora puede ejecutar el script en UModel directamente o guardarlo en un archivo. Si tiene Altova DatabaseSpy instalado también puede abrir y ejecutar el script en Altova DatabaseSpy, que ofrece una avanzada interfaz de administración de bases de datos.



Lo más recomendable es repasar el script generado y, si es necesario, modificarlo antes de ejecutarlo en la base de datos de destino.

Si la base de datos de origen contiene nombres de objetos (p. ej. índices o claves foráneas) que no son únicos a nivel de base de datos, el script de actualización de base de datos no se ejecutará correctamente. Por ejemplo, una base de datos Microsoft Access puede tener varios índices que se llamen igual. Si la base de datos de destino no acepta nombres repetidos para los índices deberá editar el script de actualización para que todos los nombres de objeto sean únicos.

En ocasiones también deberá actualizar el script para modificar el tamaño de las columnas en función de los requisitos de la base de datos de destino.

Tras ejecutar el script (desde UModel o en una herramienta como Altova DatabaseSpy) las tablas, columnas, índices y restricciones de clave necesarias se recrearán en la base de datos SQLite de destino. Recuerde que SQLite (versión 3.6.19) acepta los nombres de restricciones de clave foráneas dados por la instrucción SQL pero no ofrece ningún mecanismo para recuperarlos de la base de datos (las restricciones de clave foránea se recuperan con un nombre aleatorio y no con su nombre real). Para conseguir que el modelo de base de datos muestre los nombres de objeto reales dados por la base de datos deberá llevar a cabo una actualización inversa del modelo a partir de la base de datos. Esto se hace con el comando de menú **Proyecto | Combinar el proyecto de UModel con el código de programa**. El modelo se actualizará y mostrará los nombres de objeto tal y como vienen dados por la base de datos.

8 Generar documentación UML

Sitio web de Altova: Documentación de proyectos UML

Ejecute el comando de menú **Proyecto | Generar documentación** para generar documentación detallada sobre el proyecto UML en formato HTML, Microsoft Word, RTF o PDF. La documentación generada con este comando se puede modificar y utilizar sin permiso de Altova.

Notes

- Para generar documentación en formato PDF o para personalizar la documentación generada debe estar instalado y tener licencia Altova StyleVision (https://www.altova.com/es/stylevision).
- Para generar documentación en formato Word, necesita tener MS Word 2000 (o superior).

La documentación abarca los elementos de modelado seleccionados por el usuario en el cuadro de diálogo "Generar documentación". Además, puede generar la documentación con un diseño fijo o indicar una hoja de estilos de StyleVision (SPS) personalizada. Usar una hoja de estilos de StyleVision permite personalizar la documentación generada (*véase Hojas de estilos definidas por el usuario*).

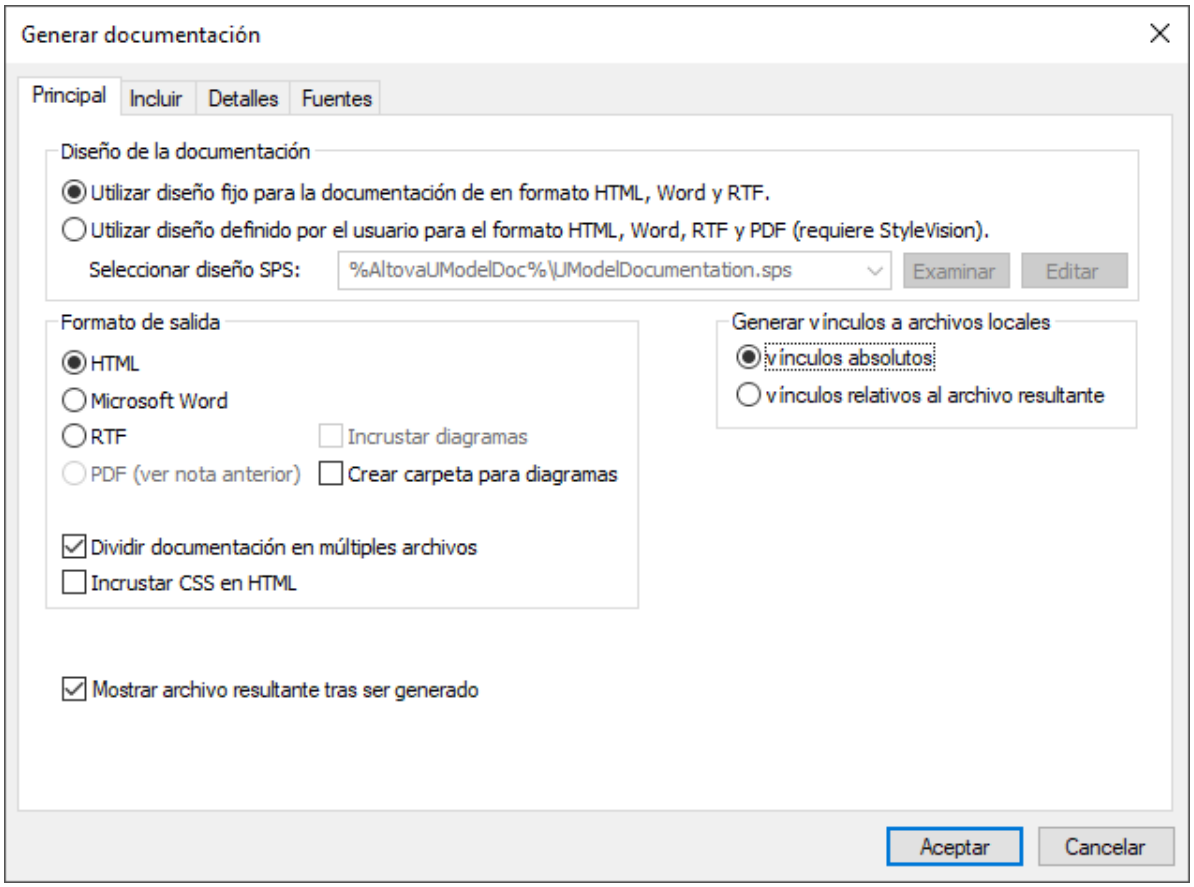
También puede crear documentación parcial de elementos de modelado. Para ello, haga clic con el botón derecho en un elemento (o en varios, usando **Ctrl+clic**) en la Estructura del modelo y seleccione **Generar documentación**. Esos elementos pueden ser una carpeta, una clase, una interfaz, etc. Las opciones de documentación son las mismas en ambos casos.

Los elementos relacionados contienen hipervínculos en el resultado generado, lo que permite navegar entre componentes. Todos los hipervínculos creados de forma manual aparecen también en la documentación.

Si un proyecto contiene perfiles de UModel (como C#, Java, VB.NET, etc.), la documentación generada los incluirá si se habilita la opción **Subproyectos incluidos** en la pestaña *Incluir* (*véase el apartado Documentation Generation Options*).

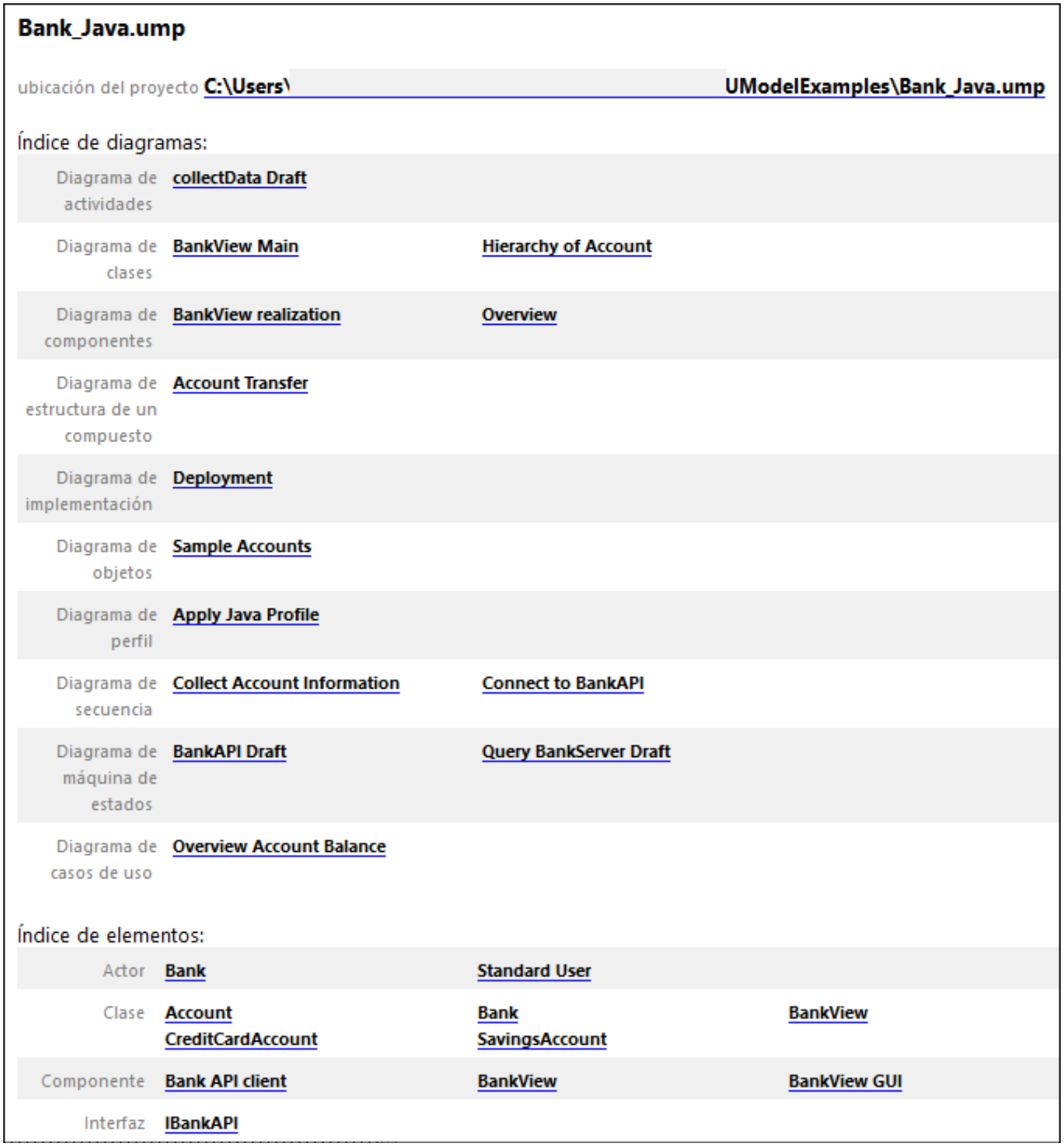
Para generar documentación:

1. Abra un proyecto (por ejemplo **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Bank_Java.ump**).
2. En el menú **Proyecto**, haga clic en **Generar Documentación**.



3. Seleccione un formato de salida (HTML, RTF, PDF).
4. También puede personalizar las opciones de generación (véaseOpciones de generación de documentación).
5. Haga clic en **Aceptar** y escoja una carpeta de destino para que se generen los resultados.

La imagen siguiente muestra un fragmento de la documentación generada con el diseño fijo de UModel desde el archivo de proyecto **Bank_Java.ump**.



Bank_Java.ump

ubicación del proyecto **C:\Users\ UModelExamples\Bank_Java.ump**

Índice de diagramas:

Diagrama de actividades	**collectData Draft**		
Diagrama de clases	**BankView Main**	**Hierarchy of Account**	
Diagrama de componentes	**BankView realization**	**Overview**	
Diagrama de estructura de un compuesto	**Account Transfer**		
Diagrama de implementación	**Deployment**		
Diagrama de objetos	**Sample Accounts**		
Diagrama de perfil	**Apply Java Profile**		
Diagrama de secuencia	**Collect Account Information**	**Connect to BankAPI**	
Diagrama de máquina de estados	**BankAPI Draft**	**Query BankServer Draft**	
Diagrama de casos de uso	**Overview Account Balance**		

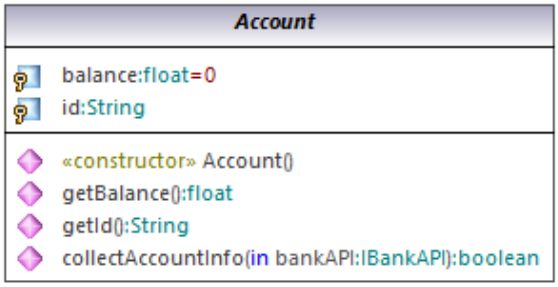
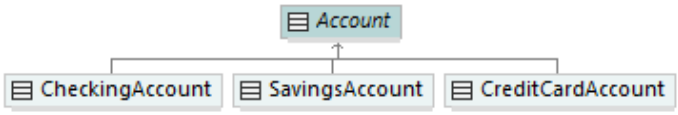
Índice de elementos:

Actor	**Bank**	**Standard User**	
Clase	**Account** **CreditCardAccount**	**Bank** **SavingsAccount**	**BankView**
Componente	**Bank API client**	**BankView**	**BankView GUI**
Interfaz	**IBankAPI**		

Como se ve en la imagen anterior, la documentación generada incluye un índice de diagramas y elementos (con enlaces) en la parte superior del archivo HTML.

La imagen siguiente muestra un fragmento de la documentación generada para la clase `Account`. Observe que los miembros individuales de los diagramas de clases también contienen hipervínculos a sus definiciones. Por ejemplo, al hacer clic en una propiedad u operación se llega a su definición. Las clases jerárquicas, así como el texto subrayado, también contienen hipervínculos.

Clase **Account**

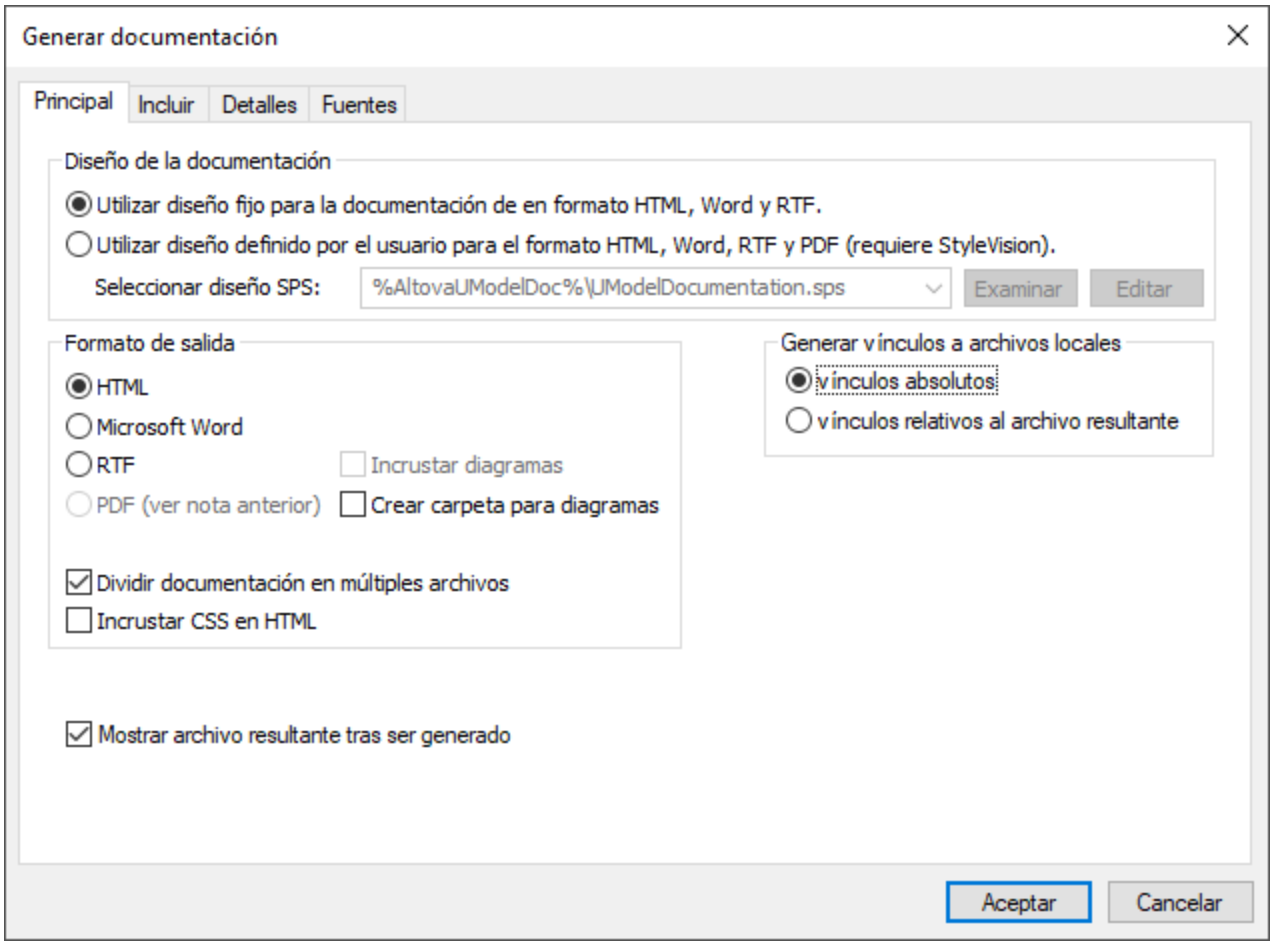
diagrama	
jerarquía	
propietario	**bankview**
propiedades	nombre completo **Design View::BankView::com::altova::bankview::Account** nivel de acceso **public** leaf **false** abstract **true** isFinalSpecialization **false** activo **false** nombre del archivo de código **Account.java** ruta de acceso del archivo de código **C:\UML_Bank_Sample\JavaCode\com\altova\bankview\Account.java** «annotations» **false** «static» **false** «strictfp» **false**

8.1 Con una hoja de estilos SPS predeterminada

Al generar documentación a partir de proyectos de UModel puede configurar varias opciones, como explicamos a continuación. Las opciones están organizadas según la pestaña en la que aparecen en el cuadro de diálogo "Generar documentación".

Pestaña principal

La pestaña *Principal* incluye las opciones generales de generación de documentación.



Diseño de la documentación:

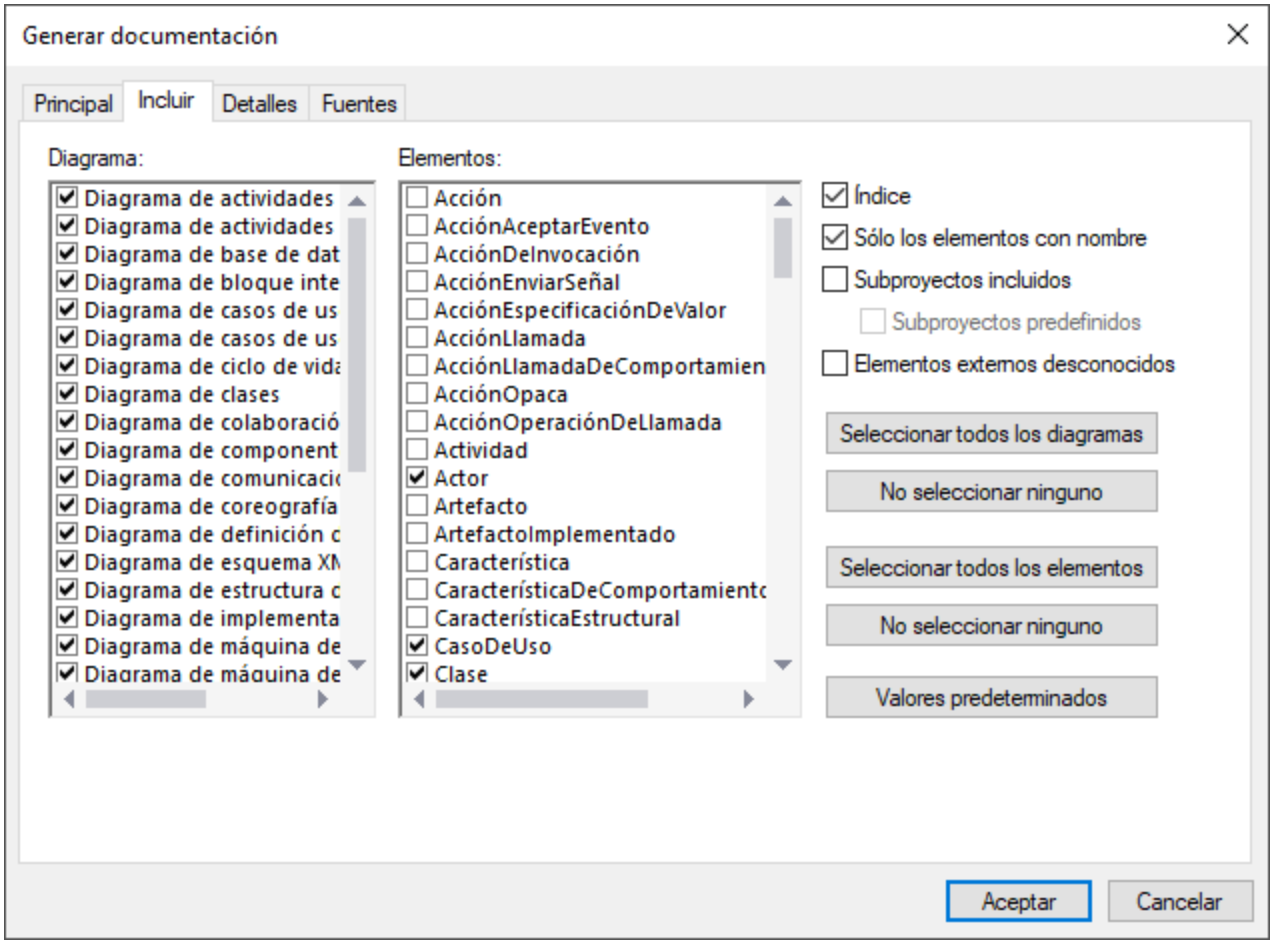
- Seleccione **Utilizar diseño fijo...** para usar el diseño de documentación integrado de UModel.
- Seleccione **Utilizar diseño definido por el usuario...** para generar documentación con formato con la ayuda de una hoja de estilos de StyleVision personalizada (archivo .sps). Nota: esta opción requiere que tenga instalado Altova StyleVision (*véase también* *Personalizar resultados con StyleVision*).
- Haga clic en **Examinar** para navegar hasta una hoja de estilos predefinida.
- Haga clic en **Editar** para abrir la hoja de estilos seleccionada en una ventana de StyleVision.

Formato de salida:

- El formato de salida puede ser uno de estos: HTML, Microsoft Word, RTF o PDF. Los documentos de Microsoft Word se crean con la extensión de archivo .doc si se generan usando un diseño fijado y con la extensión .docx si se generan usando una hoja de estilos de StyleVision. Para poder generar el formato de salida PDF necesita tener instalado Altova StyleVision.
- La opción **Dividir documentación en múltiples archivos** genera un archivo de salida por cada elemento de modelado (clase, interfaz, diagrama, etc.). Desmarque esta casilla si quiere generar un archivo global que contenga todos los elementos de modelado.
- Marque la casilla **Incrustar CSS en HTML** para incrustar el código CSS generado en la documentación HTML. Desmárquela para que el archivo CSS sea externo.
- La opción **Incrustar diagramas** está habilitada para las las opciones de salida Microsoft Word y RTF. Si esta casilla está marcada los diagramas se incrustarán en el archivo generado. Los diagramas se crean como archivos .png y se muestran en el archivo final con enlaces de objetos.
- La opción **Crear carpeta para diagramas** genera una subcarpeta bajo la carpeta de salida seleccionada; esta subcarpeta contiene todos los diagramas.
- La opción **Mostrar archivo resultante tras ser generado** está habilitada para todos los formatos de salida. Si se marca esta casilla el archivo principal generado se muestra en el navegador predeterminado (en el caso de los archivos HTML), en Microsoft Word (en el caso de los archivos Word) o en la aplicación predeterminada (en el caso de los archivos .pdf o .rtf).
- La opción **Generar vínculos a archivos locales** permite especificar si los enlaces generados deben ser absolutos o relativos al archivo de salida.

Pestaña Incluir

Esta pestaña permite seleccionar qué diagramas y elementos de modelado deben aparecer en la documentación.

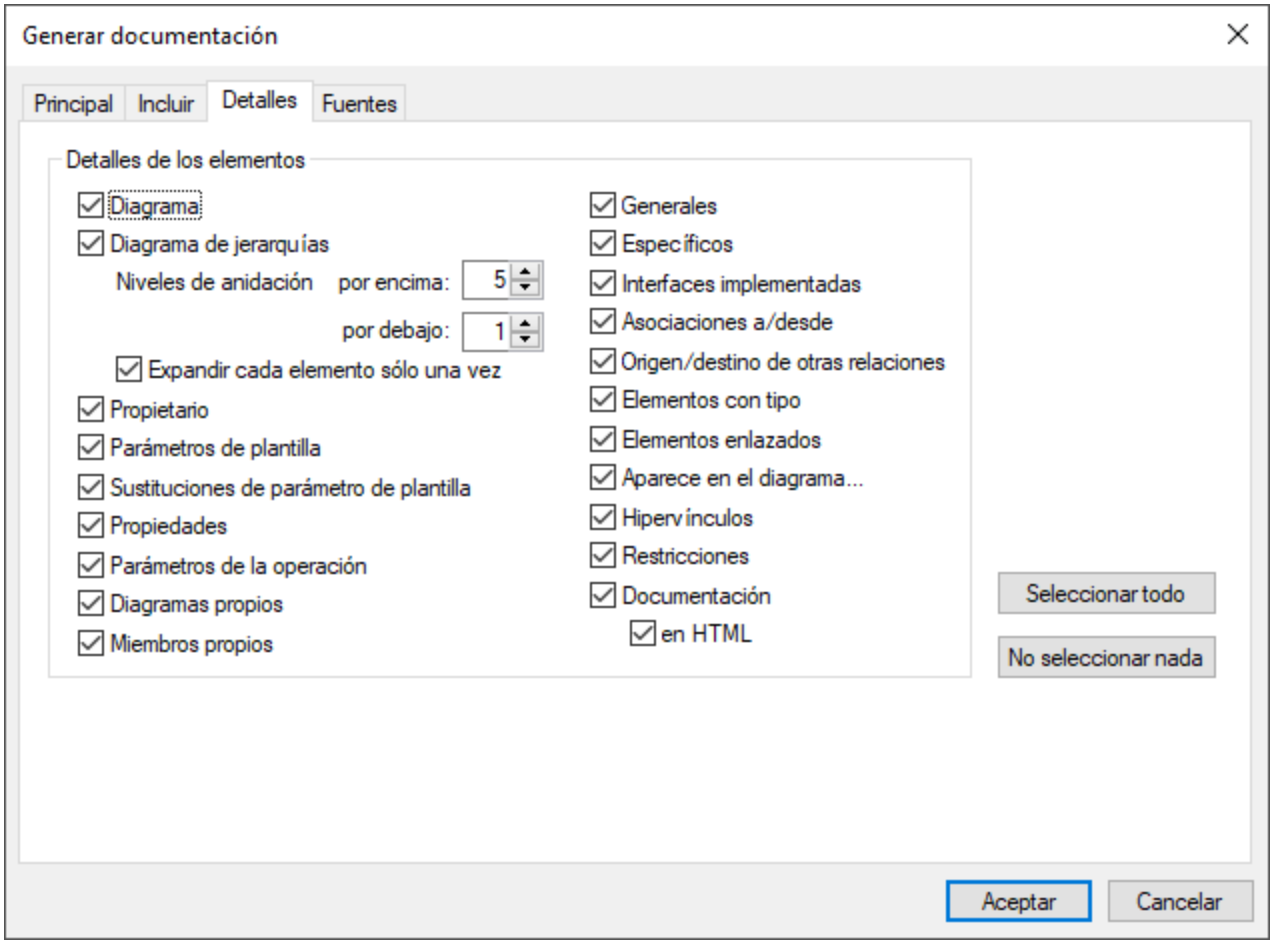


Si no quiere que se incluyan los subproyectos o perfiles en la documentación desmarque la casilla *Incluir subproyectos*. Tenga en cuenta que si desmarca esta casilla no se incluirá ningún elemento o diagrama de los subproyectos en la documentación que se genere. Marque la casilla *Subproyectos predefinidos* para incluir perfiles predefinidos de UModel, como C# o Java. Sin embargo debe tener en cuenta que generar documentación a partir de proyectos predefinidos lleva mucho tiempo. La casilla *Elementos externos desconocidos* hace referencia a elementos cuyo tipo no se puede identificar. Esto suele ocurrir si importa código fuente en UModel sin incluir primero los subproyectos integrados que corresponden a ese lenguaje o a esa versión del lenguaje (consulte Incluir otros proyectos de UModel para más información).

Pestaña Detalles

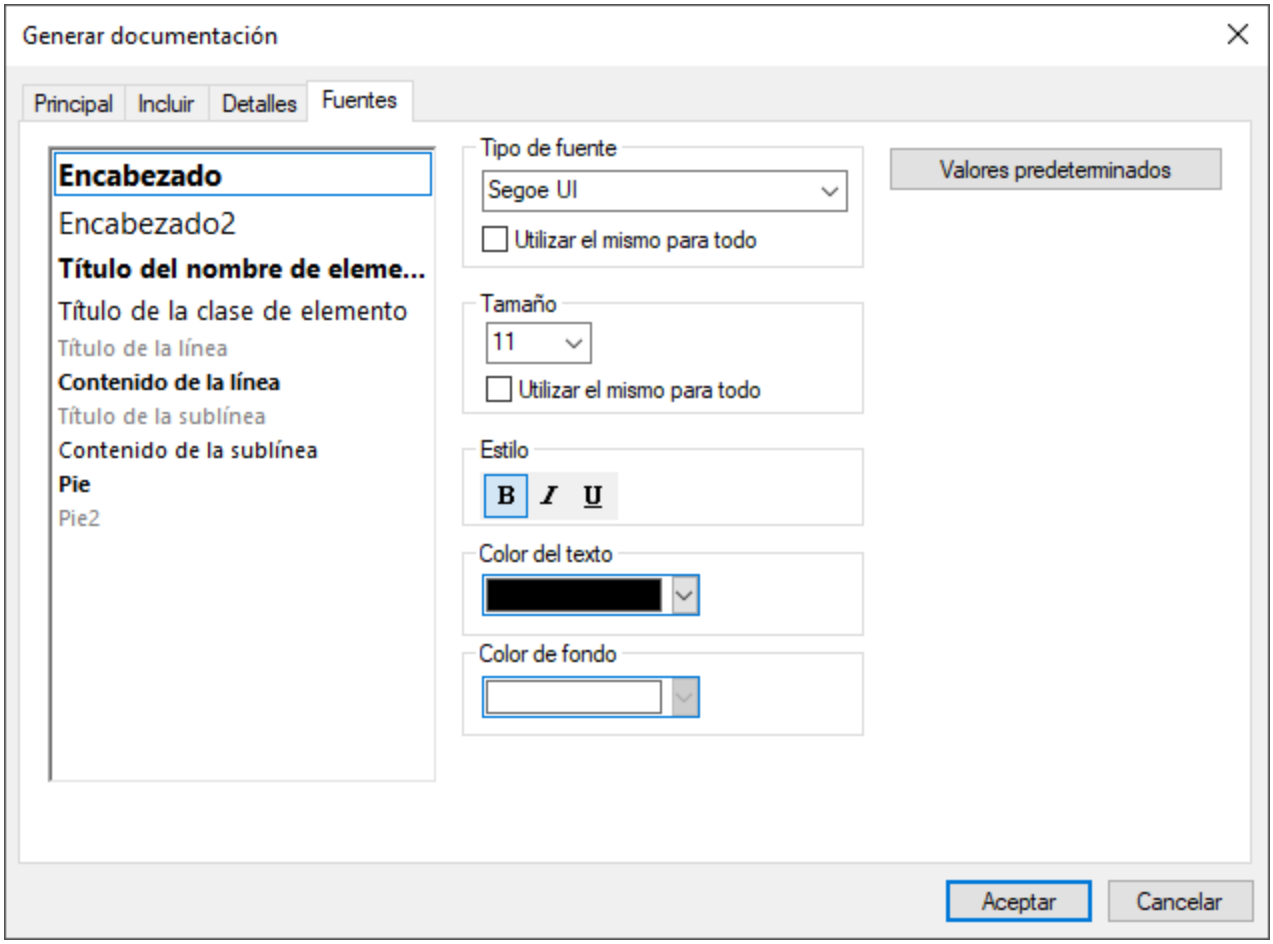
Esta pestaña permite seleccionar los detalles de los elementos que deben aparecer en la documentación.

- Si quiere importar texto de etiquetas XML en su documentación, desmarque la casilla **como HTML**, que se encuentra bajo la opción **Documentación**.
- Los campos **por encima** y **por debajo** permiten definir la profundidad de anidado que aparece por encima o por debajo de la clase actual dentro del diagrama de jerarquía.
- La opción **Expandir cada elemento sólo una vez** determina que solo un clasificador del mismo tipo se expanda en la misma imagen o el mismo diagrama.



Pestaña Fuentes

Esta pestaña permite personalizar las opciones de la fuente de los distintos encabezados y textos.

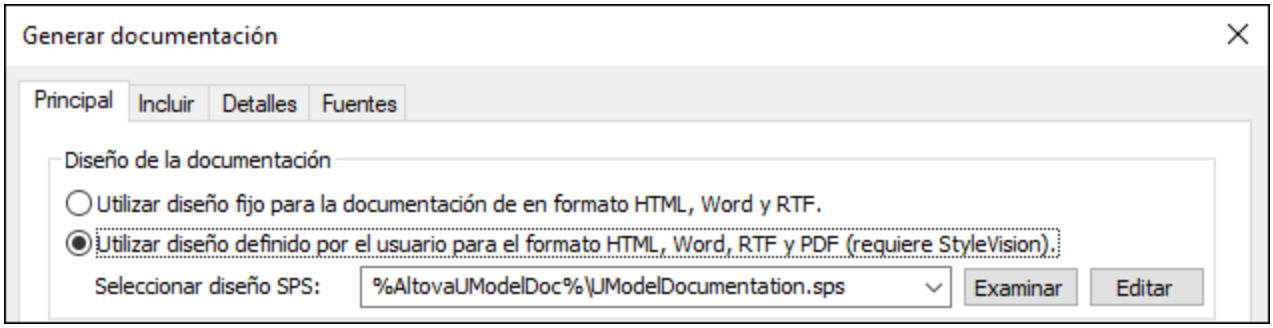

Generar documentación
Principal
Incluir
Detalles
Fuentes
Encabezado
Encabezado2
Título del nombre de eleme...
Título de la clase de elemento
Título de la línea
Contenido de la línea
Título de la sublínea
Contenido de la sublínea
Pie
Pie2
Tipo de fuente
Segoe UI
Utilizar el mismo para todo
Tamaño
11
Utilizar el mismo para todo
Estilo
B
I
U
Color del texto
Color de fondo
Valores predeterminados
Aceptar
Cancelar

8.2 Con hojas de estilos predefinidas por el usuario

Puede personalizar el diseño de la documentación generada con UModel con la ayuda de las hojas de estilos de StyleVision (archivos .sps). Estos archivos se crean en Altova StyleVision (https://www.altova.com/stylevision). La ventaja de usar archivos .sps es que tiene control absoluto sobre el diseño de la documentación. Además, usar archivos .sps permite generar los resultados en PDF.

> Para generar documentación con archivos .sps Altova StyleVision debe estar instalado y contar con una licencia.

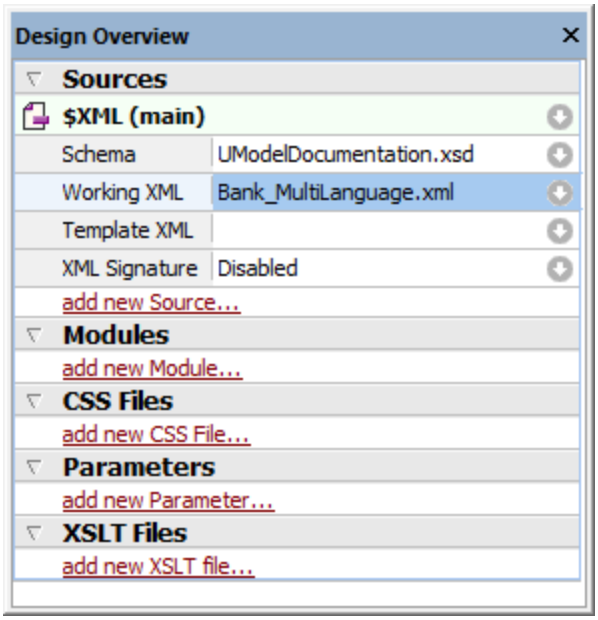
UModel incluye un archivo .sps predefinido, que está disponible en esta ruta: **C:\users\<username\Documents\UModel2023\Documentation\UModel\UModelDocumentation.sps**. Para dar formato a la documentación generada con un archivo .sps, seleccione esta opción al generar la documentación; por ejemplo:



Puede empezar por crear una copia del archivo predefinido **UModelDocumentation.sps** y editarlo en StyleVision. Por ejemplo, puede cambiar el formato actual o añadir enlaces e imágenes al diseño.

Cualquier hoja de estilos de StyleVision está basada en un esquema XML. En el caso de las hojas de estilos que controlan el diseño de la documentación generada con UModel, este esquema está disponible en **C:\users\<username\Documents\UModel2023\Documentation\UModel\UModelDocumentation.xsd**. Observe que el archivo **UModelDocumentation.xsd** hace referencia al archivo **Documentation.xsd**, que está ubicado en la carpeta que está por encima.

Al personalizar archivos .sps en StyleVision para documentación de UModel se debe usar como esquema el archivo **UModelDocumentation.xsd**. La imagen siguiente ilustra la ventana *Vista general del diseño* de StyleVision después de abrir el archivo **UModelDocumentation.sps**. Observe que este usa el archivo de esquema **UModelDocumentation.xsd** y un archivo de trabajo XML, necesario para acceder a la vista previa del diseño. El archivo de trabajo XML está disponible en la subcarpeta **SampleData**, que es relativa al archivo del esquema.



Para ver instrucciones sobre cómo editar archivos .sps consulte la documentación de StyleVision (https://www.altova.com/documentation).

9 Diagramas UML

Sitio web de Altova: Diagramas UML

Hay dos grandes tipos de diagramas UML: (i) los diagramas de estructura, que muestran la vista estática del modelo, y (ii) los diagramas de comportamiento, que muestran su vista dinámica. UModel es compatible con los 14 tipos de diagramas de la especificación UML 2.5 y otros diagramas.

- Diagramas de comportamiento: incluye diagramas de actividades, máquina de estados, máquina de estados de protocolos, casos de uso, interacción, comunicación, interacción global, secuencia y ciclo de vida.
- Diagramas de estructura: incluye diagramas de clases, estructura compuesta, componentes, implementación, objetos y paquetes.
- Otros diagramas: diagramas de esquema XML, diagramas BPMN (Business Processing Modeling Notation), diagramas SysML y diagramas de base de datos.

Nota: en la mayoría de los diagramas de modelado puede pulsar **Ctrl+Entrar** para crear una línea nueva en el nombre de algunos elementos (por ejemplo, en las etiquetas de las líneas de vida de los diagramas de secuencia y de ciclo de vida o en las condiciones de protección, nombres de estado y nombres de actividades).

9.1 Diagramas de comportamiento

Los diagramas de este tipo ilustran las características de comportamiento de un sistema o de un proceso de negocio e incluyen un subconjunto de diagramas que subrayan cómo interactúan los objetos.

Diagrama de actividades
Diagrama de máquina de estados
Diagrama de máquina de estados de protocolos
Diagrama de casos de uso

El subconjunto de diagramas de comportamiento que ilustran cómo interactúan los objetos está compuesto por estos diagramas:

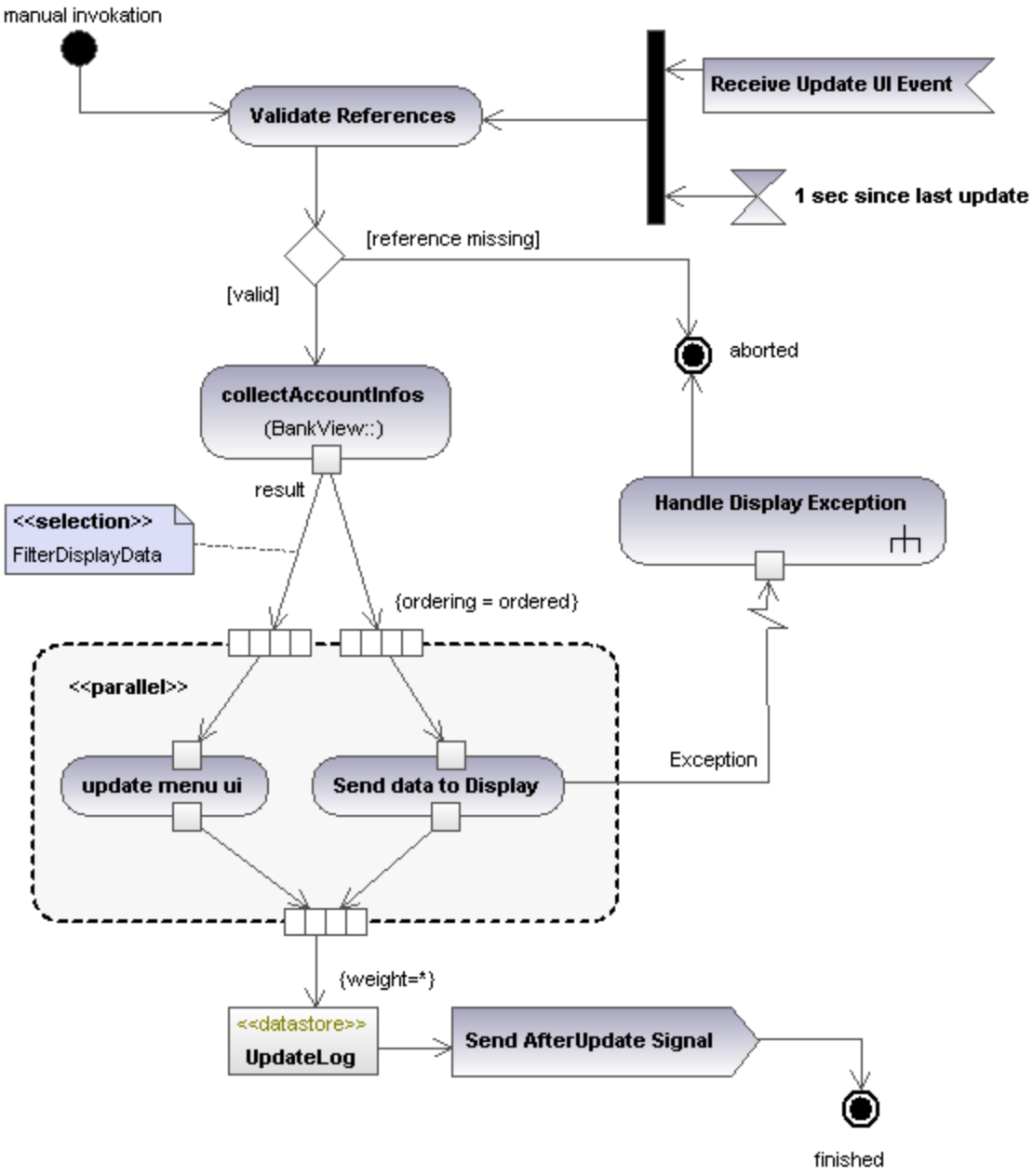
Diagrama de comunicación
Diagrama global de interacción
Diagrama de secuencia
Diagrama de ciclo de vida

9.1.1 Diagrama de actividades

Sitio web de Altova: Diagramas de actividades UML

Los diagramas de actividades sirven para modelar flujos de trabajo de procesos de negocios. Permiten ver qué acciones deben tener lugar y qué dependencias de comportamiento existen. Los diagramas de actividades describen el orden concreto de las actividades y permiten un procesamiento tanto condicional como en paralelo. Los diagramas de actividades son una especie de diagrama de estados, con actividades en lugar de estados.

El diagrama de actividades que aparece a continuación está disponible en el ejemplo **Bank_MultiLanguage.ump**, en la carpeta **...\UModelExamples**.



9.1.1.1 Insertar elementos

Para insertar elementos en el diagrama:

1. Haga clic en el icono pertinente de la barra de herramientas *Diagrama de actividades*.

2. Ahora haga clic en el área de trabajo del diagrama para insertar el elemento.

Para insertar varios elementos del mismo tipo, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos que ya existen hasta el diagrama

1. Busque en la *Estructura del modelo* el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de actividades.

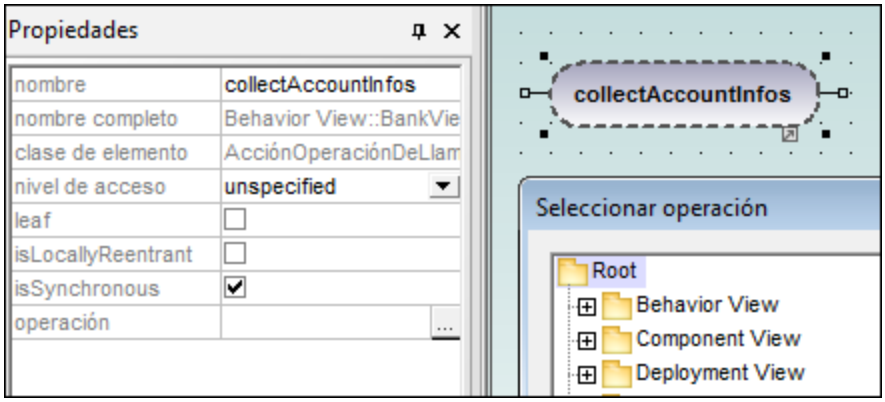
Insertar una acción (ComportamientoDeLlamada)

1. Haga clic en el icono **Acción (ComportamientoDeLlamada)**  de la barra de herramientas y haga clic en el diagrama de actividades para insertar la acción.
2. Escriba el nombre de la Acción (p. ej. Validar referencias) y pulse **Entrar** para confirmar.

Nota: pulse **Ctrl+Entrar** para crear una línea nueva en el nombre de la acción.

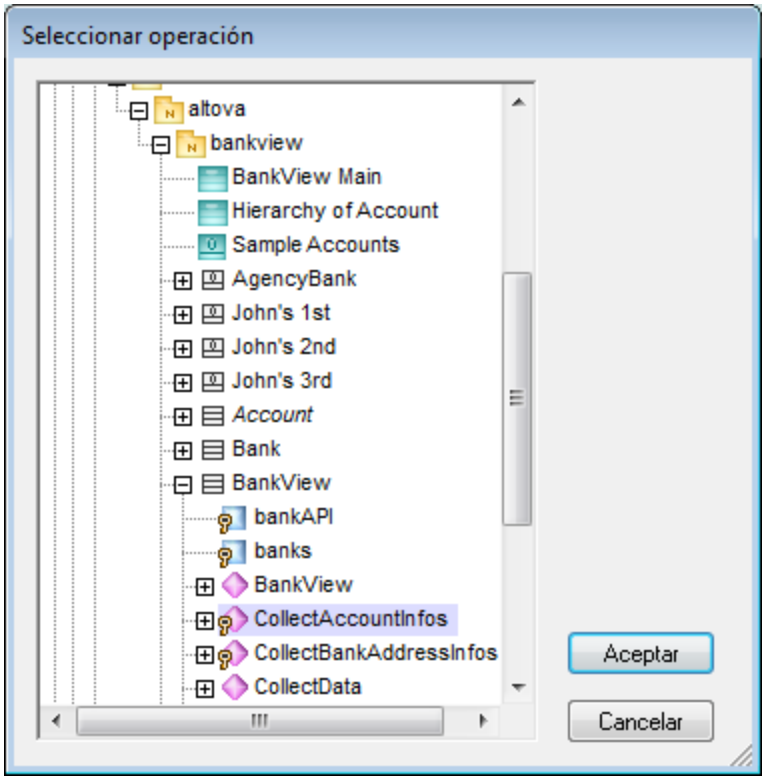
Para insertar una acción (OperaciónDeLlamada) y seleccionar una operación determinada

1. Haga clic en el icono **Acción (OperaciónDeLlamada)**  de la barra de herramientas y haga clic en el diagrama de actividades para insertar la acción.
2. Escriba el nombre de la Acción (p. ej. collectAccountInfos) y pulse **Entrar** para confirmar.
3. En la ventana *Propiedades* haga clic en el botón **Examinar** del campo operation.

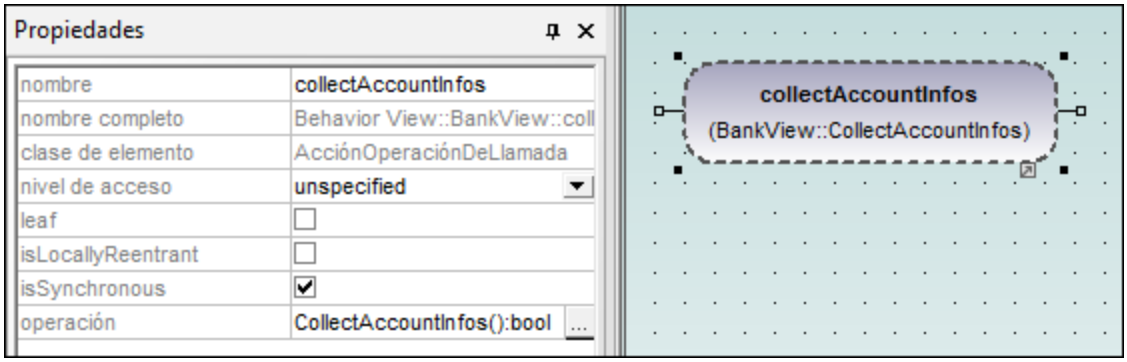


Esto abre el cuadro de diálogo "Seleccionar operación", donde puede seleccionar una operación determinada.

4. Navegue hasta la operación que desea insertar y haga clic en **Aceptar** para confirmar.



Para este ejemplo seleccionamos la operación collectAccountInfos de la clase **`BankView`**.



9.1.1.2 Crear bifurcaciones y convergencias

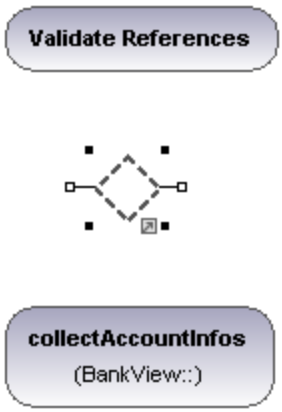
Una rama tiene un flujo de entrada y varios flujos de salida protegidos por guardas. Solo se puede recorrer uno de esos flujos de salida, así que las guardas deben excluirse mutuamente.

En el ejemplo que utilizamos a continuación vamos a validar las referencias de **`BankView`**:

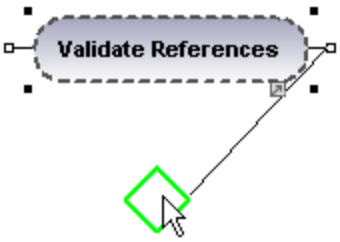
- la rama1 tiene el guarda reference missing, que pasa a la actividad abort.
- la rama2 tiene el guarda valid, que pasa a la actividad collectAccountInfos.

Crear una rama (flujo alterno)

1. Haga clic en el icono **NodoDeDecisión** de la barra de herramientas y haga clic en el área de trabajo para insertarlo en el diagrama de actividades.

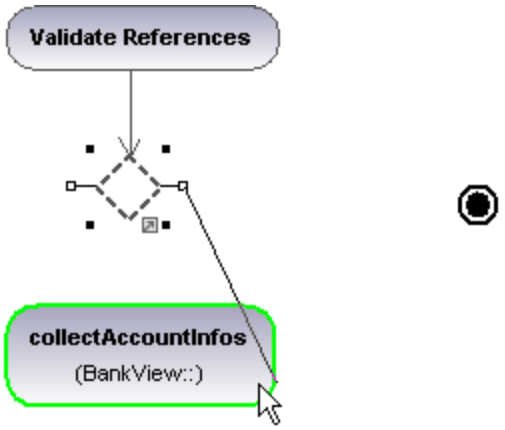


2. Haga clic en el icono **NodoFinalDeActividad**, que representa la actividad abort, e insértelo en el diagrama de actividades.
3. Haga clic en la actividad Validate References y después haga clic en su conector derecho (el controlador FlujoDeControl). Ahora arrastre el conector hasta el elemento NodoDeDecisión.

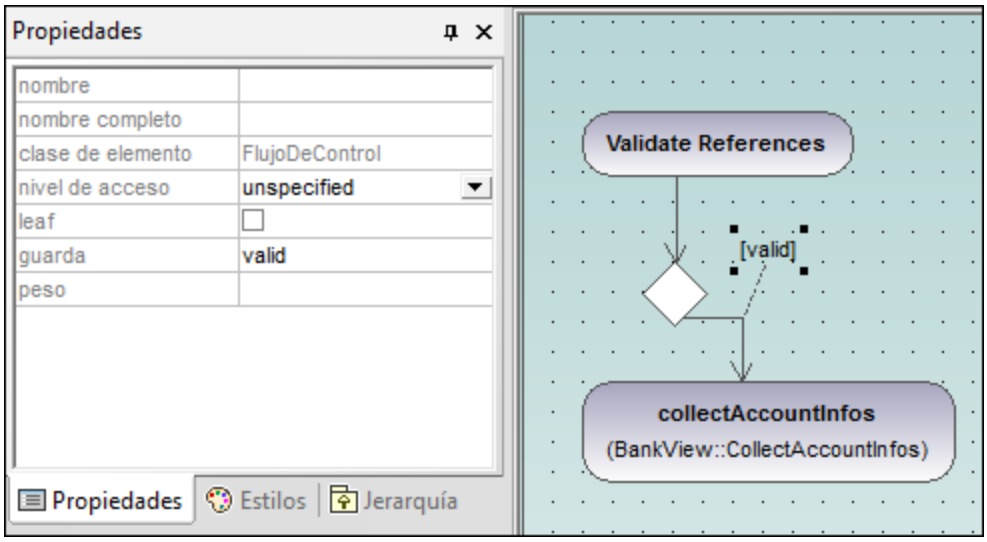


El elemento se resalta cuando sea posible colocar el conector.

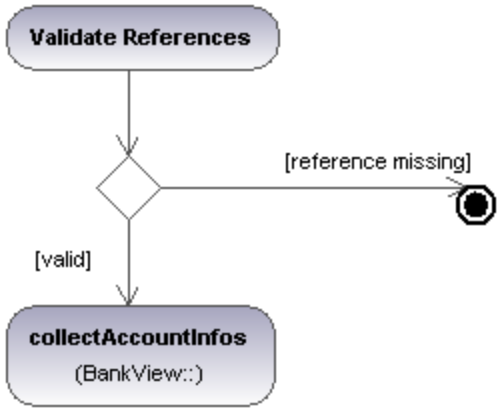
4. Haga clic en el elemento NodoDeDecisión y después en su conector derecho (el controlador FlujoDeControl). Arrástrelo hasta la acción collectAccountInfos. Consulte el apartado Insertar una acción (OperaciónDeLlamada) para obtener más información.



5. En la ventana *Propiedades* seleccione el valor valid para la propiedad guarda.



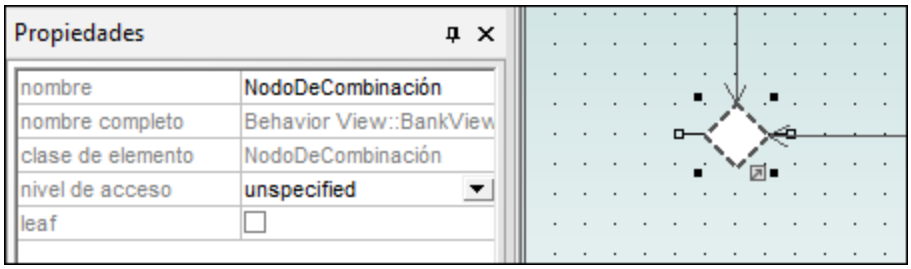
6. Haga clic en el elemento **`NodoDeDecisión`** y después en su conector derecho (el controlador `FlujoDeControl`). Arrástrelo hasta el elemento **`NodoFinalDeActividad`**.
La condición de guarda de esta transición se define automáticamente como "else". Haga doble clic en la condición de guarda del diagrama para cambiarla por "reference missing".



Nota: recuerde que UModel no valida ni revisa el número de flujos de control/objetos del diagrama.

Para crear una combinación:

1. Haga clic en el icono **NodoDeCombinación**  de la barra de herramientas y después haga clic en el diagrama para insertarlo.



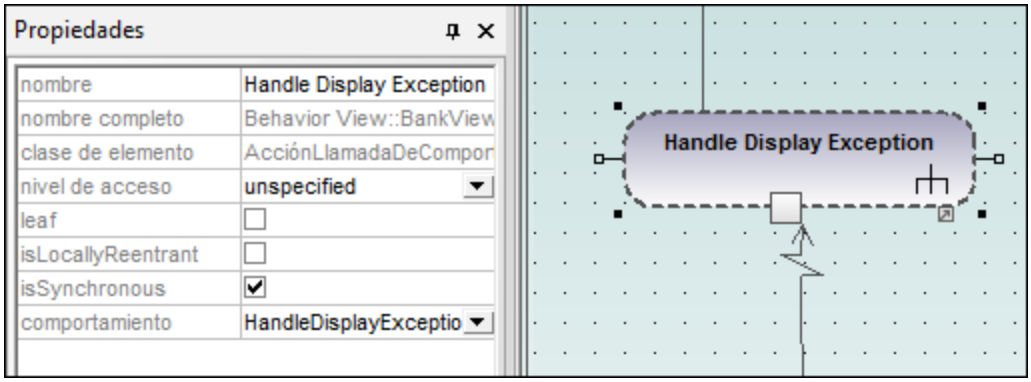
2. Haga clic en el conector FlujoDeControl (FlujoDeObjetos) de las acciones que desea combinar y arrástrelas hasta el símbolo del NodoDeCombinación.

9.1.1.3 Elementos

 Acción (ComportamientoDeLlamada)

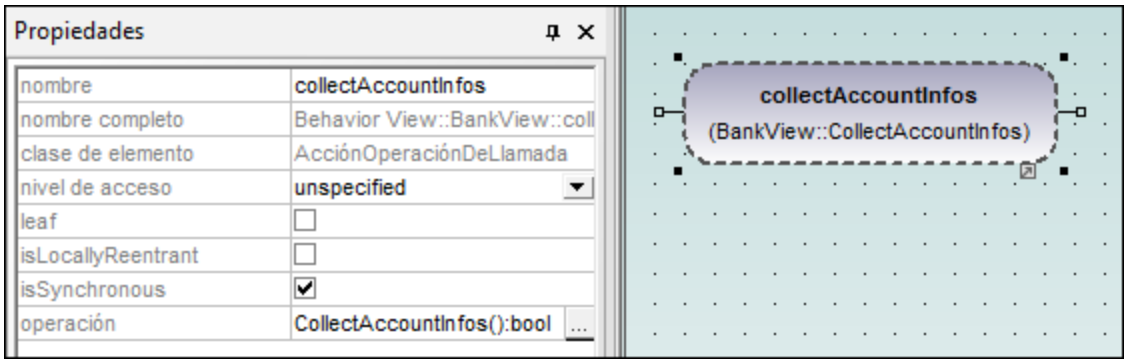
Inserte el elemento **`AcciónComportamientoDeLlamada`**, que invoca un comportamiento concreto directamente.

Si selecciona un comportamiento que ya existe desde el cuadro combinado comportamiento (de la ventana *Propiedades*), en el elemento aparece un icono en forma de rastrillo.



 Acción (OperaciónDeLlamada)

Inserta el elemento **`AcciónOperaciónDeLlamada`**, que invoca un comportamiento concreto como método directamente. Para más información consulte el apartado Insertar una acción (OperaciónDeLlamada).



Acción (AcciónOpaca)

Un tipo de acción utilizada para especificar la información de implementación. Se puede usar como marcador de posición hasta que decida qué tipo de acción desea utilizar.

Acción (AcciónEspecificaciónDeValor)

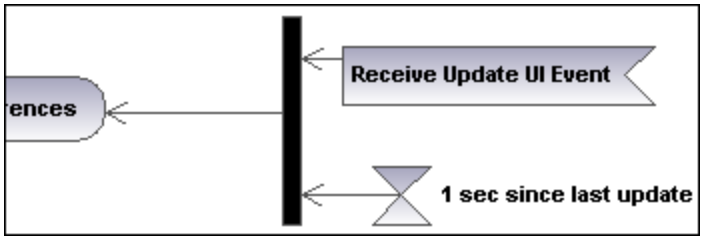
Un tipo de acción que evalúa (o genera) un valor determinado en el pin de salida. Viene definido por las propiedades (p. ej. upperBound para el límite superior.)

AcciónAceptarEvento

Inserta la acción **`AceptarEvento`**, que espera que tenga lugar un evento que cumpla determinados requisitos.

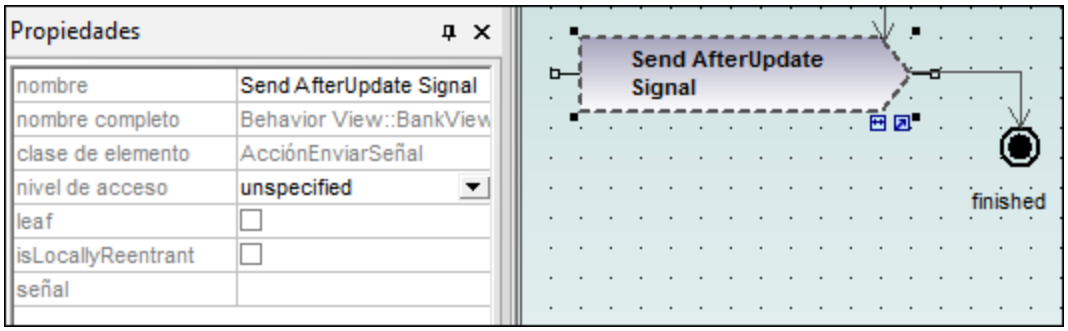
AcciónAceptarEvento (EventoDeTiempo)

Inserta una acción **`AceptarEvento`**, que está desencadenada por un evento de tiempo (un instante de tiempo que viene dado por una expresión).



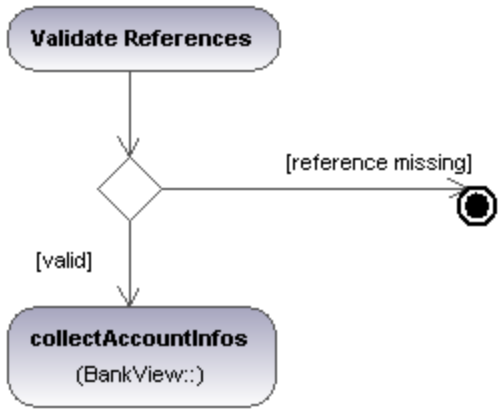
AcciónEnviarSeñal

Inserta la acción **`EnviarSeñal`**, que crea una señal desde sus entradas y transmite la señal al objeto de destino, donde puede provocar la ejecución de una actividad.



NodoDeDecisión

Inserta un **NodoDeDecisión**, que tiene una sola transición de entrada y varias transiciones de salida protegidas por guardas. Para más información consulte el apartado Crear ramas.



NodoDeCombinación

Inserta un **NodoDeCombinación**, que combina varias transiciones alternas definidas por el **NodoDeDecisión**. El **NodoDeCombinación** no sincroniza procesos simultáneos, sino que selecciona uno de los procesos.

NodoInicial

Se trata del comienzo del proceso de actividades. Una actividad puede tener varios nodos iniciales.

NodoFinalDeActividad

Se trata del final del proceso de actividades. Una actividad puede tener varios nodos finales y todos los flujos de la actividad se detienen cuando se encuentra el primer nodo final.

NodoFinalDeFlujo

Inserta un **NodoFinalDeFlujo**, que termina un flujo pero no los demás flujos de la actividad.

NodoDeBifurcación

Inserta un nodo de bifurcación vertical. Sirve para dividir los flujos en varios flujos simultáneos.

NodoDeBifurcación (Horizontal)

Inserta un nodo de bifurcación horizontal. Sirve para dividir flujos en varios flujos simultáneos.

NodoDeReunión

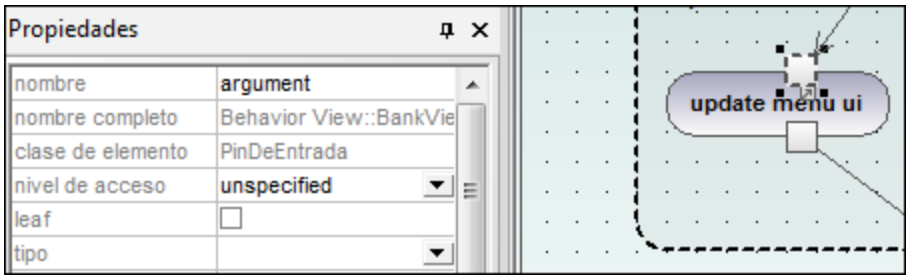
Inserta un nodo de reunión vertical. Sirve para sincronizar varios flujos definidos por un nodo de bifurcación.

NodoDeReunión (horizontal)

Inserta un nodo de reunión horizontal. Sirve para sincronizar varios flujos definidos por un nodo de bifurcación.

PinDeEntrada

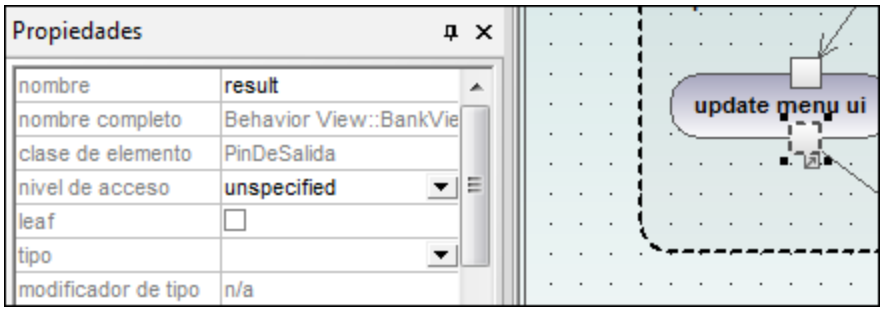
Inserta un pin de entrada en un **`ComportamientoDeLlamada`** o en una **`OperaciónDeLlamada`**. Los pins de entrada aportan los valores de entrada que utiliza la acción. A los pins de entrada se les asigna un nombre predeterminado (argumento) de forma automática.



Si al arrastrar el pin de entrada sobre un elemento el puntero se convierte en este símbolo  , significa que puede colocar el pin de entrada en el elemento.

PinDeSalida

Inserta un pin de salida. Los pins de salida contienen los valores de salida que produce una acción. Al pin de salida se le asigna automáticamente un nombre equivalente a la propiedad UML de la acción (p. ej. result).



Si al arrastrar el pin de salida sobre un elemento el puntero se convierte en este símbolo , significa que puede colocar el pin de salida en el elemento.

Pin de Excepción

Puede convertir un **`PinDeSalida`** en un pin de **`Excepción`** haciendo clic en el pin y seleccionando esPinDeExcepción en la ventana *Propiedades*.

PinDeValor

Inserta un pin de valor que es un pin de entrada que aporta un valor a una acción que no viene de un flujo de objeto de entrada. Se representa con el símbolo de un pin de entrada y tiene las mismas propiedades que un pin de entrada.

NodoDeObjeto

Inserta un nodo de objeto que es un nodo de actividad abstracto que define el flujo de objeto de una actividad. Los nodos de objeto solo contiene valores en tiempo de ejecución que se ajustan al tipo del nodo de objeto.

NodoDeBúferCentral

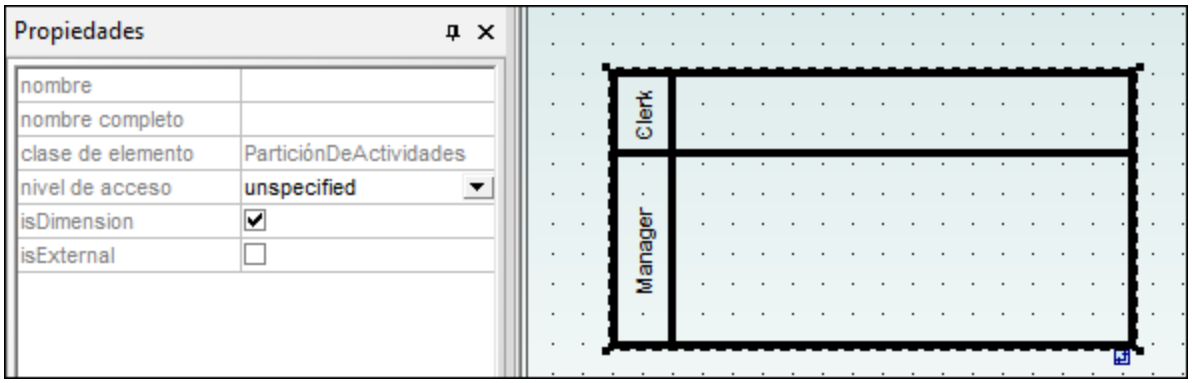
Inserta un nodo de búfer central que funciona de búfer para varios flujos de entrada y salida de otros nodos de objeto.

NodoAlmacénDeDatos

Inserta un nodo de almacén de datos, un nodo de búfer central especial que almacena datos persistentes (es decir, no transitorios).

ParticiónDeActividades (horizontal)

Inserta una partición de actividades horizontal, un tipo de grupo de actividades que sirve para identificar acciones que tienen características en común. Suelen equivaler a las unidades de organización de los modelos de negocio.



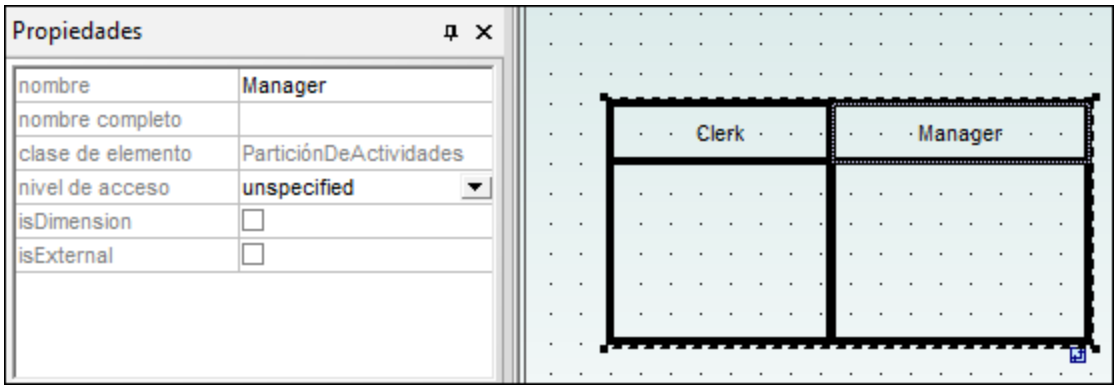
Para editar una etiqueta, haga doble clic en ella. Para orientar el texto correctamente, pulse **Entrar**.

Recuerde que las particiones de actividades de UML 2.0 son el equivalente de los *compartimentos* (swimlanes) de las versiones antiguas de UML.

- Los elementos colocados dentro de una **`ParticiónDeActividades`** pasan a formar parte de ella cuando su contorno esté resaltado.
- Los objetos colocados dentro de una **`ParticiónDeActividades`** se pueden seleccionar uno por uno con **Ctrl+clic** o con el recuadro de selección.
- Para mover la **`ParticiónDeActividades`** a otra posición, haga clic en su contorno o en su título y arrástrela.

ParticiónDeActividades (vertical)

Inserta una partición de actividades vertical, un tipo de grupo de actividades que sirve para identificar acciones que tienen características en común. Suelen equivaler a las unidades de organización de los modelos de negocio.

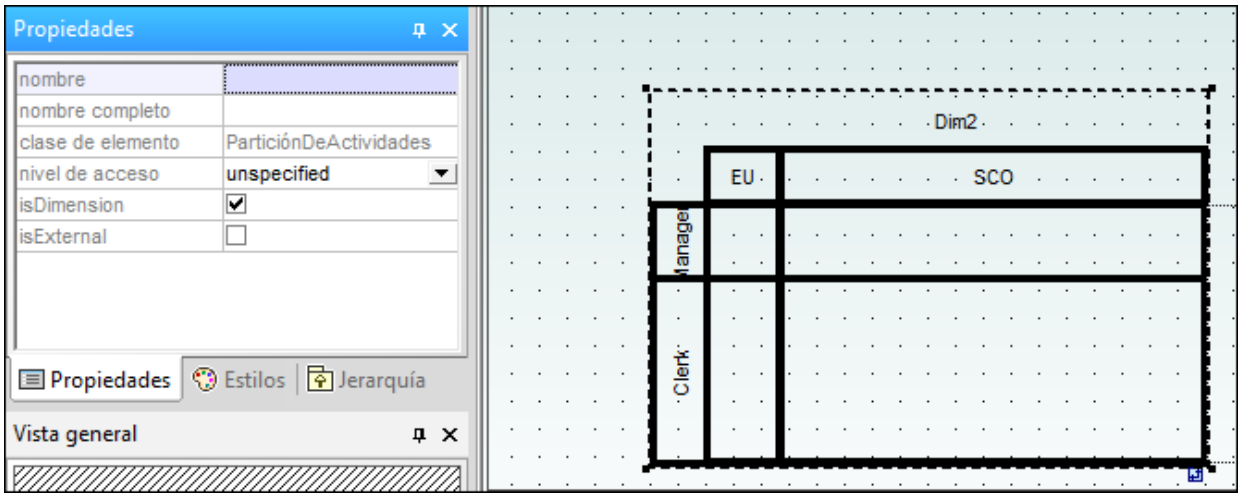


ParticiónDeActividades (2D)

Inserta una partición de actividades bidimensional, un tipo de grupo de actividades que sirve para identificar acciones que tienen características en común. Las etiquetas de los dos ejes son editables.

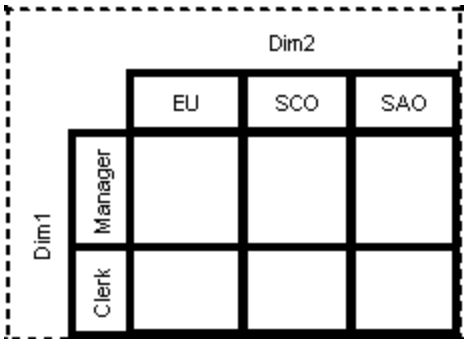
Para quitar las etiquetas de las dimensiones Dim1 y Dim2:

1. Haga clic en la etiqueta que desea eliminar (p. ej. Dim1).
2. En la ventana *Propiedades* haga doble clic en la entrada Dim1, elimínela y pulse **Entrar** para confirmar.



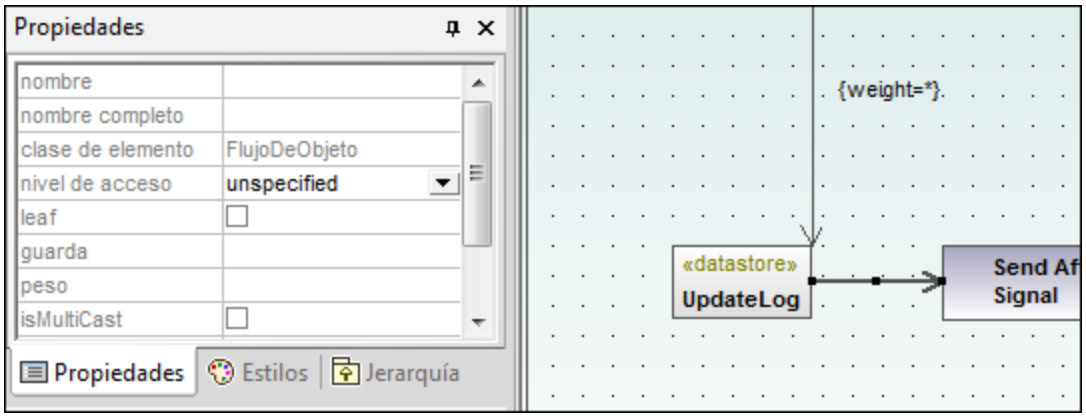
Para anidar particiones de actividades:

1. Haga clic con el botón derecho en la etiqueta de la dimensión donde desea insertar un partición nueva.
2. Seleccione **Nuevo/a | ParticiónDeActividades**.



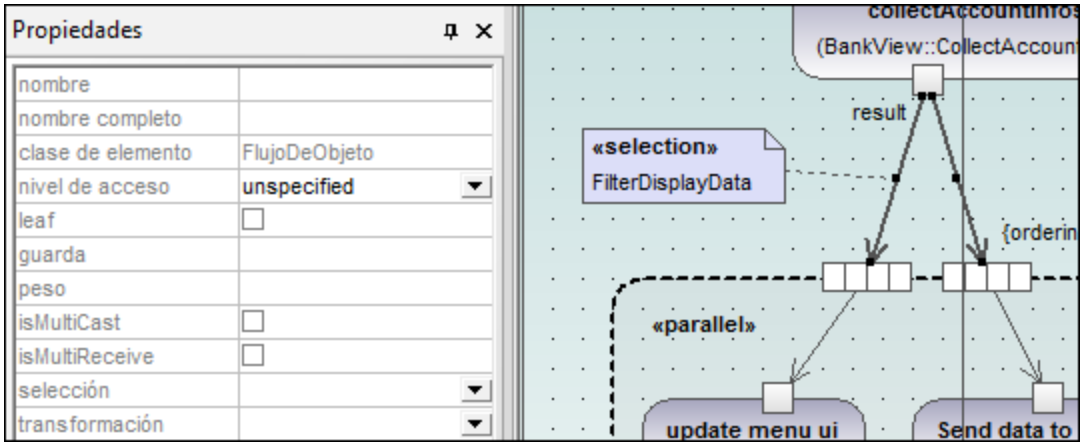
FlujoDeControl

Un flujo de control es una línea con una flecha que conecta dos actividades/comportamientos e inicia una actividad una vez finaliza la actividad anterior.



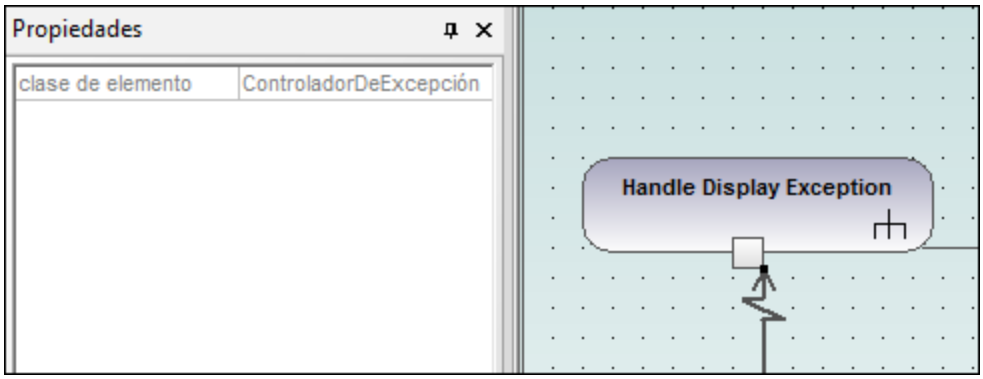
FlujoDeObjeto

Un flujo de objeto es una línea con una flecha que conecta dos acciones/nodos de objeto e inicia una actividad una vez finaliza la actividad anterior. Los objetos y datos se pueden pasar a través del flujo de objeto.



ControladorDeExcepción

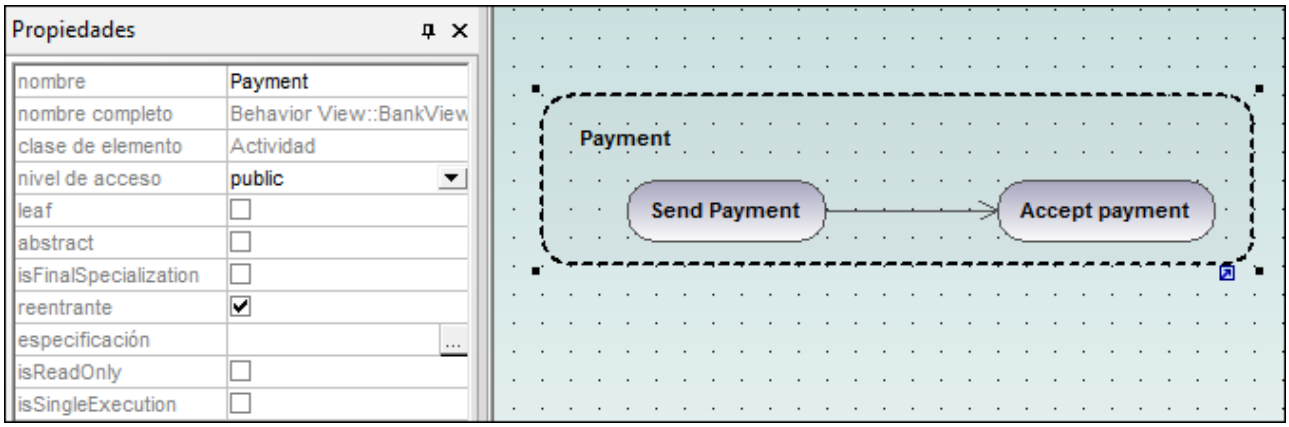
Un controlador de excepción es un elemento que especifica qué acción debe ejecutarse si se genera determinada excepción durante la ejecución del nodo protegido.



Los controladores de excepción solo se pueden colocar en el pin de entrada de una acción.

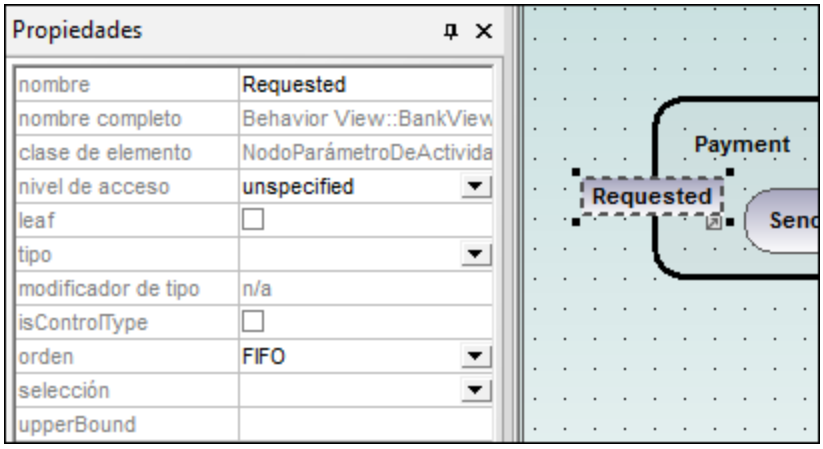
Actividad

Inserta una actividad en el diagrama de actividades.



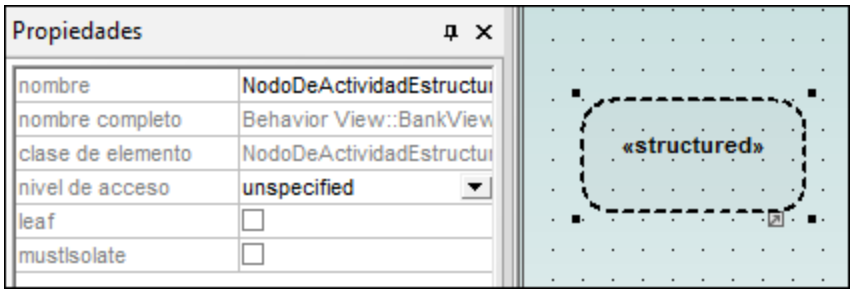
NodoParámetroDeActividad

Inserta un nodo parámetro de actividad en una actividad. Al hacer clic en la actividad se inserta el nodo parámetro en el contorno de la actividad.



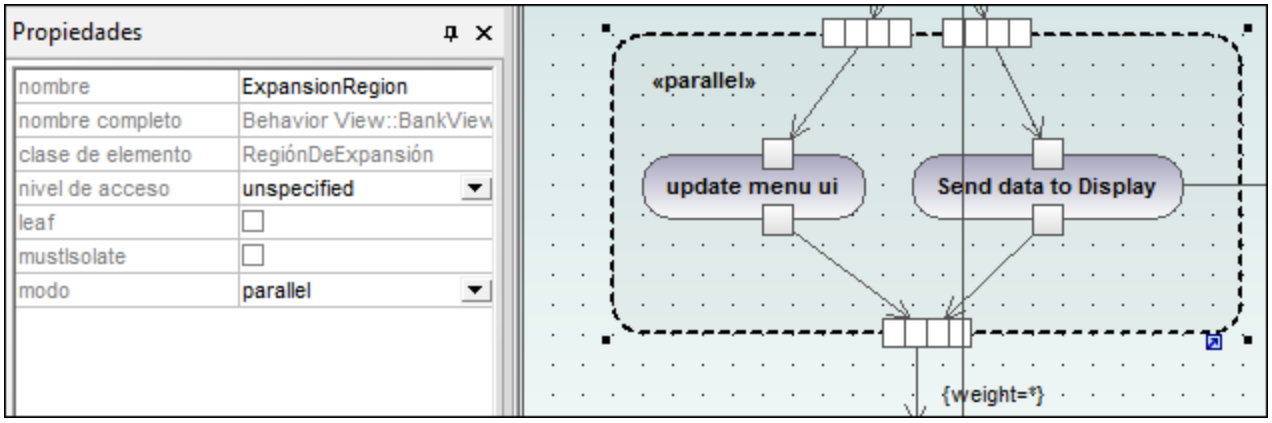
NodoDeActividadEstructurada

Inserta un nodo de actividad estructurada, que es una parte estructurada de la actividad que no se comparte con ningún otro nodo estructurado.



RegiónDeExpansión

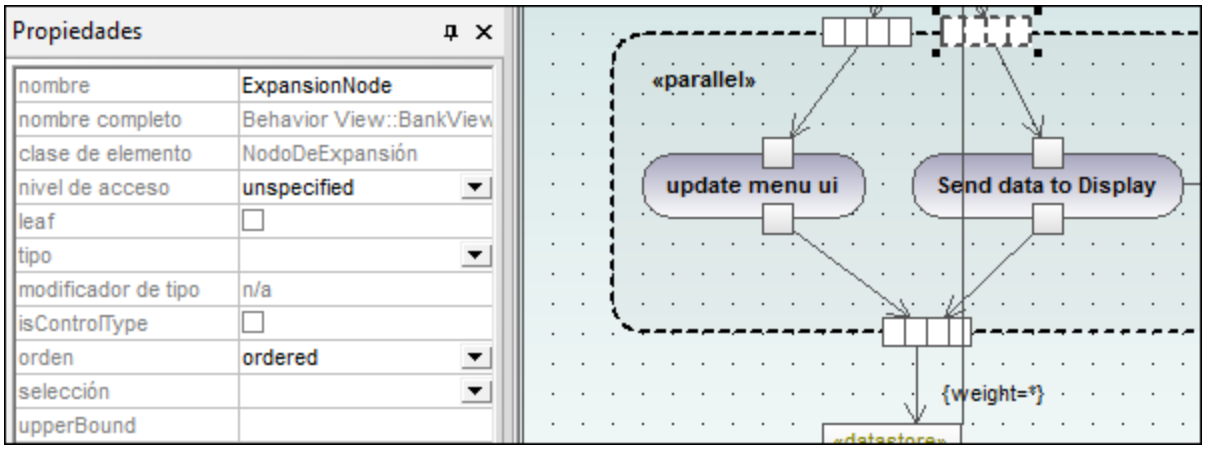
Una región de expansión es una región de una actividad que tiene entradas y salidas explícitas (usando **`NodosDeExpansión`**). Cada entrada es una colección de valores.



El modo región de expansión aparece como palabra clave y para cambiarlo basta con hacer clic en el cuadro combinado modo de la ventana *Propiedades*. Las opciones disponibles son: parallel, iterative o stream.

NodoDeExpansión

Inserta un nodo de expansión en una región de expansión. Los nodos de expansión son nodos de entrada y salida para la región de expansión, donde cada entrada/salida es una colección de valores. Las flechas que entran y salen de la región de expansión determinan el tipo concreto de nodo de expansión.

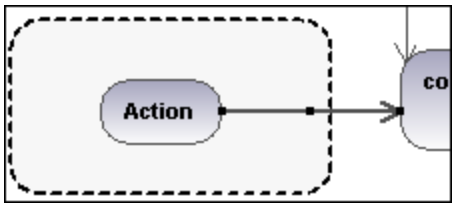


RegiónDeActividadInterrumpible

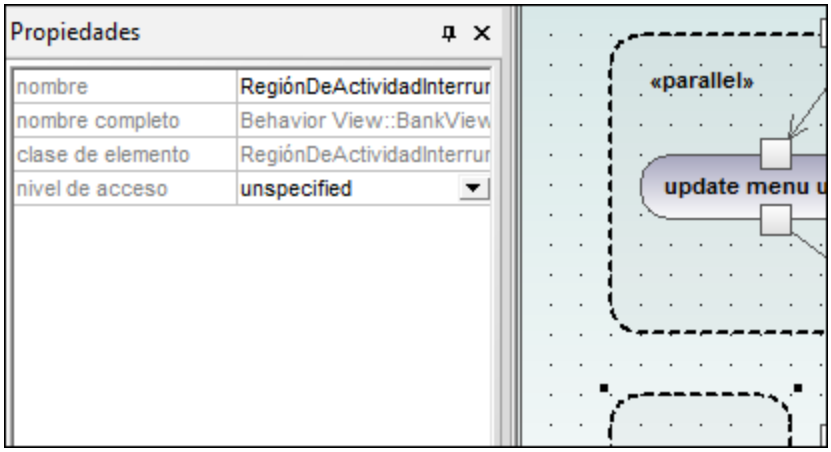
Una región interrumpible contiene nodos de actividad. Cuando un flujo de control abandona una región interrumpible, todos los flujos y comportamientos de la región finalizan.

Para añadir un encadenamiento interruptor:

Primero debe comprobar que hay un elemento **`Acción`** en la **`RegiónDeActividadInterrumpible`**, así como un flujo de control de salida hacia otra acción:



1. Haga clic con el botón derecho en la flecha del **`FlujoDeControl`** y seleccione **Nuevo/a | EncadenamientoInterruptor**.



Nota: hay otra manera de añadir un **`EncadenamientoInterruptor`**: haga clic en la **`RegiónDeActividadInterrumpible`**, después haga clic con el botón derecho en la ventana *Propiedades* y seleccione **Agregar encadenamientoInterruptor** en el menú emergente.

9.1.2 Diagrama de máquina de estados

Los diagramas de máquina de estados modelan el comportamiento de un sistema, describiendo los diferentes estados por los que puede pasar un objeto y las transiciones de unos estados a otros. Se suelen utilizar para describir el comportamiento de un objeto que pasa por varios casos de uso.

Esto se puede conseguir con dos tipos de procesos:

1. **Acciones:** están asociadas a las **transiciones** y son procesos a corto plazo que no se pueden interrumpir (por ejemplo, en la imagen siguiente: una transición inicial **`error interno / notificar admin`**).
2. **Actividades de estado (comportamientos):** están asociadas a los estados y son procesos a largo plazo que pueden ser interrumpidos por otros eventos (por ejemplo, en la imagen siguiente: **`escuchar si hay conexiones entrantes`**).

En UModel una máquina de estados puede tener varios diagramas de máquina de estados (o diagramas de estados).

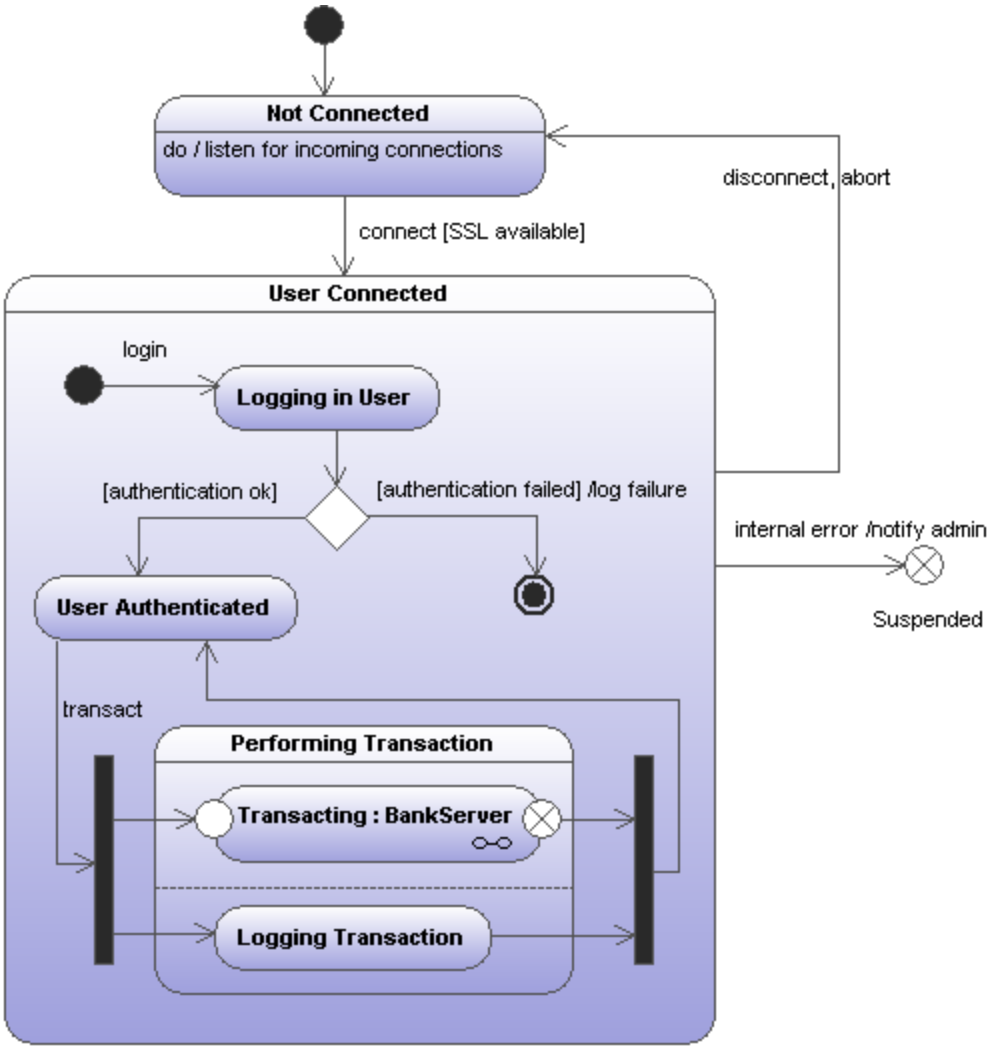


Diagrama de máquina de estados de ejemplo

El diagrama de estados anterior se encuentra en la carpeta del proyecto de ejemplo de UModel **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Bank_MultiLanguage.ump**.

9.1.2.1 Insertar elementos

Para insertar elementos con los iconos de la barra de herramientas:

1. Haga clic en el icono de máquina de estados en la barra de herramientas *Diagrama de máquina de estados*.

2. Haga clic en el área de trabajo del diagrama de estados para insertar el elemento. Si quiere insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos que ya existen hasta el diagrama

1. Busque en la *Estructura del modelo* el elemento que quiere insertar en la pestaña *Árbol de diagramas* (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de máquina de estados.

9.1.2.2 Crear estados, actividades y transiciones

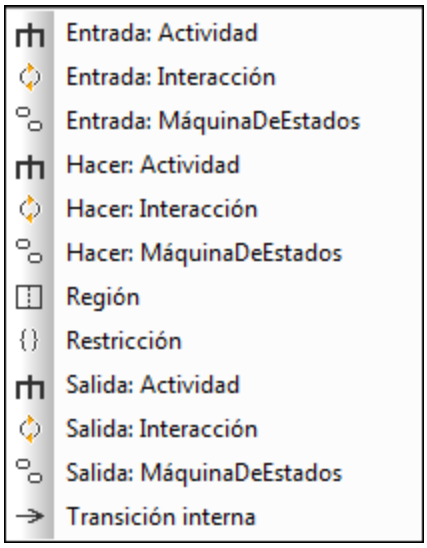
Para agregar un estado simple:

1. Haga clic en el icono **Estado** de la barra de herramientas () y después haga clic dentro del diagrama.
2. Escriba el nombre del estado y pulse **Entrar** para confirmar.

Para añadir una actividad al estado:

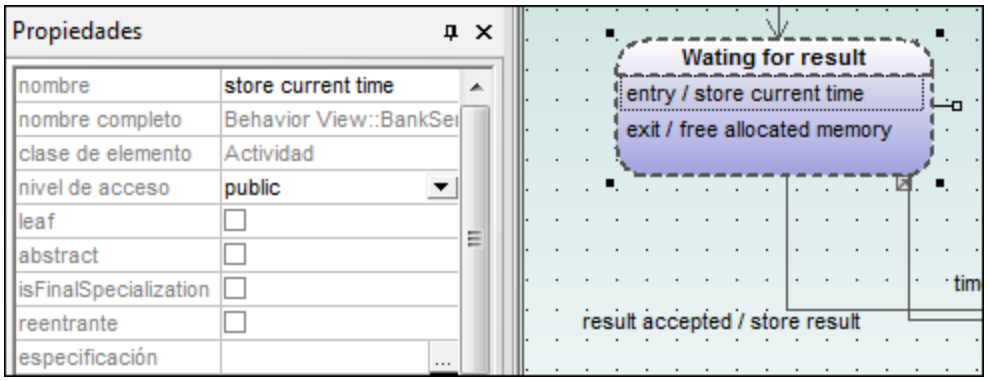
- Haga clic con el botón derecho en el estado. En el menú contextual seleccione **Nuevo/a** y después una de las opciones del submenú.

Las actividades Entrada, Salida y Hacer están asociadas con uno de estos comportamientos posibles: "Actividad", "Interacción" y "MáquinaDeEstados". Por tanto, las opciones de este menú contextual son:



Estas opciones proceden de la especificación UML. Es decir, todas estas acciones internas son comportamientos y, en la especificación UML, se derivan tres clases de la clase "Comportamiento": Actividad, MáquinaDeEstados e Interacción. En el código generado no importa qué comportamiento se seleccione.

Hay tres tipos de acciones: **Hacer** (Do), **Entrada** (Entry) y **Salida** (Exit). Las actividades se colocan dentro de su propio compartimento en el estado (no en una región distinta). El tipo de actividad seleccionada se utiliza como prefijo para la actividad (p. ej. entry / store current time).

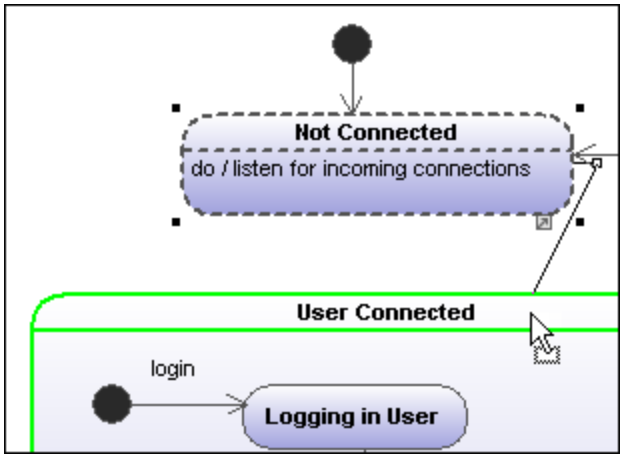


Para eliminar una actividad:

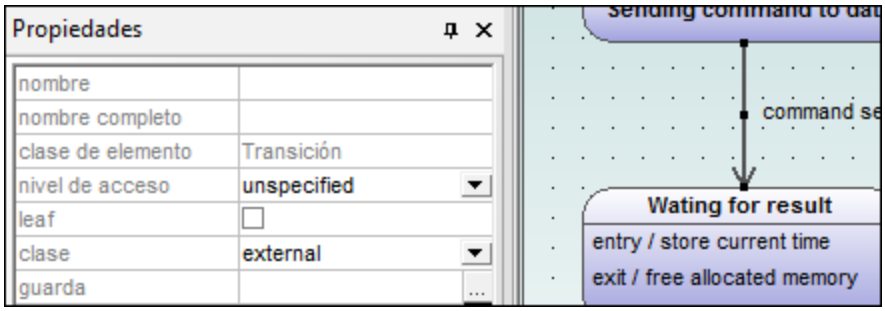
- Haga clic en la actividad del estado y pulse la tecla **Supr**.

Para crear una transición entre dos estados:

1. Haga clic en el controlador Transición del estado de origen (situado a la derecha del elemento).
2. Arrastre la flecha de la transición hasta el estado de destino.



Las propiedades de la transición se pueden ver en la ventana *Propiedades*. En el cuadro combinado del campo clase puede definir el tipo de transición: externa, interna o local.



Las transiciones pueden tener un disparador de eventos, una condición de protección y una acción con el formato **disparadorEvento [condición de protección] /actividad**.

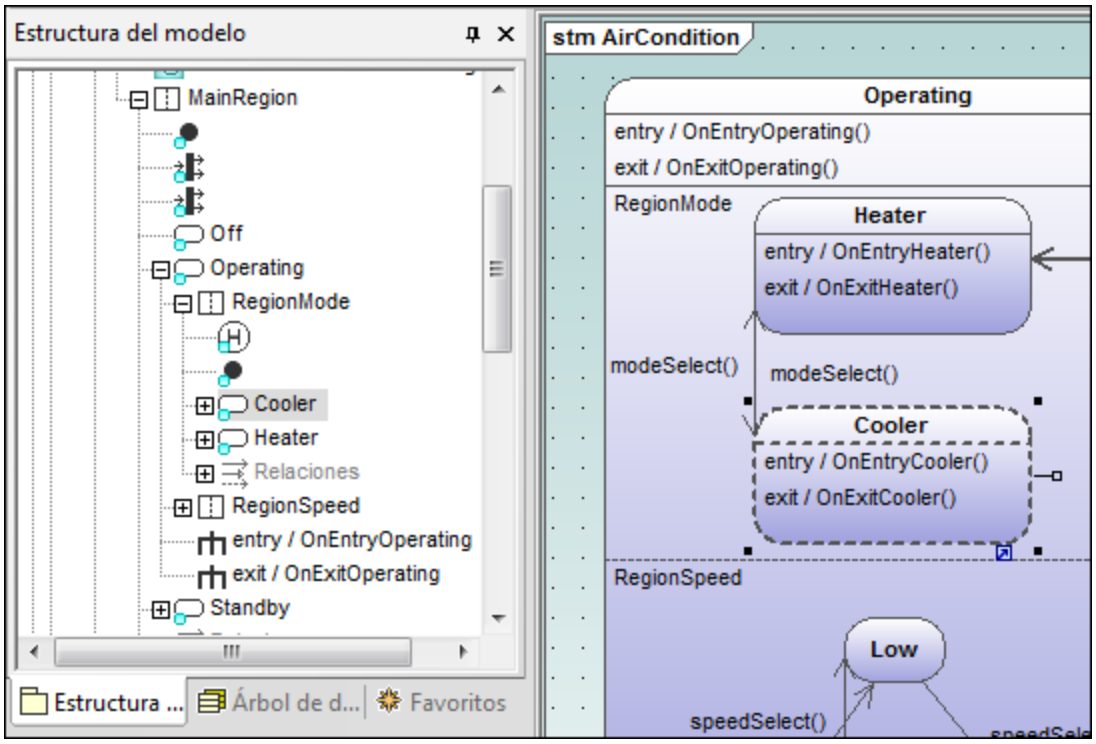
Para crear operaciones desde transiciones automáticamente:

Si activa el icono **Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación**  , operación correspondiente se crea automáticamente en la clase referenciada cuando se crea una transición y se escribe un nombre (p. ej. miOperación()).

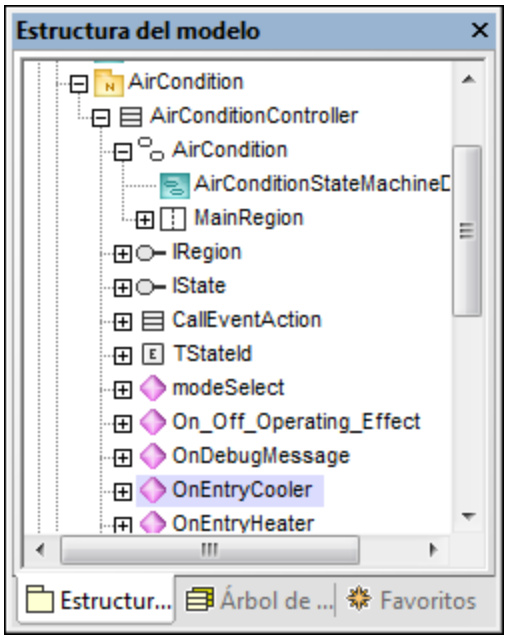
Nota: solamente se pueden crear operaciones automáticamente cuando la máquina de estados está dentro de una clase o de una interfaz.

Para crear operaciones automáticamente desde actividades:

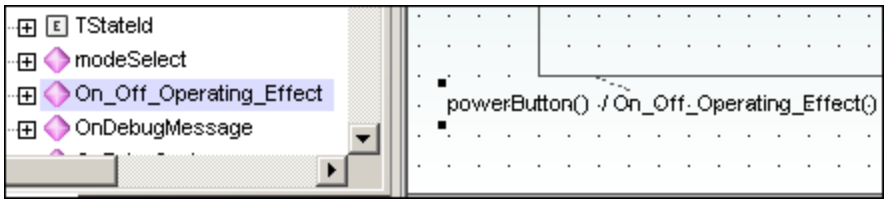
1. Haga clic con el botón derecho en el estado y seleccione la actividad/acción que desea insertar (**Nuevo/a | Entrada:Actividad**).
2. Escriba el nombre de la actividad, asegurándose de que termina con ().



El elemento nuevo también está disponible en la Estructura del modelo.
Desplácese hacia abajo en la *Estructura del modelo* y observe que la operación OnEntryCooler se añadió a la clase primaria AirConditionController.

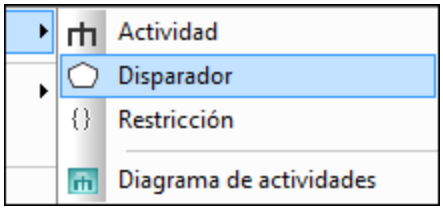


Nota: se añaden operaciones automáticamente para: **Hacer:Actividad, Entrada:Actividad, Salida:Actividad**.



Para crear un disparador de transición:

1. Haga clic con el botón derecho en una transición (en la flecha).
2. Selección **Nuevo/a | Disparador**.

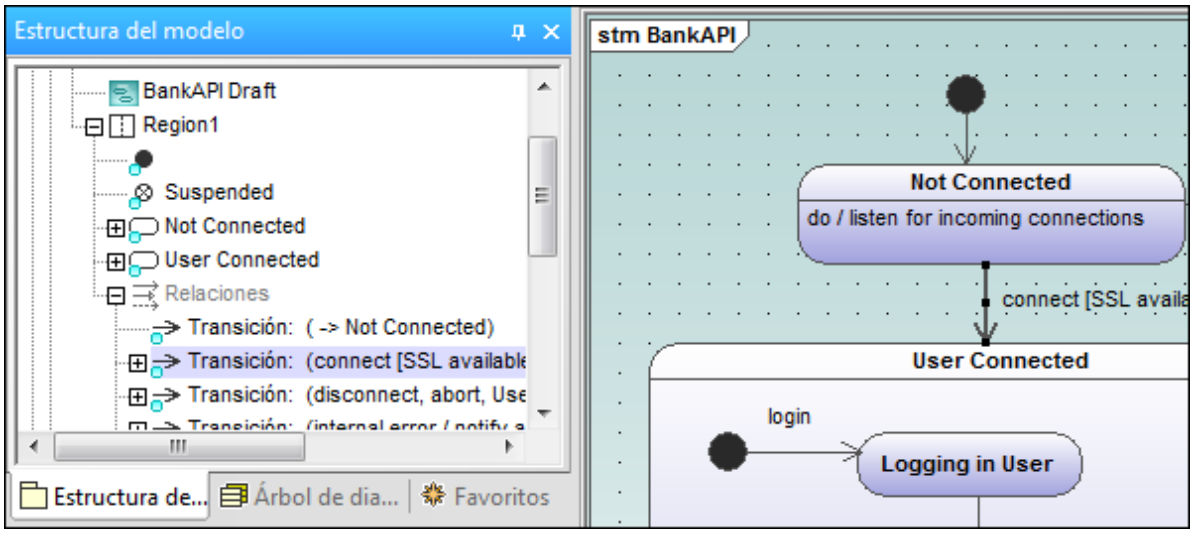


Si se trata del primer disparador del diagrama, en la etiqueta de la transición, situada encima de la flecha, aparece el carácter "a". Los disparadores tienen asignados valores predeterminados en forma de letra, estado de origen -> estado de destino.

3. Haga doble clic en el nuevo carácter e inserte las propiedades de la transición en el formato **disparadorEvento [condición de protección] / actividad**.

Sintaxis de las propiedades de la transición
el texto insertado antes de los corchetes es el disparador. Entre los corchetes va la condición de protección y después de la barra diagonal va la actividad. Manipule esta cadena para crear o eliminar automáticamente los elementos correspondientes en la *Estructura del modelo*.

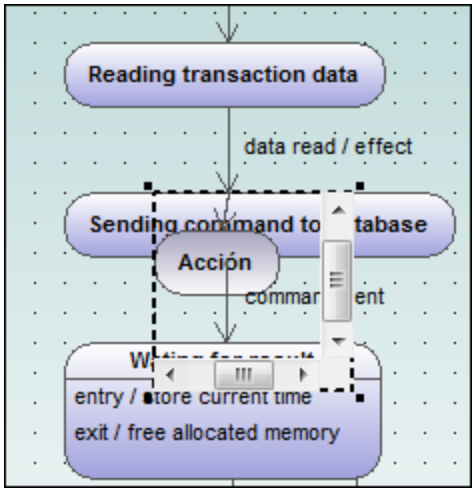
Nota: para ver las propiedades de una transición, haga clic con el botón derecho en la transición y seleccione **Seleccionar en la Estructura del modelo**. El evento, la actividad y los elementos de restricción aparece debajo de la transición seleccionada.



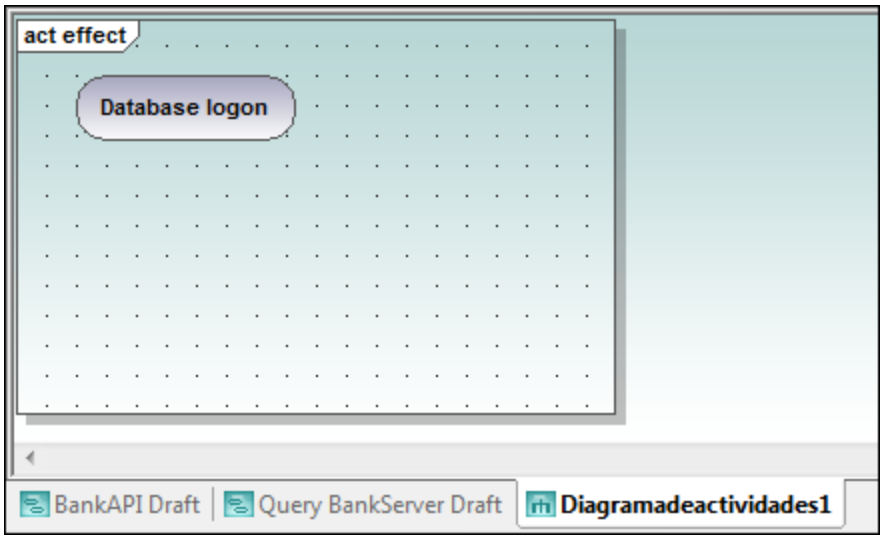
Agregar un diagrama de actividades a una transición

UModel ofrece una función única para añadir diagramas de actividades a las transiciones a fin de describir la transición en detalle.

1. Haga clic con el botón derecho en la transición y seleccione **Nuevo/a | Diagrama de actividades**. Esto inserta una ventana con un diagrama de actividades en la posición de la flecha de transición.
2. Haga clic en la ventana recién insertada y utilice las barras de desplazamiento para desplazarse por la ventana.

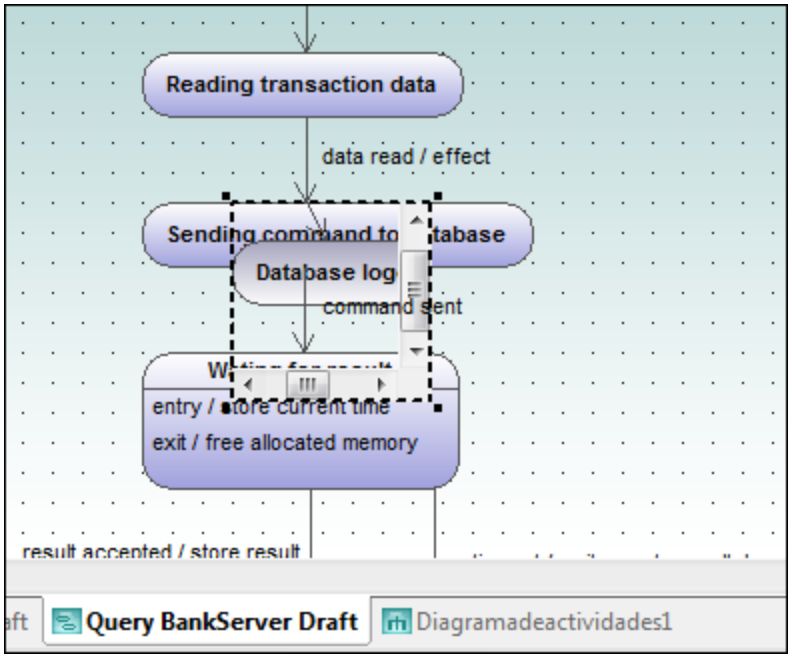


3. Haga doble clic en la ventana para abrir el diagrama de actividades en otra pestaña y seguir definiendo la transición (p. ej. cambiando el nombre de la acción a `Database logon`).

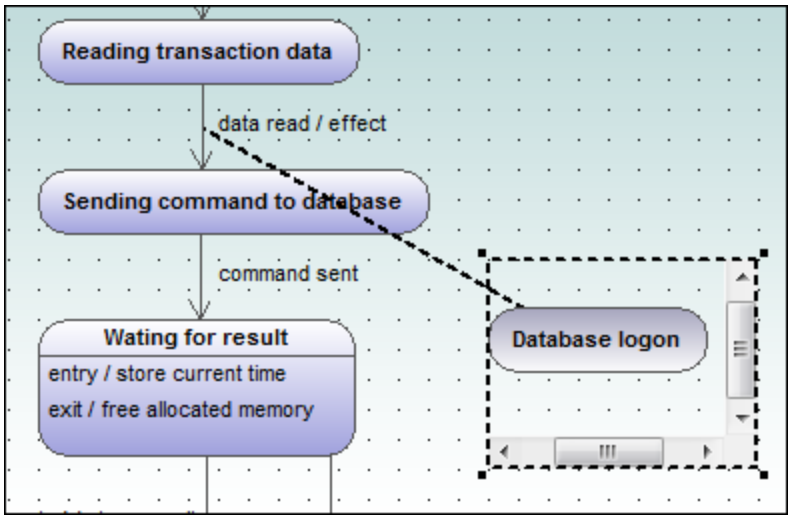


Al proyecto se añade un diagrama de actividades nuevo. Para aprender a añadir elementos de modelado de actividades nuevos al diagrama, consulte el apartado Diagrama de actividades.

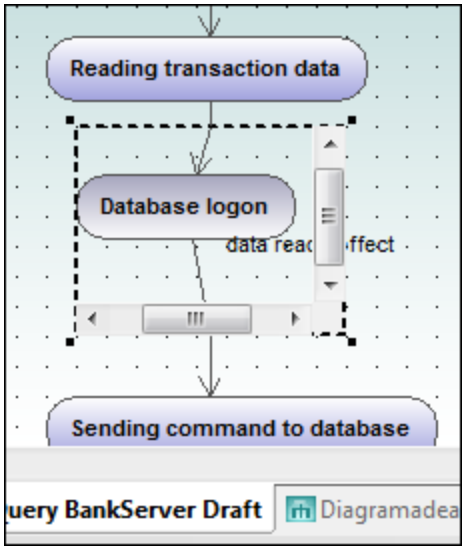
4. Haga clic en la pestaña del diagrama de máquina de estados para ver la transición actualizada.



5. Arrastre la ventana de la actividad a una posición nueva donde no moleste y ajuste el tamaño de la ventana si es necesario.



Si arrastra la ventana de la actividad y la pone entre los dos estados, la ventana ilustra la transición hacia y desde la actividad.



9.1.2.3 Estados compuestos

Estado compuesto

Este tipo de estado contiene un segundo compartimento, compuesto por una única región. Dentro de esta región puede colocar un número ilimitado de estados.

Para añadir una región a un estado compuesto:

- Haga clic con el botón derecho en el estado compuesto y seleccione **Nuevo/a | Región** en el menú contextual. Al estado se le añade una región nueva. Las regiones se dividen con líneas discontinuas.

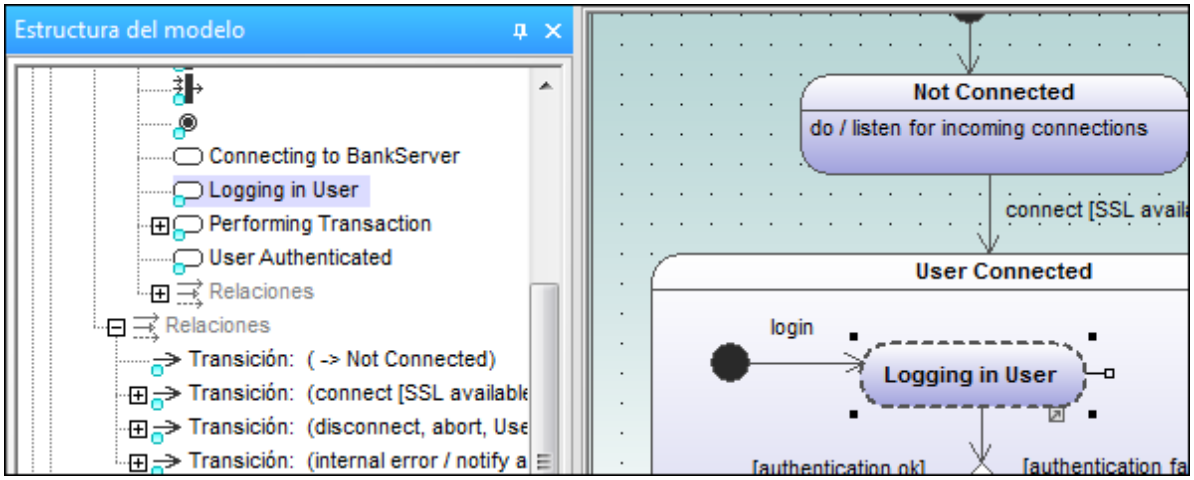
Para eliminar una región:

- Haga clic en la región que desea eliminar y pulse **Supr**. Cuando se elimina una región de un estado ortogonal, el estado vuelve a ser un estado compuesto. Cuando se elimina la última región de un estado compuesto, el estado pasa a ser un estado simple.

Para poner un estado dentro de un estado compuesto:

- Haga clic en el estado que desea insertar (p. ej. **Logging in User**) y arrástrelo hasta el compartimento de la región del estado compuesto.

  El compartimento de la región se resalta al soltar el elemento. El elemento insertado ahora forma parte de la región y aparece como elemento secundario de la región en la ventana *Estructura del modelo*.

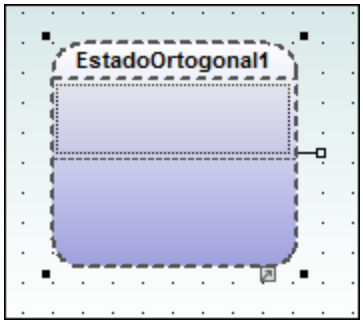


  Cuando se mueve el estado compuesto también se mueven los estados que están dentro de él.

Estado ortogonal

Este tipo de estado contiene un segundo compartimento que está compuesto por dos o más regiones que indican simultaneidad.

Haga clic con el botón derecho en un estado y seleccione **Nuevo/a | Región** para añadir regiones nuevas.



Para mostrar/ocultar el nombre de las regiones:

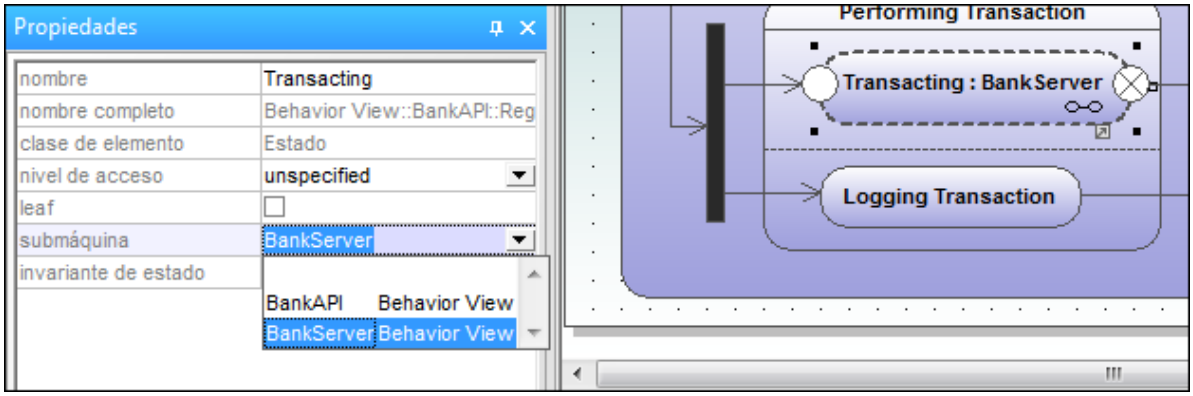
- Haga clic en la ventana *Estilos*, desplácese hasta el estilo Mostrar los nombres de región en los estados y seleccione el valor verdadero/falso.

Estado de submáquina

Este estado sirve para ocultar los detalles de una máquina de estados y no tiene regiones, sino que está asociado a una máquina de estados distinta.

Para definir un estado de submáquina:

1. Tras seleccionar un estado, haga clic en el cuadro combinado submáquina de la ventana *Propiedades*. Aparece una lista con todas las máquinas de estados que están definidas.
2. Seleccione la máquina de estados a la que debe hacer referencia esta submáquina.



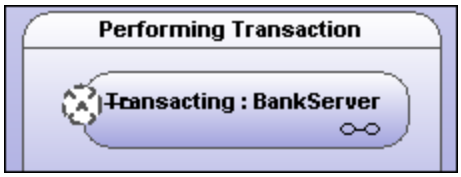
Observe que en la submáquina aparece automáticamente un icono de hipervínculo. Al hacer clic en este icono se abre la máquina de estados a la que se hace referencia (**`BankServer`**, por ejemplo).

Para añadir puntos de entrada/ salida a un estado de submáquina:

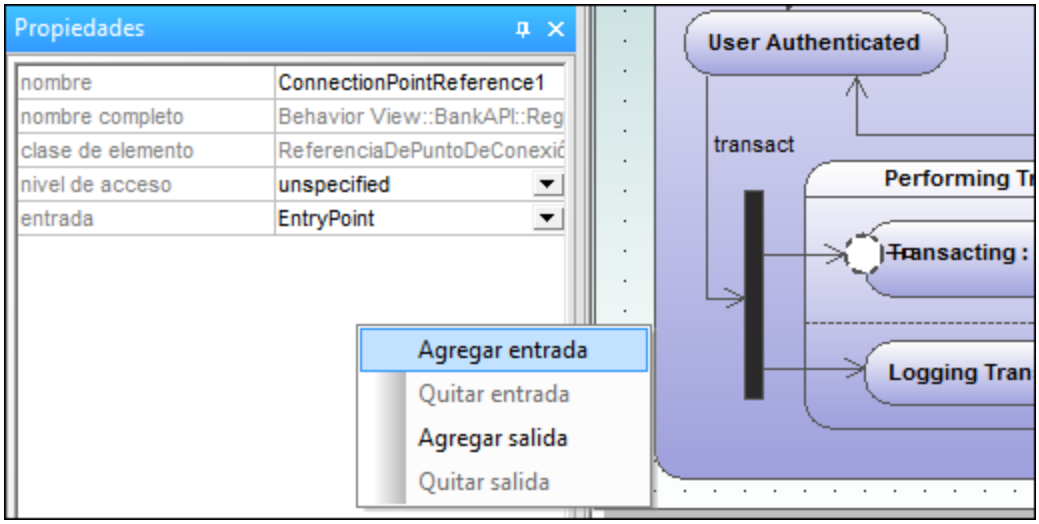
- El estado de submáquina al que está conectado el punto de entrada/salida debe hacer referencia a una máquina de estados (visible en la ventana *Propiedades*).

- Esta submáquina debe contener al menos un punto de entrada y uno de salida.

1. Haga clic en el icono **ReferenciaDePuntoDeConexión** de la barra de herramientas y después haga clic en el estado de submáquina en el que quiere insertar el punto de entrada/salida.



2. Haga clic con el botón derecho en la ventana *Propiedades* y seleccione **Agregar entrada**. Recuerde que este menú emergente solamente aparece si en el diagrama ya existe un punto de entrada o de salida.



El comando **Agregar entrada** añade un punto de entrada (EntryPoint) nuevo en la ventana *Propiedades* y cambia el aspecto de del elemento de referencia del punto de conexión ConnectionPointReference.

3. Use el mismo método para insertar un punto de salida (ExitPoint) con la opción **Agregar salida** del menú contextual.

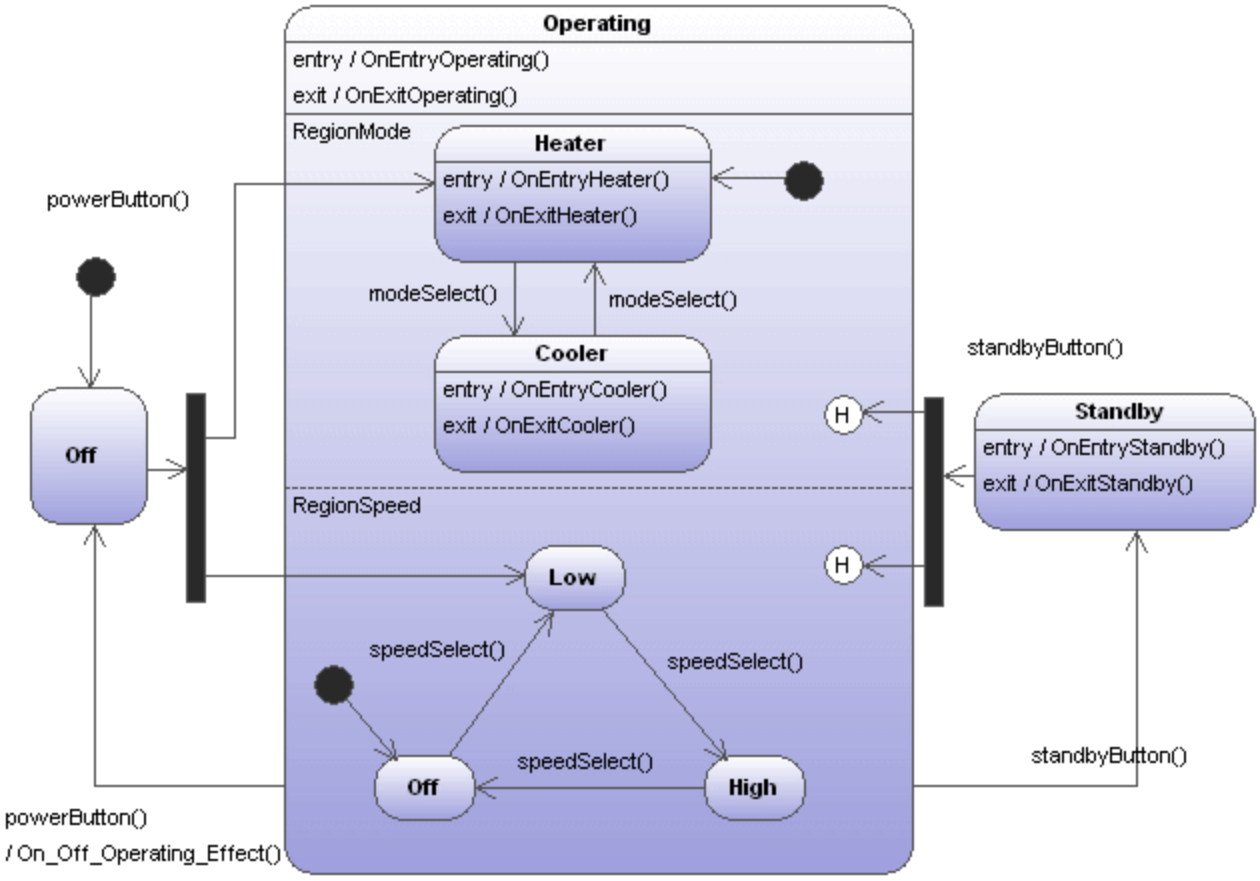
9.1.2.4 Generar código a partir de diagramas de máquina de estados

Con UModel puede generar código ejecutable a partir de diagramas de máquina de estados (C++, C#, Java, VB.NET). Esta función de generación de código es compatible con casi todos los elementos y las características de los diagramas de máquina de estados:

- Estado
- EstadoCompuesto, con cualquier nivel jerárquico
- EstadoOrtogonal, con cualquier número de regiones
- Región
- EstadoInicial
- EstadoFinal
- Transición

- Guarda
- Disparador
- Evento de llamada
- Bifurcación
- Reunión
- Elección
- Unión
- HistorialDetallado
- HistorialSuperficial
- Acciones de entrada/salida/hacer
- Efectos

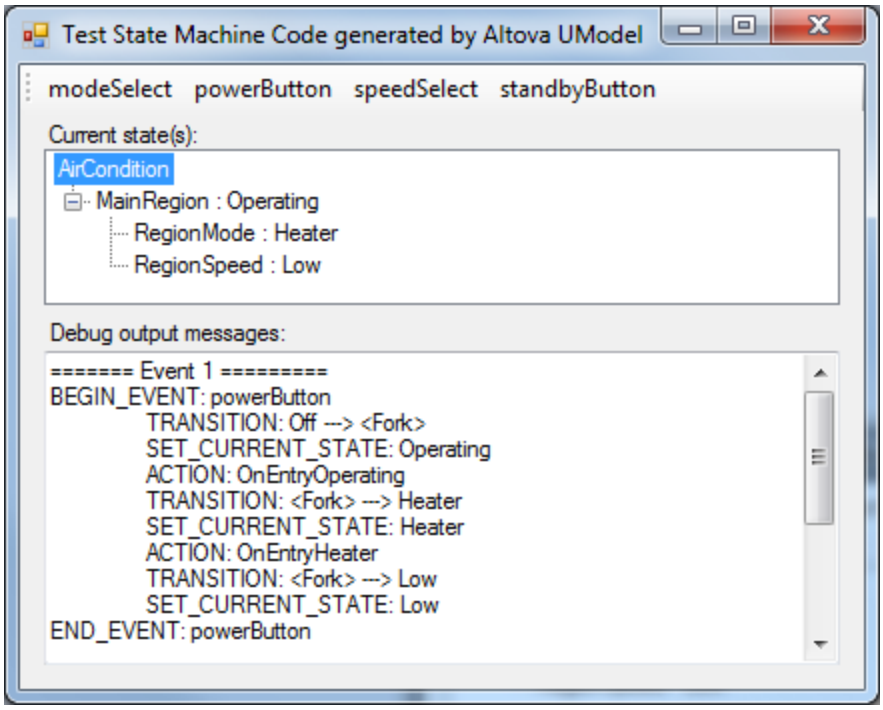
La generación de código de máquina de estados se integra en el proceso "normal" de ingeniería de ida y vuelta. Esto significa que el código de máquina de estados se puede actualizar automáticamente con cada proceso de ingeniería directa.



La imagen anterior muestra el diagrama de máquina de estados **`AirCondition`** de la carpeta .. \StateMachineCodeGeneration del directorio ...\UModelExamples. Hay una carpeta por cada lenguaje de programación compatible con UModel.

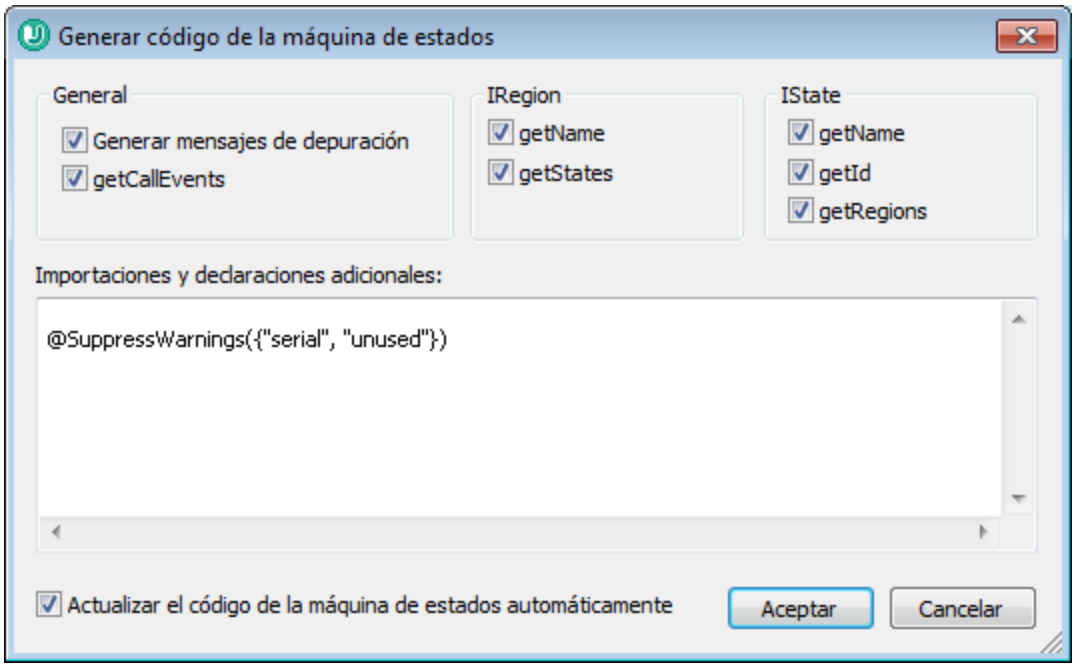
Cada directorio contiene dos carpetas: AirCondition y Complex. Cada una contiene el proyecto de UModel correspondiente, los archivos de proyecto del lenguaje de programación y los archivos de código generados. El archivo de proyecto Complex.ump contiene casi todos los elementos y funciones de modelado compatibles con la función de generación de código de UModel para diagramas de máquina de estados.

Además, cada carpeta contiene una aplicación de prueba (p. ej. TestSTMAirCondition.sln para C#) para que pueda trabajar inmediatamente con los archivos de código generados.



Para generar código a partir de un diagrama de máquina de estados:

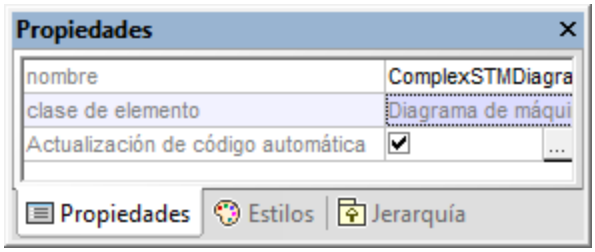
1. Haga clic con el botón derecho en el diagrama de máquina de estados y seleccione el comando **Generar código de la máquina de estados** o
2. Haga clic en **Proyecto | Generar código de la máquina de estados**.



Aparece un cuadro de diálogo (*imagen siguiente*). Si es necesario, ajuste las opciones de configuración predeterminadas y haga clic en **Aceptar** para generar el código.

El código de máquina de estados se actualiza automáticamente cuando se inicia el proceso de ingeniería directa. Sin embargo, esta configuración se puede cambiar. Para ello haga clic en el fondo del diagrama de máquina de estados y marque la casilla Actualización de código automática de la ventana *Propiedades*.

No es recomendable realizar cambios a mano en el código generado porque estos cambios no se traspasarán al diagrama de máquina de estados durante el proceso de ingeniería inversa.



En la ventana *Propiedades* haga clic en el icono **Examinar** ... del campo Actualización de código automática para abrir el cuadro de diálogo "Generar código de la máquina de estados" y cambiar las opciones de configuración.

Para revisar la sintaxis de un diagrama de máquina de estados:

- Haga clic con el botón derecho en el diagrama y seleccione **Revisar la sintaxis de la máquina de estados** en el menú contextual.

9.1.2.5 Trabajar con código de máquina de estados

La clase primaria de la máquina de estados (es decir, la clase controladora controller o la clase de contexto) es la única interfaz que existe entre el usuario de la máquina de estados y su implementación.

La clase controladora controller aporta los métodos que se pueden usar desde "fuera" para cambiar los estados (p. ej. después de que tengan lugar eventos externos).

No obstante, la implementación de la máquina de estados llama a los métodos de la clase `controller` (devoluciones de llamada) para informar al usuario de la máquina de estados sobre cambios de estado (`OnEntry`, `OnExit`, ...), efectos de las transiciones y la posibilidad de invalidar e implementar métodos para condiciones (guardas).

UModel puede crear operaciones simples (sin parámetros) automáticamente para comportamientos entrar/salir/hacer, efectos de transición, etc. cuando se activa la opción correspondiente (consulte el apartado Crear estados, actividades y transiciones.) Estos métodos se pueden cambiar (añadiéndoles parámetros, configurándolos como métodos abstractos, etc.).

Puede generar instancias de una máquina de estados (es decir, de su clase controladora controller) y todas las instancias funcionan independientemente.

- La ejecución de la máquina de estados UML está diseñada para el *modelo de ejecución hasta el final*.
- Las máquinas de estados UML suponen que el procesamiento de cada evento finaliza antes de que empiece a procesarse el siguiente evento.
- Esto también significa que las acciones entrar/salir/hacer y los efectos de las transiciones no pueden disparar transiciones/cambios de estado nuevos directamente.

Inicialización

- Cada región de una máquina de estados debe tener un estado inicial.
- El código generado con UModel inicializa automáticamente todas las regiones de la máquina de estados (o cuando se llama al método Initialize() de la clase controladora).
- Si no necesita eventos `OnEntry` durante la inicialización, puede llamar a mano al método Initialize() e ignorar los eventos `OnEntry` durante el inicio.

Obtener el estado actual

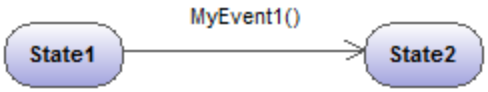
UModel admite estados compuestos y estados ortogonales, así que no hay un solo estado actual: cada región (de cualquier nivel jerárquico) puede tener un estado actual.

En el proyecto de ejemplo `AirCondition.ump` puede ver cómo se pueden recorrer las regiones hasta llegar a los estados actuales:

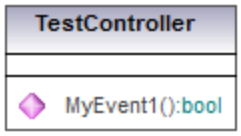
```
TreeNode rootNode = m_CurrentStateTree.Nodes.Add(m_STM.getRootState().getName());
UpdateCurrentStateTree(m_STM.getRootState(), rootNode);

private void UpdateCurrentStateTree(AirCondition.AirConditionController.IState state,
TreeNode node)
{
    foreach (AirCondition.AirConditionController.IRegion r in state.getRegions())
    {
```

```
      TreeNode childNode = node.Nodes.Add(r.getName() + " : " +
r.getCurrentState().getName());
      UpdateCurrentStateTree(r.getCurrentState(), childNode);
    }
}
```

Ejemplo nº1: una transición simple



La operación correspondiente se genera automáticamente en UModel.



Método generado en el código:

```
private class CTestStateMachine : IState
{
        …
        public bool MyEvent1()
        {
                …
        }
}
```

Notas:

- El usuario de la máquina de estados debería llamar al método generado "MyEvent1" cuando tenga lugar el evento correspondiente (fuera de la máquina de estados).
- El parámetro de devolución de estos métodos-evento aporta información si el evento provocó un cambio de estado (es decir, si tuvo un efecto o no en la máquina de estados). Por ejemplo, si "State1" está activo y ocurre el evento "MyEvent1()", entonces el estado actual cambia a "State2" y "MyEvent1()" devuelve true. Si "State2" está activo y ocurre el evento "MyEvent1()", nada cambia en la máquina de estados y MyEvent1() devuelve false.

Ejemplo nº2: una transición simple con un efecto

La operación correspondiente se genera automáticamente en UModel

Método generado en el código:

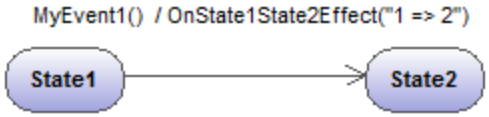
```
private class CTestStateMachine : IState
{
        …
        // Override to handle entry/exit/do actions, transition effects,...:
        public virtual void OnState1State2Effect() {}
}
```

Notas:

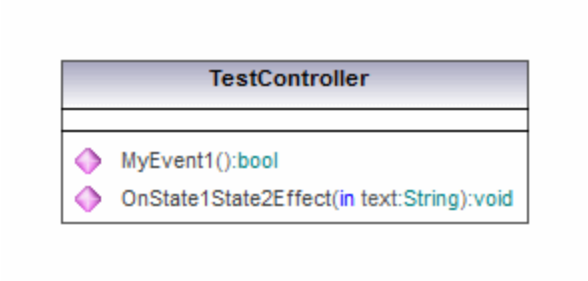
- La implementación de la máquina de estados llamará a "OnState1State2Effect()" cuando se dispare la transición del estado "State1" al estado "State2".
- Para reaccionar a este efecto "OnState1State2Effect()" debería sobrescribirse en una clase derivada de "CTestStateMachine".
- "CTestStateMachine:: OnState1State2Effect()" también puede configurarse como abstract y obtendrá errores de compilación hasta que se sobrescriba el método.
- Cuando "OnState1State2Effect()" no es abstracto y está activa la opción Generar mensajes de depuración, UModel genera este resultado:

```
// Override to handle entry/exit/do actions, transition effects,...:
public virtual void OnState1State2Effect() {OnDebugMessage("ACTION:
OnState1State2Effect");}
```

Ejemplo nº3: una transición simple con un efecto y un parámetro



La operación correspondiente se genera automáticamente en UModel.



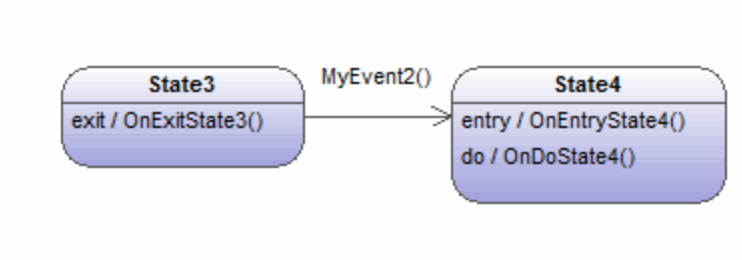
Método generado en el código:

```
private class CTestStateMachine : IState
{
        …
        // Override to handle entry/exit/do actions, transition effects,...:
        public virtual void OnState1State2Effect() {}
}
```

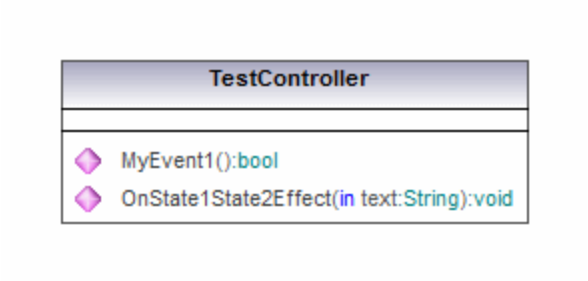
Notas:

- Para llevar a cabo las operaciones (creadas automáticamente por UModel), puede añadir parámetros manualmente (UModel no puede conocer el tipo necesario).
- En este ejemplo el parámetro "text:String" se añadió al método Effect de TestController. Es necesario especificar un argumento adecuado cuando se llame a este método (en este caso: "1 => 2").
- Otra posibilidad es llamar a los métodos estáticos ("MyStatic.OnState1State2Effect("1 => 2")") o a los métodos de singleton ("getSingleton().OnState1State2Effect("1 => 2")").

Ejemplo nº4: acciones entrar/salir/hacer



Las operaciones correspondientes se generan automáticamente en UModel.



Método generado en el código:

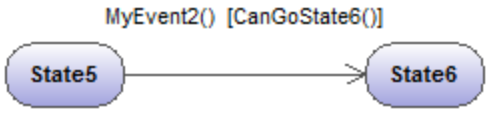
```
private class CTestStateMachine : IState
{
        …
        // Override to handle entry/exit/do actions, transition effects,...:
        public virtual void OnExitState3() {}
        public virtual void OnEntryState4() {}
        public virtual void OnDoState4() {}
}
```

Notas:

- Los estados pueden tener comportamientos entrar/salir/hacer. UModel crea automáticamente las operaciones necesarias para ellos.
- Cuando tiene lugar "MyEvent2()", la implementación de la máquina de estados llama a "OnExitState3()", si "MyEvent2" tuviera un efecto, se le llamaría después y posteriormente se llamaría a "OnEntryState4" y "OnDoState4".
- Por lo general estos métodos deberían sobrescribirse. Cuando no son abstractos y está activa la opción *Generar mensajes de depuración*, UModel genera el resultado que se describe en el ejemplo nº2.
- Estos métodos también pueden tener los parámetros que aparecen en el ejemplo nº3.

Ejemplo nº5: guardas

Las transiciones pueden tener guardas, que determinan si la transición se dispara realmente.



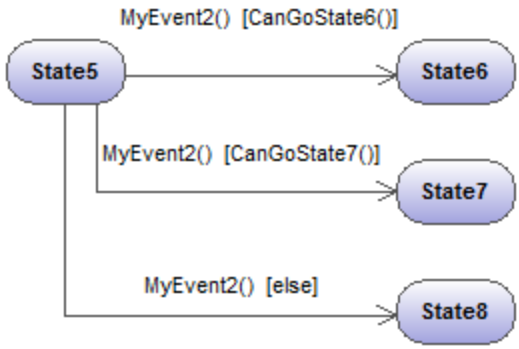
La operación correspondiente se genera automáticamente en UModel.

Método generado en el código:

```
private class CTestStateMachine : IState
{
        …
        // Additional defined operations of the controller class:
        public virtual bool CanGoState6()
        {
            return true; // Override!
        }
}
```

Notas:

- Si "State5" es el estado activo y tiene lugar "MyEvent2", la implementación de la máquina de estados llamará a "CanGoState6" y, dependiendo de su resultado, la transición se disparará o no.
- Por lo general estos métodos deberían sobrescribirse. Cuando no son abstractos y está activa la opción *Generar mensajes de depuración*, UModel genera el resultado que se describe en el ejemplo nº2.
- Estos métodos también pueden tener los parámetros que aparecen en el ejemplo nº3.
- Varias transiciones pueden tener el mismo evento, pero guardas diferentes. No hay un orden definido para sondear los guardas. Si una transición no tiene guarda o si su guarda es "else", se trata como la última transición (es decir, esta transición solo se disparará si los guardas de las demás transiciones devuelven false. Por ejemplo, no está definido si primero se dispara CanGoState6 o CanGoState7, pero lo que está claro es que la tercera transición solo se disparará si CanGoState6 y CanGoState7 devuelven `false`.



Para ver más funciones y construcciones consulte los ejemplos de los archivos AirCondition.ump y Complex.ump.

9.1.2.6 Elementos de diagramas de máquinas de estados

 EstadoInicial (pseudoestado)

El inicio del proceso.

 EstadoFinal

El final de la sucesión de procesos.

 PuntoDeEntrada (pseudoestado)

El punto de entrada de una máquina de estados o de un estado compuesto.

 PuntoDeSalida (pseudoestado)

El punto de salida de una máquina de estados o de un estado compuesto.

 Elección

Representa una rama condicional dinámica donde se evalúan disparadores de guardas que se excluyen mutuamente (operación OR).

 Unión (pseudoestado)

Representa el final de la operación OR definida por el elemento **`Elección`**.

 Terminar (pseudoestado)

La detención de la ejecución de la máquina de estados.

 Bifurcación (pseudoestado)

Inserta una barra de bifurcación vertical. Sirve para dividir secuencias en subsecuencias simultáneas.

 Bifurcación horizontal (pseudoestado)

Inserta una barra de bifurcación horizontal. Sirve para dividir secuencias en subsecuencias simultáneas.

Reunión (pseudoestado)

Reúne/combina subsecuencias definidas previamente. Para poder continuar es necesario completar todas las actividades.

Reunión horizontal (pseudoestado)

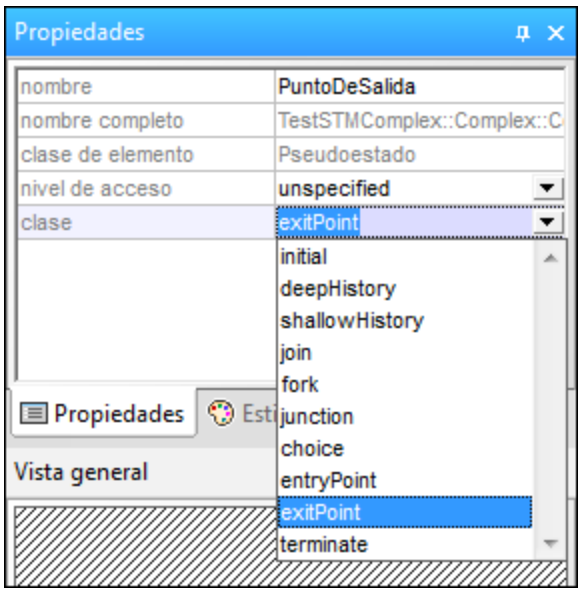
Reúne/combina subsecuencias definidas previamente. Para poder continuar es necesario completar todas las actividades.

HistorialDetallado

Pseudoestado que restaura el estado activo previo del estado dentro de un estado compuesto.

HistorialSuperficial

Pseudoestado que restaura el estado inicial de un estado compuesto. Para cambiar el tipo de pseudoestado, cambie el valor del cuadro combinado clase en la ventana *Propiedades*.



ReferenciaDePuntoDeConexión

Una referencia de punto de conexión representa un uso (como parte de un estado de submáquina) de un punto de entrada/salida definido por el estado de submáquina en la referencia de la máquina de estados.

Para agregar puntos de entrada o salida a una referencia de punto de conexión:

- El estado al que está conectado el punto debe hacer referencia a una máquina de estados de submáquina (visible en la ventana *Propiedades*).

- Esta submáquina debe contener un punto de entrada y salida como mínimo.

 Transición

La relación directa que existe entre dos estados. Un objeto del primer estado realiza una acción o más y después hace referencia al segundo estado, dependiendo de un evento y de que se cumplan las condiciones de protección.

Las transiciones tienen un disparador de eventos, condiciones de protección, una acción (comportamiento) y un estado de destino.Los subelementos de complemento compatibles son:

- **EventoRecibirSeñal**
- **EventoSeñal**
- **EventoEnviarSeñal**
- **EventoRecibirOperación**
- **EventoEnviarOperación**
- **EventoDeCambio**.

 Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación

Active este icono para crear automáticamente la operación correspondiente en la clase a la que se hace referencia cuando se cree una transición y se inserte el nombre de la operación.

Nota: solamente se pueden crear operaciones automáticamente cuando la máquina de estado está dentro de una clase o de una interfaz.

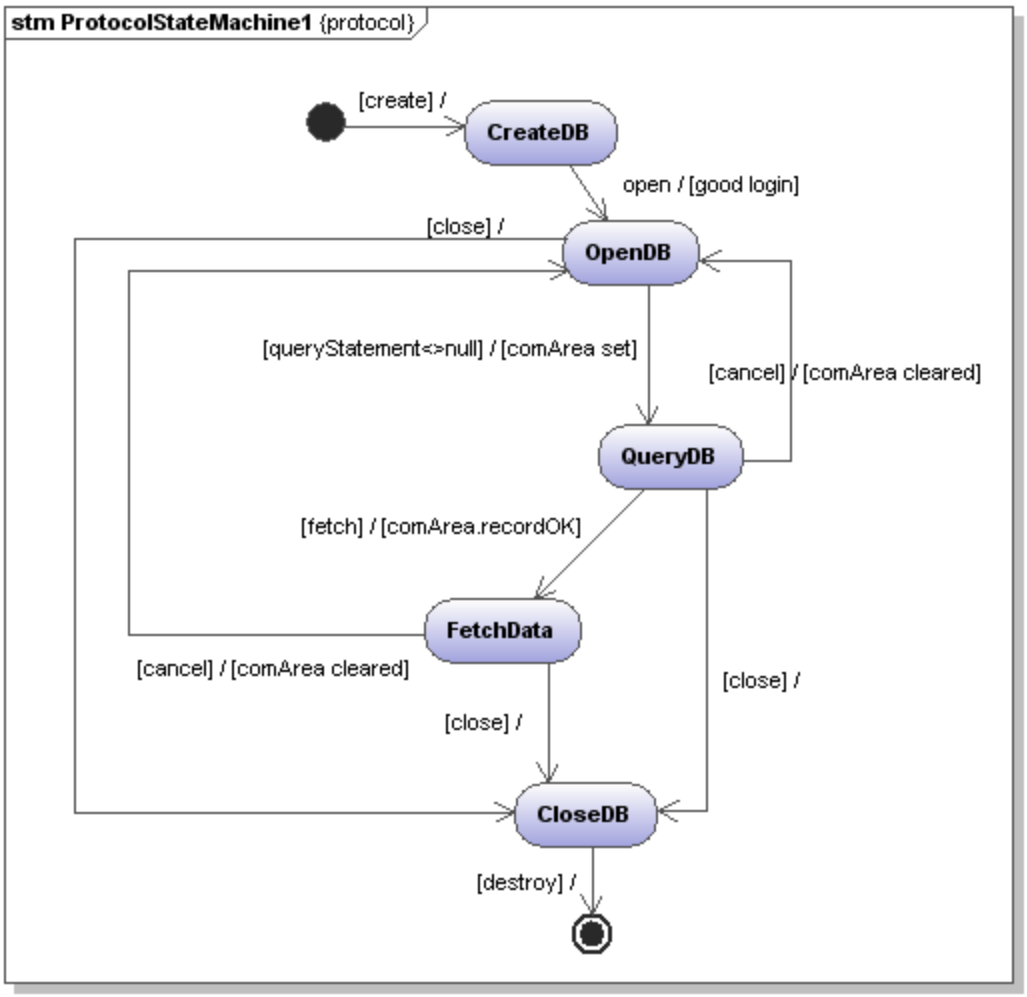
9.1.3 Diagrama de máquina de estados de protocolos

Sitio web de Altova: Diagramas de máquina de estados de protocolos UML

Las máquinas de estados de protocolos ilustran una **secuencia** de eventos a los que responde un objeto, sin necesidad de ilustrar su comportamiento propiamente dicho. La secuencia necesaria de eventos y los cambios resultantes en el estado del objeto se modelan en este tipo de diagramas.

Las máquinas de estados de protocolos se usan sobre todo para describir protocolos complejos. Por ejemplo, el acceso a bases de datos a través de una interfaz determinada o protocolos de comunicación como TCP/IP.

Las máquinas de estados de protocolos se crean igual que los diagramas de máquina de estados, pero tienen menos elementos de modelado. Las transiciones de protocolo entre los estados pueden tener condiciones previas o posteriores que definen qué debe ocurrir para que tenga lugar la transición a otro estado o cuál debe ser el estado resultante una vez tiene lugar la transición.



9.1.3.1 Insertar elementos

Para insertar elementos con los iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de máquina de estados de protocolos.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama:

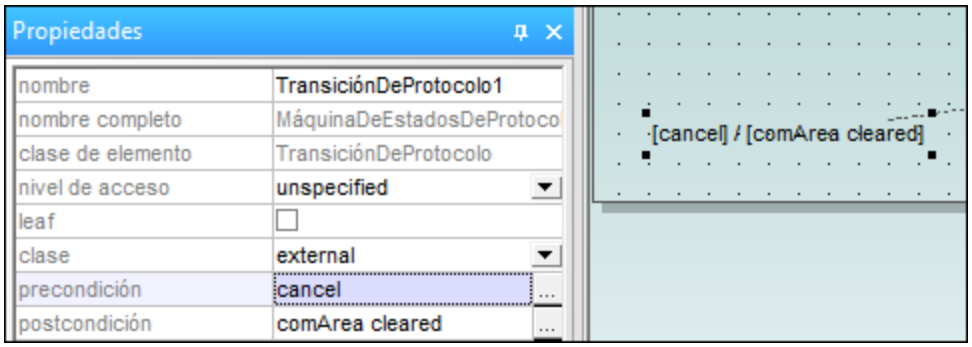
1. En la *Estructura del modelo* busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de máquina de estados de protocolos.

Para insertar un estado simple:

1. Haga clic en el icono **Estado** de la barra de herramientas y haga clic en el diagrama para insertarlo.
2. Escriba el nombre del estado y pulse **Entrar** para confirmar.
   Los estados simples no tienen regiones ni subestructuras.

Para crear una transición de protocolo entre dos estados:

1. Haga clic en el controlador Transición del estado de origen (situado a la derecha del elemento) o en el icono **TransiciónDeProtoclo** de la barra de herramientas.
2. Arrastre la flecha de la transición hasta el estado de destino.
   El cursor de texto se habilita automáticamente para que pueda insertar la condición previa o posterior.
   Recuerde que es obligatorio utilizar corchetes y la barra diagonal en las condiciones.
   Si inserta la condición (previa o posterior) en la ventana Propiedades, los corchetes y la barra diagonal se escriben automáticamente en el diagrama.



Para crear e insertar estados compuestos y estados de submáquina, consulte el apartado Estados compuestos.

9.1.3.2 Elementos

Estado

Estado simple con un compartimento.

Estado compuesto

Este tipo de estado contiene un compartimento más que tiene una sola región. Dentro de esta región puede colocar un número ilimitado de estados.

Estado ortogonal

Este tipo de estado contiene un compartimento más, formado por dos o más regiones, que indican simultaneidad.

Haga clic con el botón derecho en un estado y seleccione **Nuevo/a | Región** para añadir una región nueva.

Estado de submáquina

Este estado sirve para ocultar detalles de una máquina de estados. Este estado no tiene regiones pero está asociado a una máquina de estados distinta.

EstadoInicial (pseudoestado)

El principio del proceso

EstadoFinal

El fin de la secuencia de los procesos

PuntoDeEntrada (pseudoestado)

El punto de entrada de una máquina de estados o de un estado compuesto.

PuntoDeSalida (pseudoestado)

El punto de salida de una máquina de estados o de un estado compuesto.

Elección

Representa una rama condicional dinámica en la que se evalúan disparadores de guardas que se excluyen mutuamente (operación OR).

Unión (pseudoestado)

Representa el final de la operación OR definida por el elemento Elección.

Terminar (pseudoestado)

La detención de la ejecución de la máquina de estados.

Bifurcación (pseudoestado)

Inserta una barra de bifurcación vertical. Sirve para dividir secuencias en subsecuencias simultáneas.

Bifurcación horizontal (pseudoestado)

Inserta una barra de bifurcación horizontal. Sirve para dividir secuencias en subsecuencias simultáneas.

Reunión (pseudoestado)

Reúne/combina subsecuencias definidas previamente. Para poder continuar todas las actividades deben completarse.

Reunión horizontal (pseudoestado)

Reúne/combina subsecuencias definidas previamente. Para poder continuar todas las actividades deben completarse.

ReferenciaDePuntoDeConexión

Representa un uso (como parte de un estado de submáquina) de un punto de entrada/salida definido en la referencia de máquina de estados por el estado de submáquina.

Para añadir puntos de entrada/salida en una referencia de punto de conexión:

- El estado al que está conectado el punto debe hacer referencia a una máquina de estado de submáquina (visible en la ventana *Propiedades*).
- Esta submáquina debe contener un punto de entrada y otro de salida como mínimo.

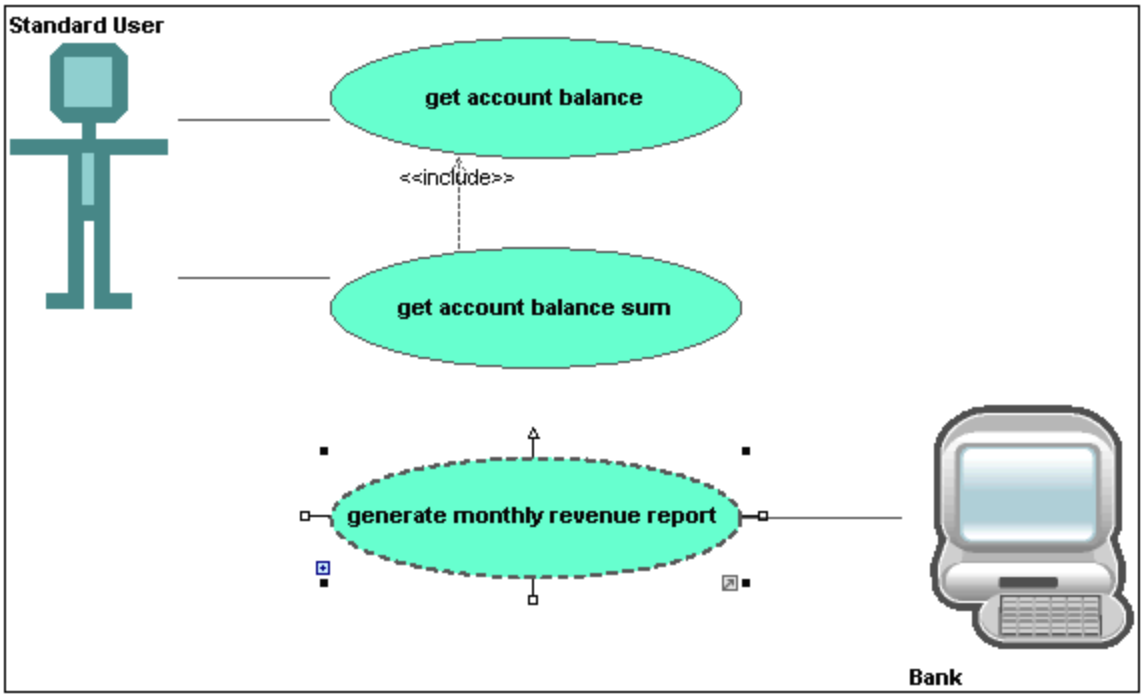
TransiciónDeProtocolo

Relación directa entre dos estados. Un objeto del primer estado realiza una operación o más y después hace referencia al segundo estado, dependiendo de un evento y de que se cumplan las condiciones previas o posteriores.

Para más información consulte el apartado Insertar elementos.

9.1.4 Diagrama de casos de uso

Consulte Casos de uso del tutorial para obtener más información sobre cómo usar los diagramas de casos de uso.



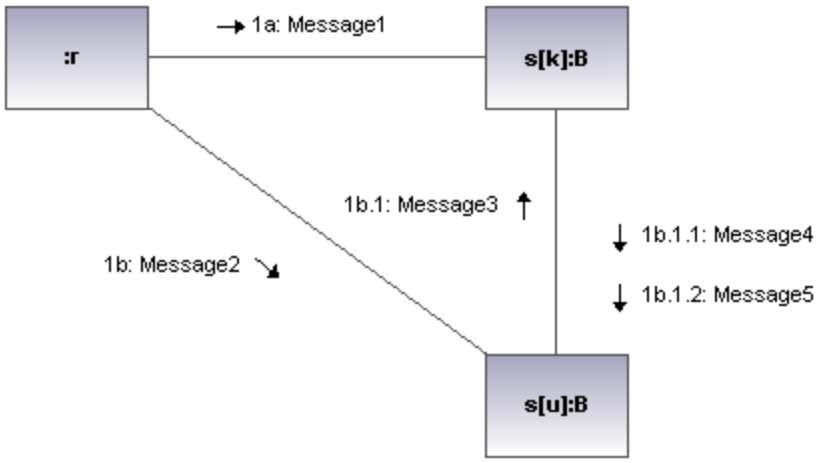
9.1.5 Diagrama de comunicación

Sitio web de Altova: Diagramas de comunicación UML

Los diagramas de comunicación muestran cómo interactúan los objetos en tiempo de ejecución (p. ej. los flujos de mensaje) e ilustran las relaciones que existen entre los objetos. Básicamente modelan el comportamiento dinámico de los casos de uso.

Los diagramas de comunicación se diseñan igual que los diagramas de secuencia, excepto que la notación tiene otro formato. Los mensajes se numeran para ilustrar su secuencia y su anidamiento.

Con UModel puede generar diagramas de comunicación a partir de diagramas de secuencia y viceversa. Para más información consulte el apartado Generar diagramas de secuencia.



9.1.5.1 Insertar elementos

Para insertar elementos con los iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de comunicación.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Para arrastrar elementos desde la Estructura del modelo hasta el diagrama:

1. En la *Estructura del modelo* busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de comunicación.

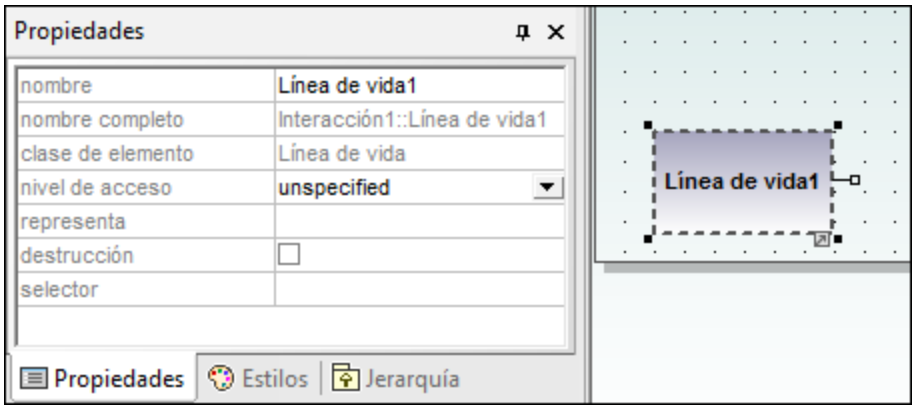
Línea de vida

El elemento línea de vida es un participante de la interacción. En UModel puede insertar otros elementos (clases, por ejemplo) en el diagrama de secuencia. Cada elemento aparece como una línea de vida nuevas. El color y el degradado de las líneas de vida se pueden redefinir en el cuadro combinado Título - color de degradado de la ventana *Estilos*.

Para crear un nombre de línea de vida **multilínea** pulse **Ctrl+Entrar**.

Para insertar una línea de vida de comunicación:

1. Haga clic en el icono **Línea de vida** de la barra de herramientas y después haga clic en el área de trabajo del diagrama para insertarla.



2. Escriba el nombre de la línea de vida o conserve el nombre predeterminado `Línea de vida 1`.

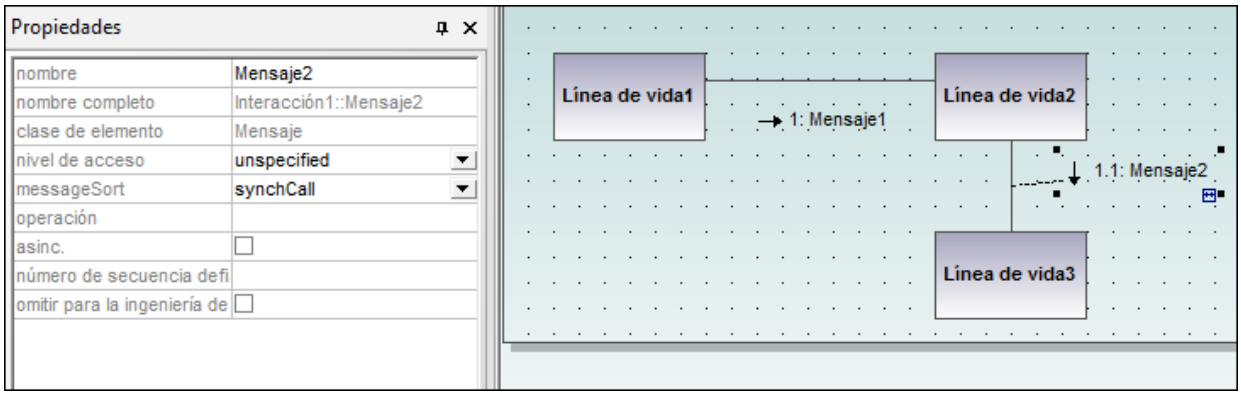
Mensajes

Un mensaje es un elemento de modelado que define un tipo concreto de comunicación en una interacción. Una comunicación puede lanzar una señal, invocar una operación, crear o destruir una instancia, etc. El mensaje especifica el tipo de comunicación, así como el remitente y el destinatario.

Mensaje (Llamada) **Mensaje (Respuesta)** **Mensaje (Creación)** **Mensaje (Destrucción)**

Para insertar un mensaje:

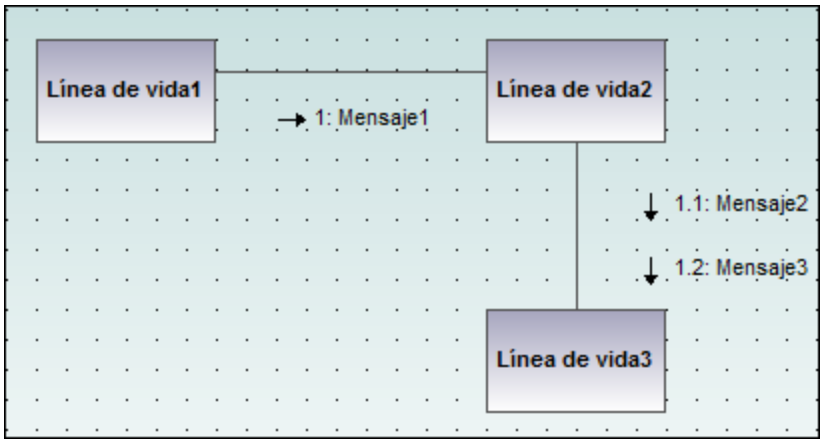
1. En la barra de herramientas haga clic en el icono del mensaje que desea insertar.
2. Ahora haga clic en el remitente y arrastre el puntero hasta el destinatario (un destinatario válido es el que aparece resaltado al pasar el puntero por encima).



Nota: mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo para insertar varios mensajes.

Para insertar más mensajes:

1. Haga clic con el botón derecho en una línea de comunicación del diagrama y seleccione **Nuevo/a | Mensaje**.



- La dirección en la que se arrastra la flecha define la dirección del mensaje.
- Los mensajes de respuesta pueden apuntar en ambas direcciones.

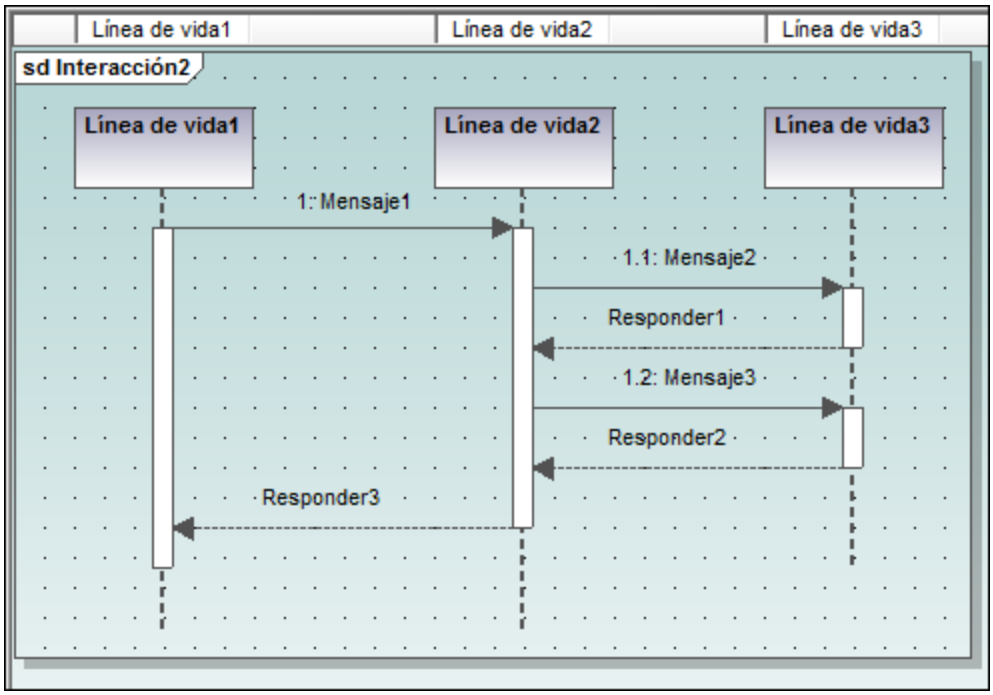
Numeración de los mensajes

Los diagramas de comunicación utilizan la notación decimal para numerar los mensajes, lo cualquier facilita comprender la estructura jerárquica de los mensajes del diagrama. La secuencia es una lista separada por puntos de números en secuencia seguidos por dos puntos y el nombre del mensaje.

Generar diagramas de secuencia a partir de diagramas de comunicación

UModel puede generar diagramas de comunicación a partir de diagramas de secuencia y viceversa:

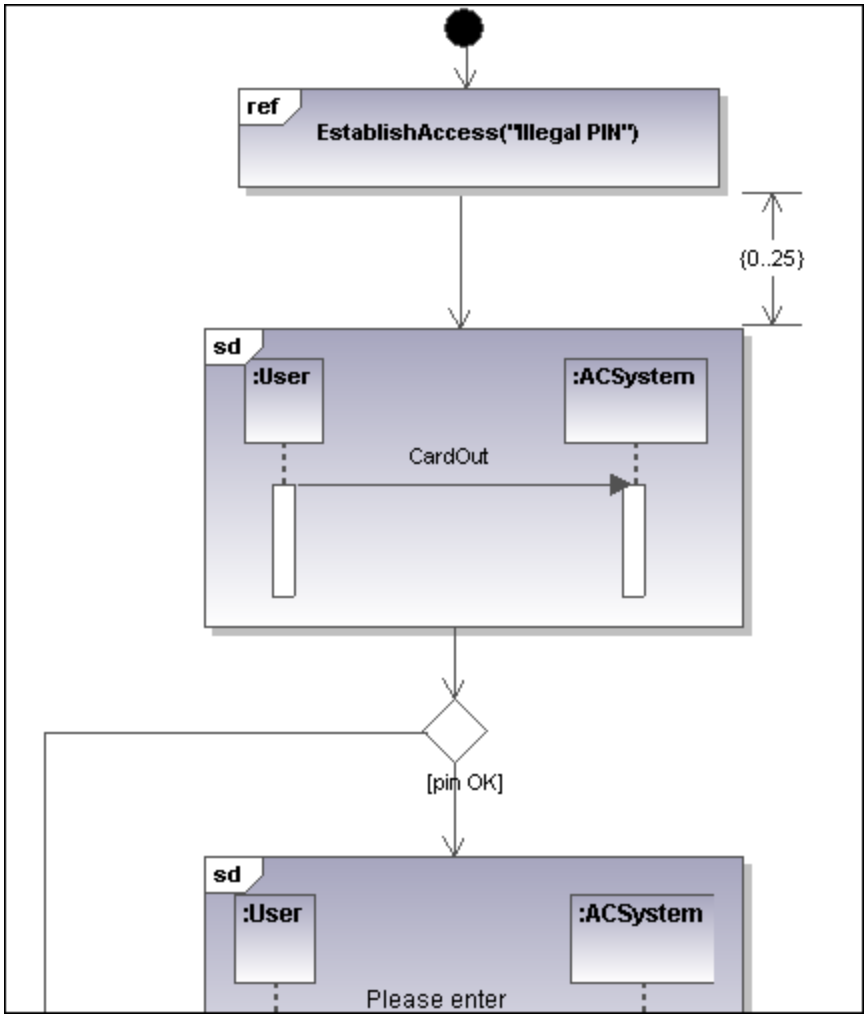
1. Haga clic con el botón derecho en el área de trabajo del diagrama de comunicación.
2. Seleccione **Generar diagrama de secuencia** en el menú contextual.



9.1.6 Diagrama global de interacción

Sitio web de Altova: Diagramas globales de interacción

Los diagramas globales de interacción son un tipo de diagrama de actividades que ofrecen un resumen de la interacción entre otros diagramas de interacción como diagramas de secuencia, de actividades, de comunicación o de ciclo de vida. El método para construir este tipo de diagramas es similar al utilizado para los diagramas de actividades y usa los mismos elementos de modelado: bifurcaciones, reuniones, nodo inicial, nodo final, etc.



En lugar de actividades, este diagrama utiliza dos tipos distintos de interacciones: **Interacción** y **UsoDeInteracción**.

Los elementos **Interacción** se presentan como iconos de un diagrama de secuencia, comunicación, ciclo de vida o diagrama global de interacción, en un marco que tiene la abreviatura `SD` en la esquina superior izquierda.

Las instancias de los elementos **Interacción** son referencias a diagramas de interacción ya disponibles. Éstas tienen la abreviatura `Ref` y el nombre de la instancia en el marco de título.

9.1.6.1 Insertar elementos

Para insertar elementos con los iconos de la barra de herramientas

1. Haga clic en un icono de la barra de herramientas Diagrama global de interacción.

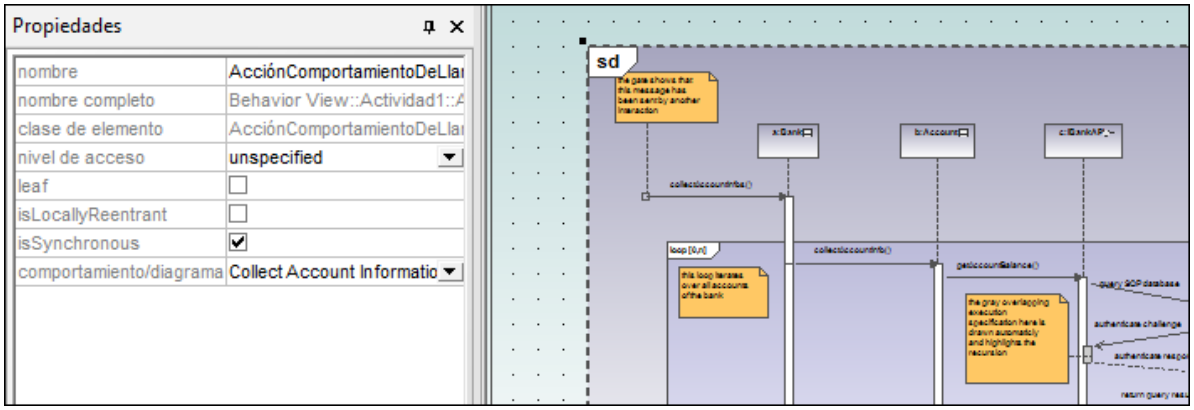
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Para arrastrar elementos desde la Estructura del modelo hasta el diagrama

1. En la *Estructura del modelo* busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama global de interacción.

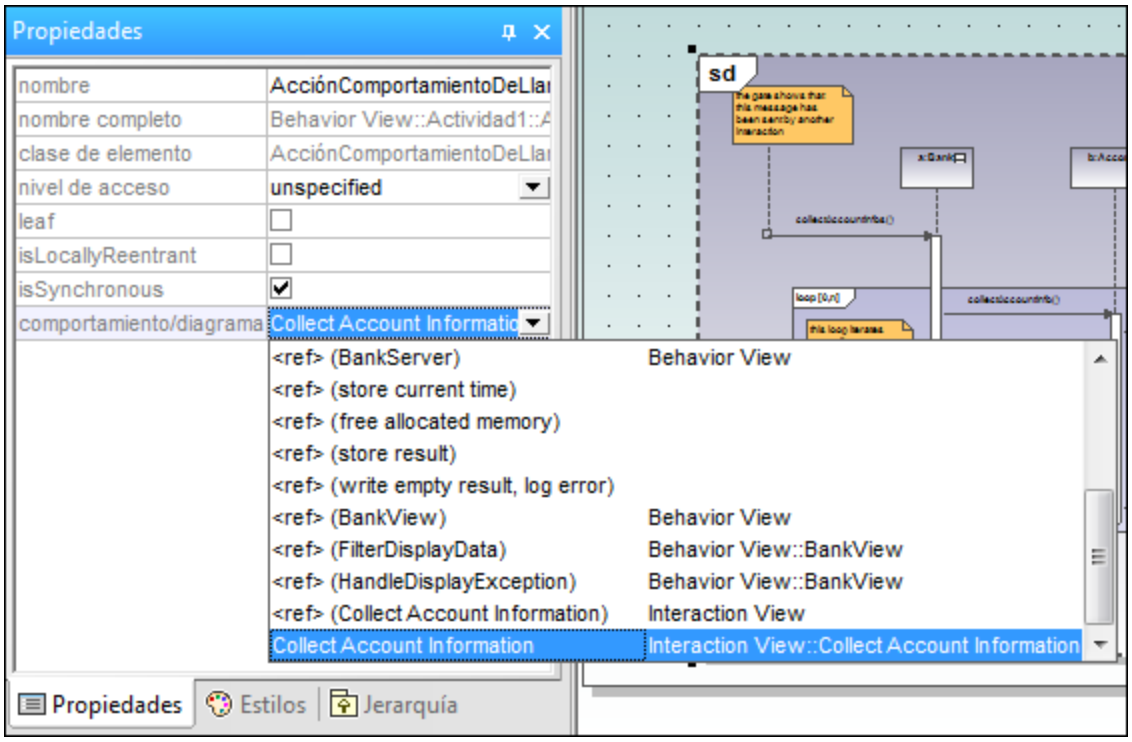
Para insertar un elemento Interacción

1. Haga clic en el icono **AcciónComportamientoDeLlamada (Interacción)**  de la barra de herramientas y haga clic en el área de trabajo del diagrama para insertar la interacción.

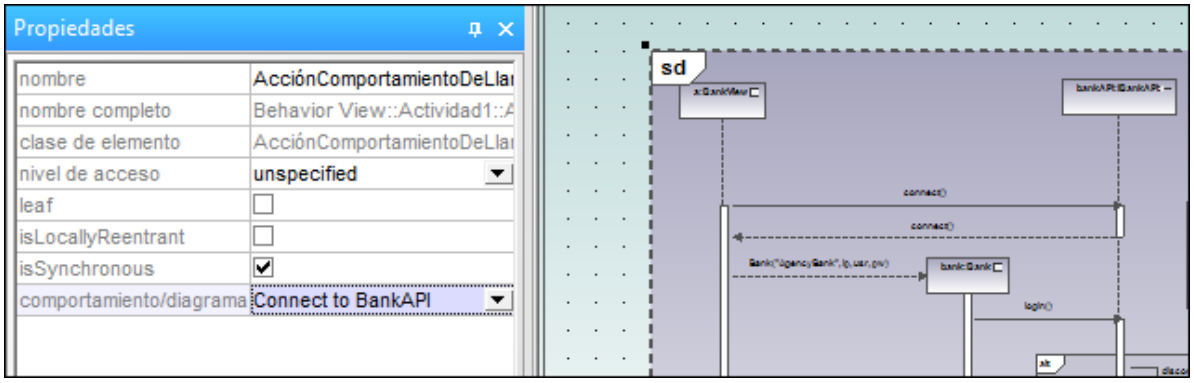


Si usa el archivo de ejemplo Bank_MultiLanguage.ump de la carpeta ...\UModelExamples, el diagrama de secuencia **`Collect Account Information`** se inserta automáticamente. El primer diagrama de secuencia de la *Estructura del modelo* se selecciona por defecto.

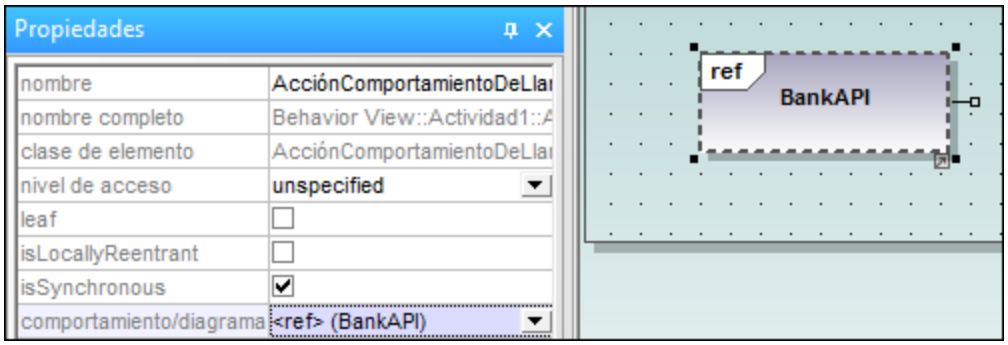
2. Para cambiar el elemento **Interacción** predeterminado, haga clic en el cuadro combinado comportamiento/diagrama de la ventana *Propiedades*. La lista desplegable incluye todos los elementos que se pueden insertar.



3. Haga clic en el elemento que desea insertar (p. ej. **`Connect to BankAPI`**).

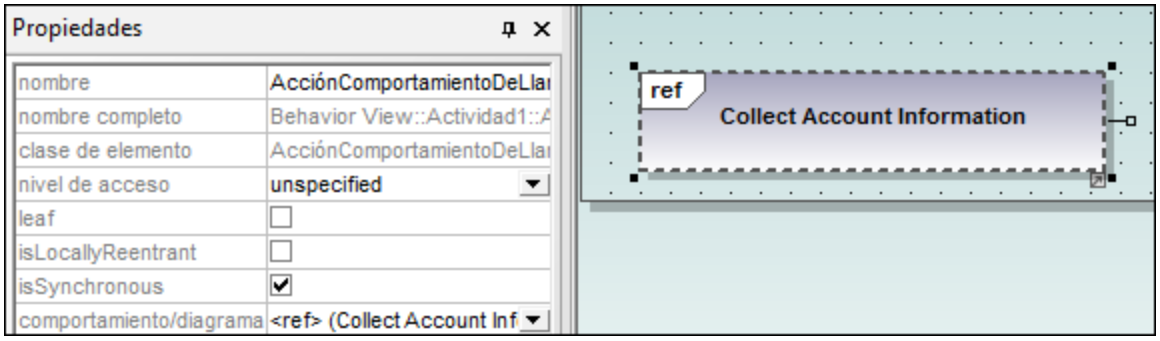


Como este también es un diagrama de secuencia, el elemento **Interacción** aparece como un icono que representa el diagrama de secuencia.
Si selecciona **`<ref> BankAPI`**, aparece la instancia del elemento **Interacción**.



Para insertar una instancia del elemento Interacción

1. Haga clic en el icono **AcciónComportamientoDeLlamada (UsoDeInteracción)** de la barra de herramientas y haga clic en el área de trabajo del diagrama para insertar la instancia.
   Si usa el archivo de ejemplo Bank_MultiLanguage.ump de la carpeta ...\UModelExamples, UModel inserta `Collect Account Information` automáticamente como instancia de interacción. El primer diagrama de secuencia disponible se selecciona por defecto.



2. Para cambiar de elemento **Interacción** haga clic en el cuadro combinado comportamiento/diagrama de la ventana *Propiedades*. La lista desplegable incluye todos los elementos disponibles que se pueden insertar.
3. Seleccione la instancia que desea insertar.

Nota: todos los elementos que se insertan de esta manera aparecen como en la imagen anterior, es decir, con la abreviatura **ref** en el marco de título.

NodoDeDecisión

Inserta un nodo de decisión que tiene una sola transición entrante y varias transiciones salientes protegidas por guardas. Para más información consulte el apartado Crear una rama.

NodoDeCombinación

Inserta un nodo de combinación que une las transiciones alternas definidas por el nodo de decisión. El nodo de combinación no sincroniza los procesos simultáneos, sino que selecciona uno de los procesos.

 NodoInicial

El principio del proceso. Una interacción puede tener más de un nodo inicial.

 NodoFinalDeActividad

El final del proceso de interacción. Una interacción puede tener más de un nodo final. Todos los flujos se detienen cuando se encuentra el primer nodo final.

 NodoDeBifurcación

Inserta un nodo de bifurcación vertical. Sirve para dividir flujos en varios flujos simultáneos.

 NodoDeBifurcación (Horizontal)

Inserta un nodo de bifurcación horizontal. Sirve para dividir flujos en varios flujos simultáneos.

 NodoDeReunión

Inserta un nodo de reunión vertical. Sirve para sincronizar varios flujos definidos por el nodo de bifurcación.

 NodoDeReunión (horizontal)

Inserta un nodo de reunión horizontal. Sirve para sincronizar varios flujos definidos por el nodo de bifurcación.

 RestricciónDeDuración

Una duración define una EspecificaciónDeValor que denota una duración entre un punto inicial y un punto final. Las duraciones suelen ser expresiones que representan el tiempo que puede pasar.

 FlujoDeControl

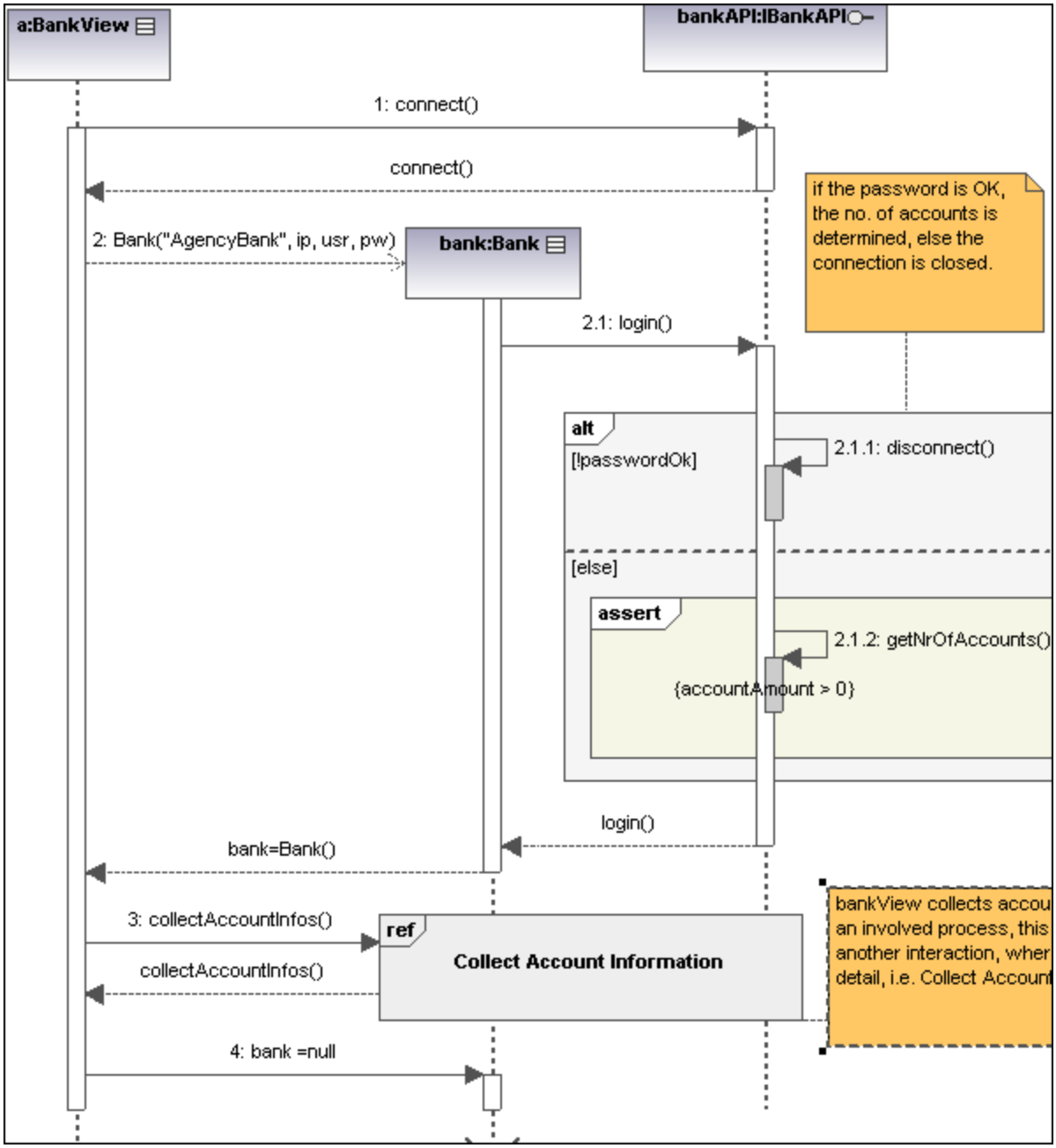
Un flujo de control es una línea con una flecha que conecta dos comportamientos e inicia una interacción después de que finalice la anterior.

9.1.7 Diagrama de secuencia

Sitio web de Altova: Diagramas de secuencia UML

En UModel puede crear los diagramas de secuencia estándar definidos por UML y manipular objetos y mensajes con total facilidad para modelar casos de uso. Los diagramas de secuencia que aparecen en esta sección proceden de los proyectos de ejemplo Bank_Java.ump, Bank_CSharp.ump y Bank_MultiLanguage.ump del directorio ...\UModelExamples.

Puede modelar diagramas de secuencia manualmente o, como alternativa, generarlos con ingeniería inversa a partir de código fuente, como se describe en el apartado Generar diagramas de secuencia.La API de UModel también permite generar o modelar diagramas de secuencia de forma programática; *consulte Cómo crear diagramas de secuencia*.



9.1.7.1 Insertar elementos

Los diagramas de secuencia modelan las interacciones dinámicas de los objetos en tiempo de ejecución por medio de mensajes. Suelen utilizarse para explicar casos de uso.

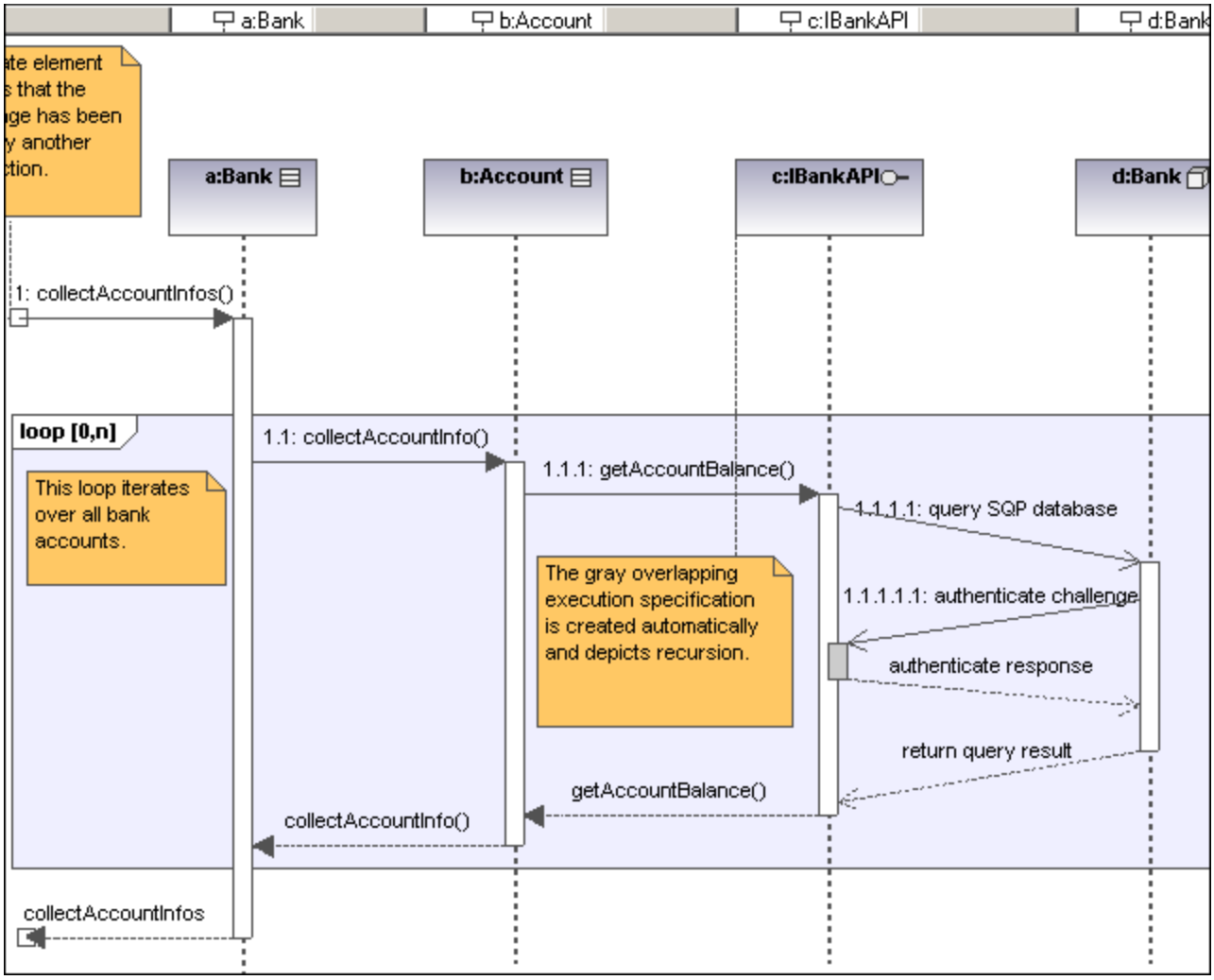
- Las **líneas de vida** son recuadros alineados horizontalmente en la parte superior del diagrama y tienen una línea de puntos vertical que representa la vida del objeto durante la interacción. Los mensajes se dibujan como flechas entre las líneas de vida de los objetos.

- Los **mensajes** se envían de un objeto a otro, se dibujan en forma de flecha y tienen una etiqueta de texto. Pueden tener un número secuencial y otros atributos opcionales como listas de argumentos, etc. Los mensajes pueden ser condicionales, opcionales y alternativos. Para más información consulte el apartado Fragmentos combinados.

Esta sección se divide en varios apartados:

- Líneas de vida
- Fragmentos combinados
- Usos de interacción
- Puertas
- Invariantes de estado
- Mensajes

En UModel hay varias maneras de insertar elementos en los diagramas de secuencia.



Insertar elementos con los iconos de la barra de herramientas

1. Haga clic en un icono de la barra de herramientas Diagrama de secuencia.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama

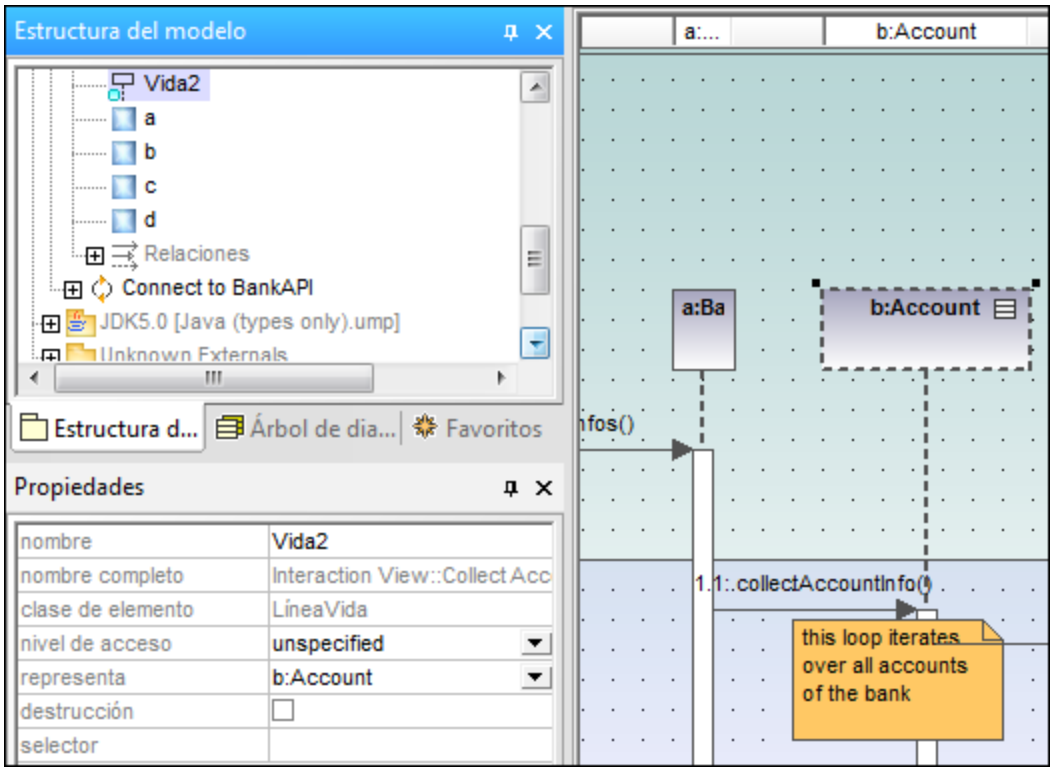
1. En la *Estructura del modelo* busque el elemento que quiere insertar en el diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de secuencia.

9.1.7.1.1 Líneas de vida

La línea de vida  es un participante de una interacción. En los diagramas de secuencia de UModel también puede insertar elementos como clases y actores. Estos elementos se representan como una línea de vida nueva.

La etiqueta de la línea de vida aparece en una barra situada en la parte superior del diagrama. Puede cambiar la posición de las etiquetas y también su tamaño. Además puede redefinir el color y el degradado de las etiquetas (en el cuadro combinado Título - color de degradado de la ventana *Estilos*). Pulse **Ctrl+Entrar** para crear una línea nueva en el nombre de la línea de vida.

En el diagrama de secuencia también puede insertar clasificadores. El campo `representa` de la ventana *Propiedades* muestra el tipo de elemento que actúa como línea de vida. Si arrastra una propiedad **con tipo** hasta el diagrama de secuencia, también se crea una línea de vida.



Especificación de ejecución (activación de objetos)

Una especificación de ejecución (activación) se representa en forma de cuadro (rectángulo) en la línea de vida del objeto. Una activación es la ejecución de un procedimiento y el tiempo necesario para ejecutar los procedimientos anidados correspondientes.

Cuando se crea un mensaje entre dos líneas de vida, se crean automáticamente los cuadros de activación.

Y un mensaje recursivo o automensaje (es decir, uno que llama a otro método de la misma clase) crea cuadros de activación apilados.

Para mostrar/ocultar los cuadros de activación:

- Abra la ventana *Estilos* y desplácese hasta el cuadro combinado Mostrar especificaciones de ejecución.

En este estilo puede definir si los cuadros de activación se muestran o se ocultan en el diagrama de secuencia.

Atributos de las líneas de vida

La casilla destrucción sirve para añadir un marcador de destrucción (o freno) a la línea de vida sin necesidad de usar un mensaje de destrucción.

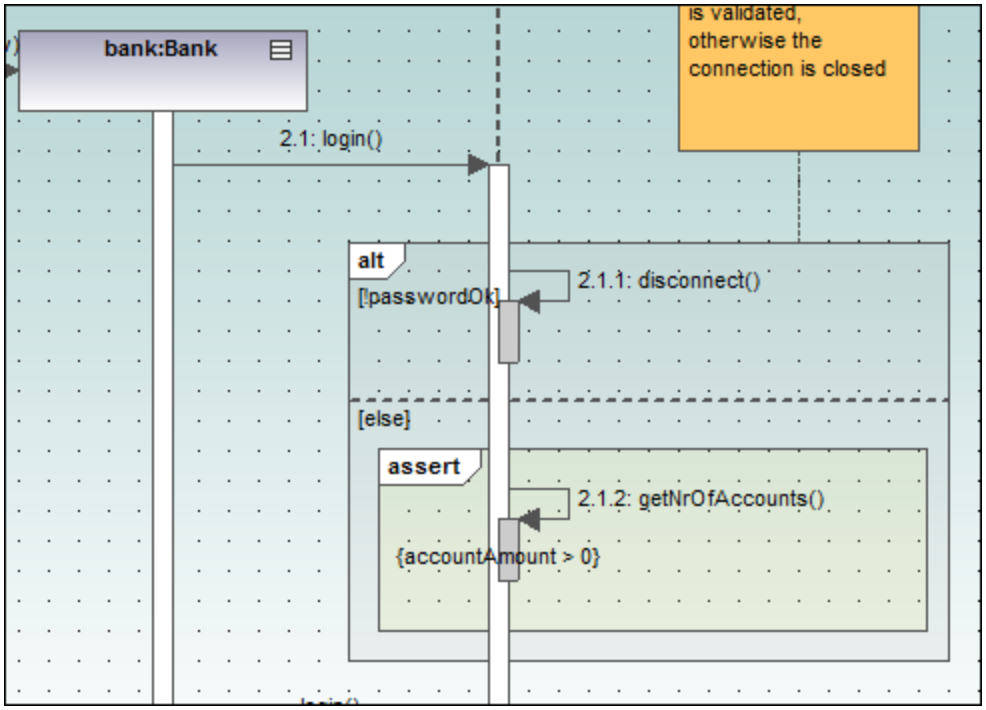
En el campo selector puede insertar una expresión que indique la parte que representa la línea de vida si el `ElementoConectable` tiene más de un valor (es decir, si su multiplicidad es mayor que 1).

Ir a una línea de vida

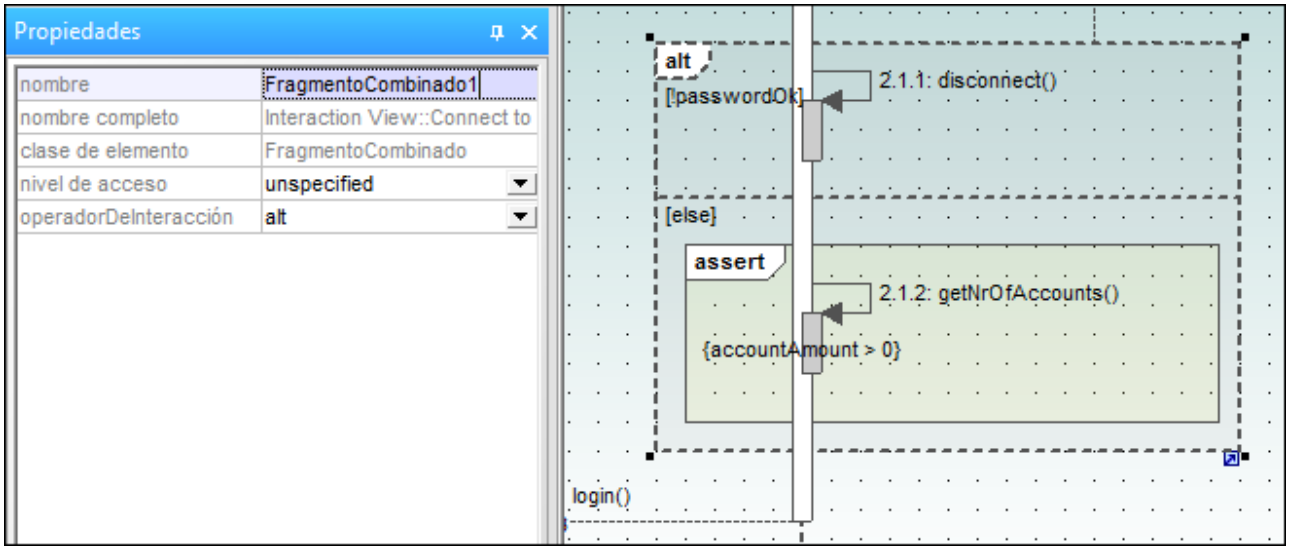
Haga clic con el botón derecho en una línea de vida y en el menú contextual elija la opción **Ir a XXX** (XXX es el tipo de línea de vida seleccionada). El elemento se resalta en la ventana *Estructura del modelo*.

9.1.7.1.2 Fragmentos combinados

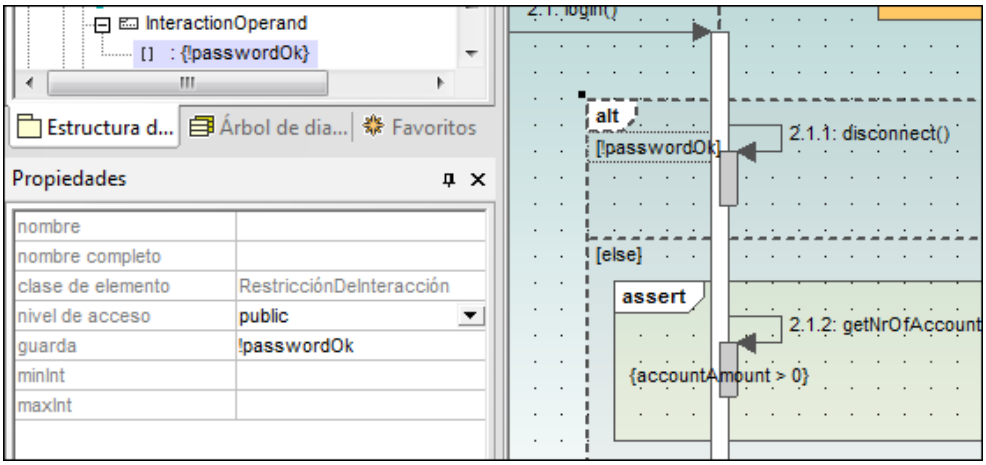
Los fragmentos combinados  son subunidades o secciones de una interacción. El operador de interacción que aparece en el pentágono de la esquina superior izquierda define el tipo de fragmento combinado. Por tanto, la restricción define el tipo de fragmento (p. ej. de bucle, alternativo, etc.) utilizado en la interacción.



La barra de herramientas de los diagramas de secuencia también incluye iconos para insertar fragmentos combinados en el diagrama: **`seq`** (secuencia), **`alt`** (alternativo) o **`loop`** (bucle). Haga clic en el cuadro combinado operadorDeInteracción para definir el tipo de fragmento de interacción.



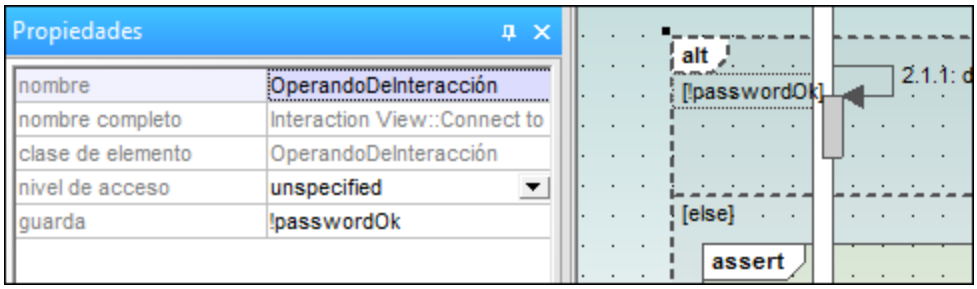
OperadoresDeInteracción

Secuencias débiles	**seq**	 El fragmento combinado representa secuencias débiles entre los comportamientos de los operandos.
Alternativas	**alt**	Solo se eligirá uno de los operandos definidos. El operando debe tener una expresión de guarda cuyo resultado sea true.  Si uno de los operandos utiliza el guarda "else", el operando se ejecuta si todas las demás guardas devuelven false. La expresión de guarda se puede introducir inmediatamente después de la inserción (y aparecerá entre corchetes).  La **RestricciónDeInteracción** es de hecho la expresión de guarda que aparece entre corchetes.
Opción	**opt**	Representa una opción entre ejecutar el operando o no hacer nada.
Pausa	**break**	El operador **break** se elige cuando el guarda es true. El resto del fragmento se ignora.
Paralelo	**par**	Indica que el fragmento combinado representa una combinación paralela de operandos.
Secuencias estrictas	**strict**	El fragmento combinado representa una secuencia estricta entre los comportamientos de los operandos.
Bucle	**loop**	El operando **loop** se repetirá tantas veces como defina la expresión de guarda. 

		Tras seleccionar este operando puede editar la expresión directamente (en el pentágono `loop`) haciendo doble clic.
Región crítica	**critical**	El fragmento combinado representa una región crítica. La secuencia no se puede interrumpir ni intercalar con otros procesos.
Negativo	**neg**	El fragmento no es válido y los demás se suponen válidos.
Aserción	**assert**	Designa el fragmento combinado válido y sus secuencias. Se suele usar junto con los operandos **consider** o **ignore**.
Ignorar	**ignore**	Define qué mensajes deben ignorarse en la interacción. Se suele usar junto con los operandos **assert** o **consider**.
Considerar	**consider**	Define qué mensajes se deben tener en cuenta en la interacción. Se suele usar junto con los operandos **assert** o **ignore**.

Para agregar operandosDeInteracción a un fragmento combinado

1. Haga clic con el botón derecho en el fragmento combinado y seleccione **Nuevo/a | OperandoDeInteracción**.
   La condición de guarda se puede editar inmediatamente.
2. Inserte la condición de guarda para el **OperandoDeInteracción** (p. ej. !passwordOK) y pulse **Entrar** para confirmar.



   Pulse **Ctrl+Entrar** para crear una línea nueva en el nombre del operando de interacción.

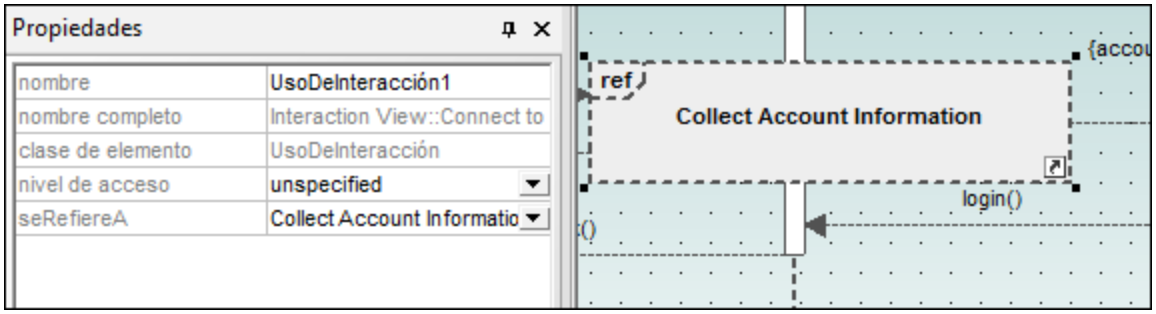
3. Use el mismo método para añadir otro operando de interacción con la condición de guarda "else".
   Los operandos aparecen separados por líneas de puntos en el fragmento.

Para eliminar operandos de interacción

1. Haga doble clic en la expresión de guarda del fragmento combinado en el área de trabajo del diagrama (no en la ventana *Propiedades*).
2. Elimine la expresión de guarda y pulse **Entrar** para confirmar.
   Como resultado se elimina la expresión de guarda / el operando de interacción y el tamaño del fragmento combinado se ajusta automáticamente.

9.1.7.1.3 Usos de interacción

El elemento **UsoDeInteracción**  es una referencia a un elemento de interacción y sirve para compartir porciones de una interacción con otras interacciones.



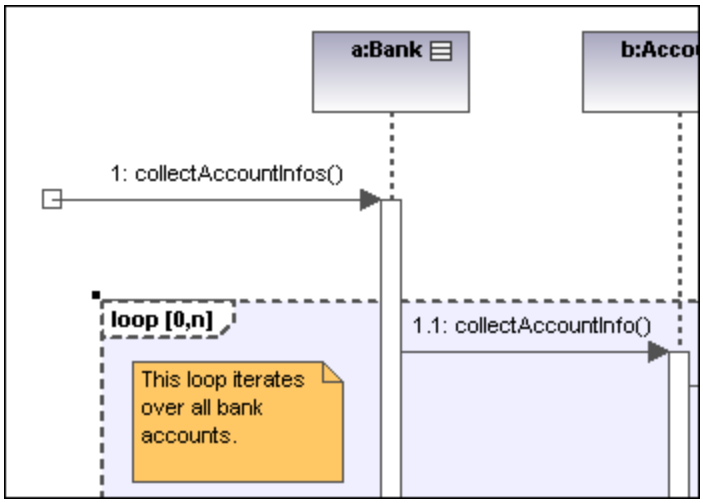
La casilla seRefiereA de la ventana *Propiedades* sirve para seleccionar la interacción a la que desea hacer referencia. El nombre del uso de interacción seleccionado aparecerá en el elemento.

Nota: también puede arrastrar un `UsoDeInteracción` desde la *Estructura del modelo* hasta el área de trabajo del diagrama.

9.1.7.1.4 Puertas

Una puerta  es un punto de conexión que permite transmitir mensajes de un fragmento a otro de la interacción. Las puertas se conectan por medio de mensajes.

1. Inserte una puerta en el diagrama.
2. Cree un mensaje nuevo, haga clic en la puerta y arrastre el puntero hasta la línea de vida (o desde la línea de vida hasta la puerta).
   Esto conecta los dos elementos. El cuadrado pequeño representa la puerta.

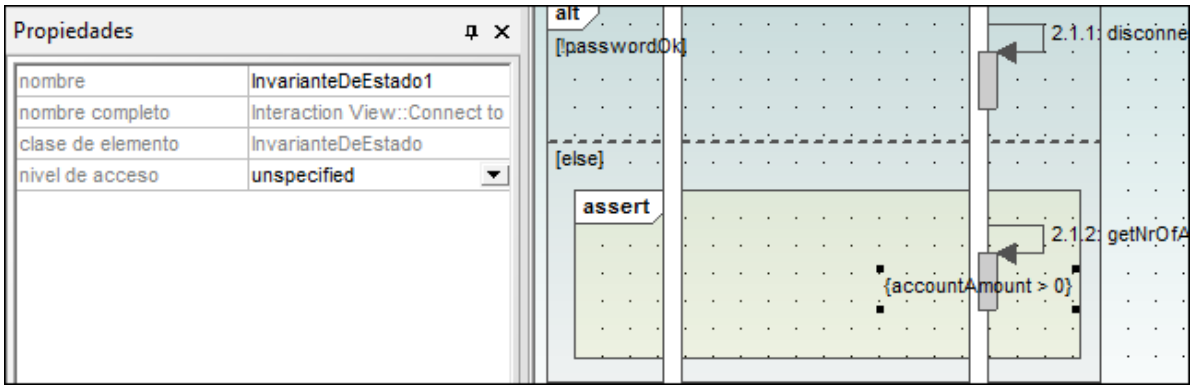


9.1.7.1.5 Invariantes de estado

Una invariante de estado {s} es una condición o restricción aplicada a una línea de vida. Para que exista la línea de vida es obligatorio que la condición se cumpla.

Para definir una InvarianteDeEstado:

1. Haga clic en el icono **InvarianteDeEstado** de la barra de herramientas y después en la línea de vida o en la activación de objetos.
2. Inserte la condición/restricción que desea aplicar (p. ej. accountAmount > 0) y pulse **Entrar** para confirmar.



9.1.7.1.6 Mensajes

Las líneas de vida envían y reciben mensajes que se representan en forma de flechas etiquetadas. Los mensajes pueden estar numerados de forma secuencial y tener atributos opcionales (como listas de

argumentos, etc.). Los mensajes se presentan de arriba a abajo, es decir, el eje vertical es el componente de tiempo del diagrama de secuencia.

- Una **llamada** es una comunicación síncrona o asíncrona que invoca una operación que permite al control volver al objeto remitente. La flecha de una llamada apunta a la parte superior de la activación que inicia la llamada.
- La **recursión** (o llamada a otra operación del mismo objeto) se representa apilando recuadros de activación (`EspecificacionesDeEjecución`).

Para insertar un mensaje:

1. En la barra de herramientas Diagrama de secuencia haga clic en el icono del mensaje que desea insertar.
2. Haga clic en la línea de vida o cuadro de activación del objeto remitente.
3. Arrastre la línea del mensaje hasta la línea de vida o el cuadro de activación del objeto destinatario. Si el mensaje se puede colocar en una línea de vida, esta se resalta.

- La dirección en la que se arrastra la flecha define la dirección del mensaje. Los mensajes de respuesta pueden apuntar en ambas direcciones.
- Mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo para insertar varios mensajes.
- En los objetos remitentes/destinatarios se crean automáticamente cuadros de activación. Para ajustar el tamaño de los cuadros a mano haga clic en los controladores de tamaño y arrástrelos.
- La secuencia de numeración se actualiza, dependiendo de las opciones de numeración que estén activas.

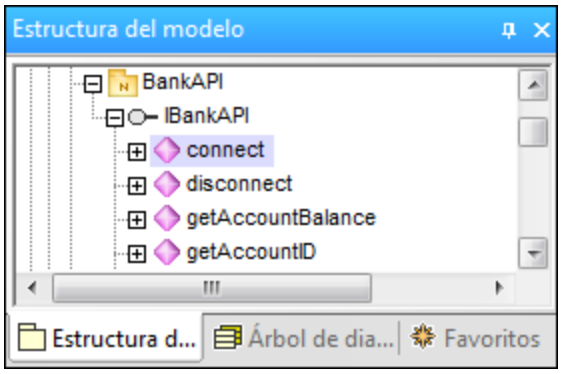
Para eliminar un mensaje:

1. Haga clic en el mensaje.
2. Pulse la tecla **Supr** para eliminar el mensaje del modelo. Para eliminarlo del diagrama solamente, haga clic con el botón derecho en el mensaje y elija **Eliminar solo en el diagrama**. La numeración de los mensajes y los cuadros de activación de los demás objetos se actualizan.

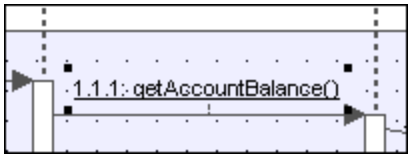
"Ir a la operación" para mensajes de llamada

En los diagramas de secuencia y de comunicación puede buscar las operaciones a las que hacen referencia los mensajes de llamada.

1. Haga clic con el botón derecho en un mensaje de llamada y elija la opción **Ir a la operación**. La operación aparece resaltada en la ventana *Estructura del modelo*.



Nota: los nombres de operaciones estáticas aparecen subrayados.



Para cambiar la posición de mensajes dependientes:

1. Haga clic en el mensaje que quiere mover y arrástrelo hasta su nueva posición.
   Cuando cambia la posición de un mensaje, UModel mueve también los mensajes dependientes relacionados con el mensaje activo.

Para seleccionar varios mensajes utilice **Ctrl+clic**.

Para cambiar la posición de los mensajes uno por uno:

1. Desactive el icono **Activar/desactivar el movimiento de mensajes dependientes** de la barra de herramientas.
2. Haga clic en el mensaje que desea mover y arrástrelo hasta su nueva posición.

En este caso solamente se mueve el mensaje seleccionado. Este mensaje se puede colocar en cualquier posición del eje vertical entre las líneas de vida de los objetos.

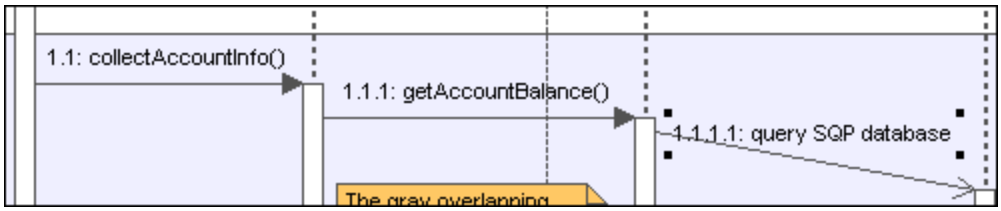
Para crear mensajes de respuesta automáticamente:

1. Active el icono **Activar/desactivar la creación automática de respuestas para mensajes**.
2. Cree un mensaje nuevo entre dos líneas de vida.
   UModel inserta automáticamente un mensaje de respuesta.

Numeración de los mensajes

Puede usar tres tipos de numeración para los mensajes: numeración anidada, numeración sencilla o ninguna numeración.

- **Sin numeración**  **:** este icono elimina la numeración de todos los mensajes.
- **Sencilla**  **:** asigna una secuencia numérica a todos los mensajes de arriba a abajo, es decir, en el orden en el que aparecen a lo largo del eje temporal del diagrama.
- **Anidada**  **:** utiliza la notación decimal, que permite ver la estructura jerárquica de los mensajes del diagrama. La secuencia de numeración es una lista de números secuenciales separada por puntos, seguidos de dos puntos y del nombre del mensaje.



Para seleccionar el tipo de numeración:

Hay dos formas de seleccionar el tipo de numeración:

- Con el icono correspondiente de la barra de herramientas Diagrama de secuencia.
- En la ventana *Estilos*.

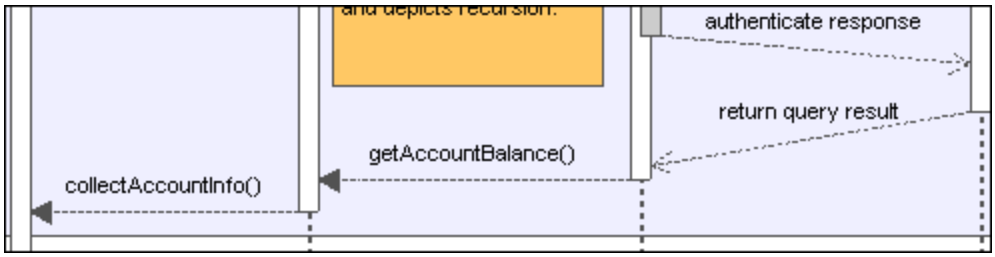
Para seleccionar el tipo de numeración en la ventana Estilos:

1. Haga clic en la ventana *Estilos* y desplácese hasta el campo Numeración de los mensajes.
2. Haga clic en el cuadro combinado y seleccione el tipo de numeración en la lista desplegable. La opción de numeración seleccionada aparece en el diagrama de secuencia automáticamente.

Nota: en ocasiones el tipo de numeración no numera todos los mensajes correctamente si existen ambigüedades. Si esto ocurre, puede solucionar el problema añadiendo mensajes de respuesta.

Mensajes de respuesta

Los mensajes de respuesta se representan con flechas de línea discontinua.

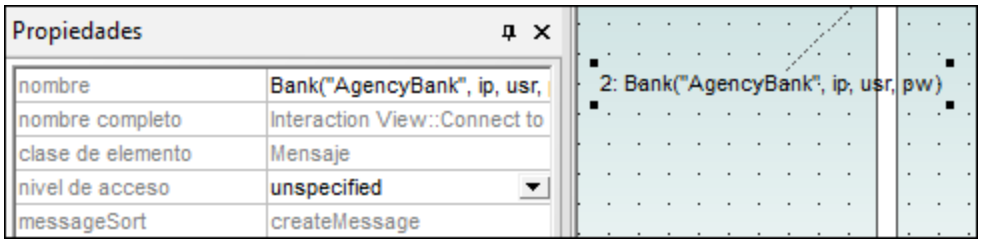


Por lo general, los mensajes de respuesta vienen señalados por la parte inferior del cuadro de activación. Si los cuadros de activación están deshabilitados (panel *Estilos*, Mostrar especificaciones de ejecución=false), entonces se recomienda usar flechas de respuesta para evitar ambigüedades.

Si activa el icono **Activar/desactivar la creación automática de respuestas para mensajes**  , cada vez que cree un mensaje de llamada entre dos líneas de vida/cuadros de activación UModel creará mensajes de respuesta sintácticamente correctos de forma automática.

Crear objetos con mensajes

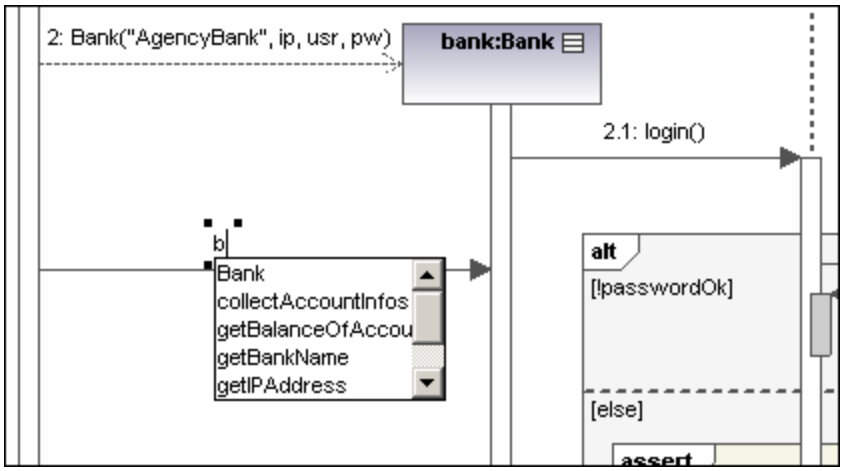
1. Los mensajes pueden crear objetos nuevos. Esto se hace con el icono **Mensaje (Creación)**  .
2. Arrastre la flecha del mensaje hasta la línea de vida de un objeto para crear ese objeto.
   Este tipo de mensaje termina en la mitad del rectángulo del objeto y a menudo pone el cuadro del objeto en posición vertical.



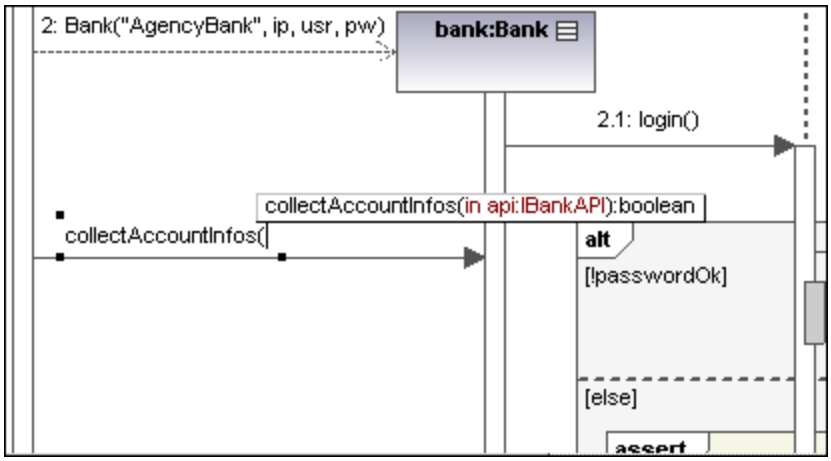
Enviar mensajes a métodos/operaciones de clases de diagramas de secuencia

Tras insertar una clase de la *Estructura del modelo* en un diagrama de secuencia, puede crear un mensaje entre una línea de vida y un método de la clase destinataria (línea de vida). Para ello puede usar la ayuda sintáctica y de las funciones de finalización automática de UModel.

1. Cree un mensaje entre dos líneas de vida (el objeto destinatario debe ser una línea de vida de una clase).
   En cuanto suelte la flecha del mensaje, el nombre del mensaje se resalta automáticamente.
2. Escriba un carácter (p. ej. "b"). Aparece una ventana emergente que enumera los métodos de clase que ya existen.



3. Seleccione una operación de la lista y pulse **Entrar** para confirmar (p. ej. collectAccountInfos).
4. Pulse la barra espaciadora y después la tecla **Entrar** para seleccionar el paréntesis que sugiere automáticamente UModel.
   Ahora aparece una ventana de ayuda sintáctica, que le ayuda a insertar correctamente el parámetro.



Crear operaciones en clases referenciadas

Si activa el icono **Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación**  , cada vez que cree un mensaje y escriba un nombre (p. ej. miOperación()) UModel creará automáticamente la clase correspondiente en la clase referenciada.

Nota: solamente se pueden crear operaciones automáticamente cuando la línea de vida hace referencia a una clase, a una interfaz...

Iconos de la barra de herramientas para trabajar con mensajes

 **Mensaje (Llamada)**

 **Mensaje (Respuesta)**

 **Mensaje (Creación)**

 **Mensaje (Destrucción)**

 **Mensaje asíncrono (Llamada)**

 **Mensaje asíncrono (Respuesta)**

 **Mensaje asíncrono (Destrucción)**

 **Activar/desactivar el movimiento de mensajes dependientes**

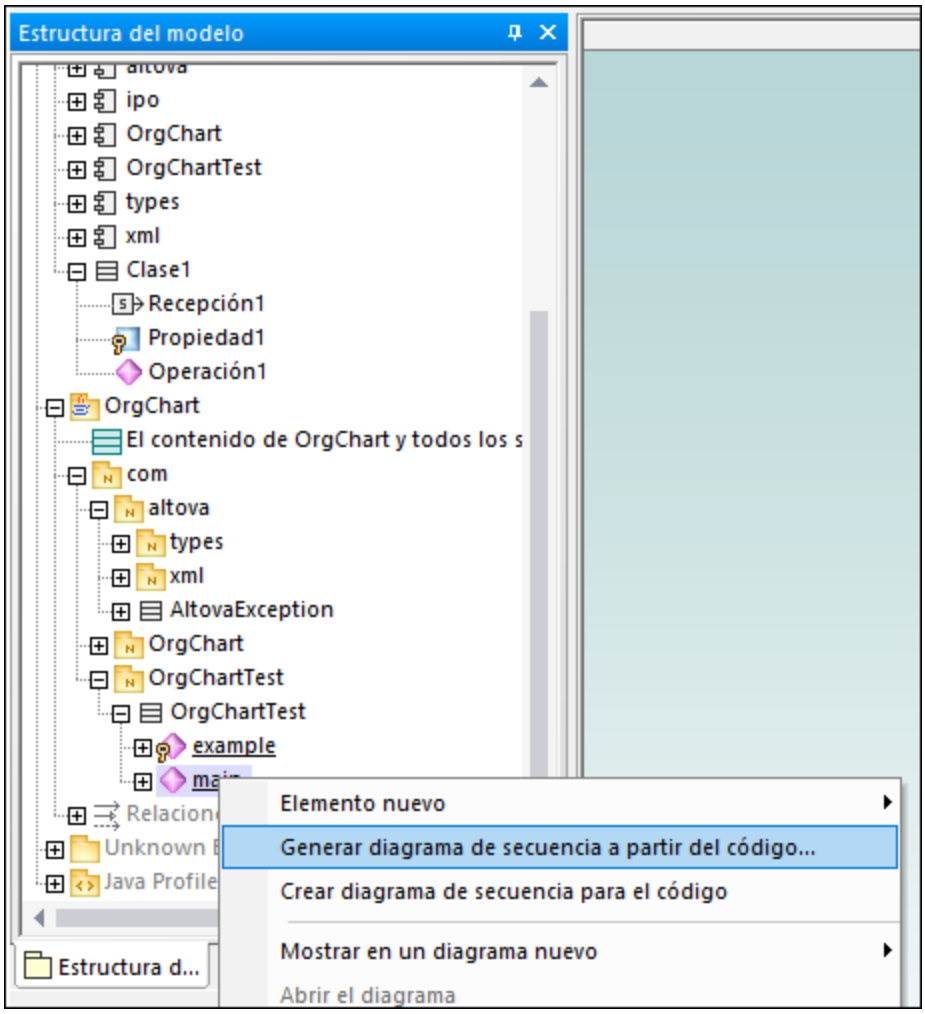
 **Activar/desactivar la creación automática de respuestas para mensajes**

 **Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación**

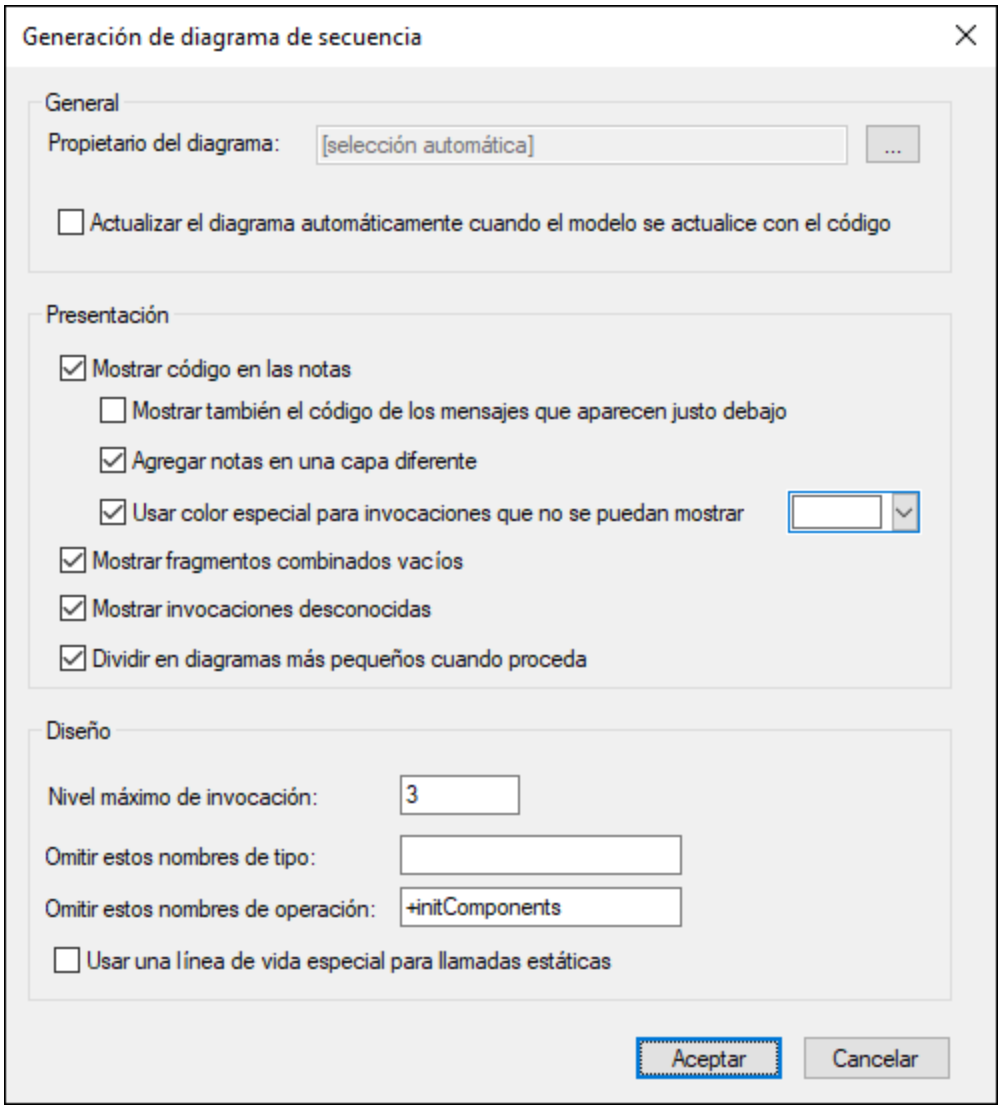
9.1.7.2 Generar diagramas de secuencia a partir de código fuente

Con el siguiente ejemplo puede aprender a crear un diagrama de secuencia a partir de un método. El proyecto que contiene este método proviene de aplicar ingeniería inversa al código fuente Java. Puede encontrar este código fuente Java en la ruta: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\OrgChart.zip**. Primero, descomprima el archivo **OrgChart.zip** (haga clic con el botón derecho en el explorador de Windows y seleccione **Extraer todos**).

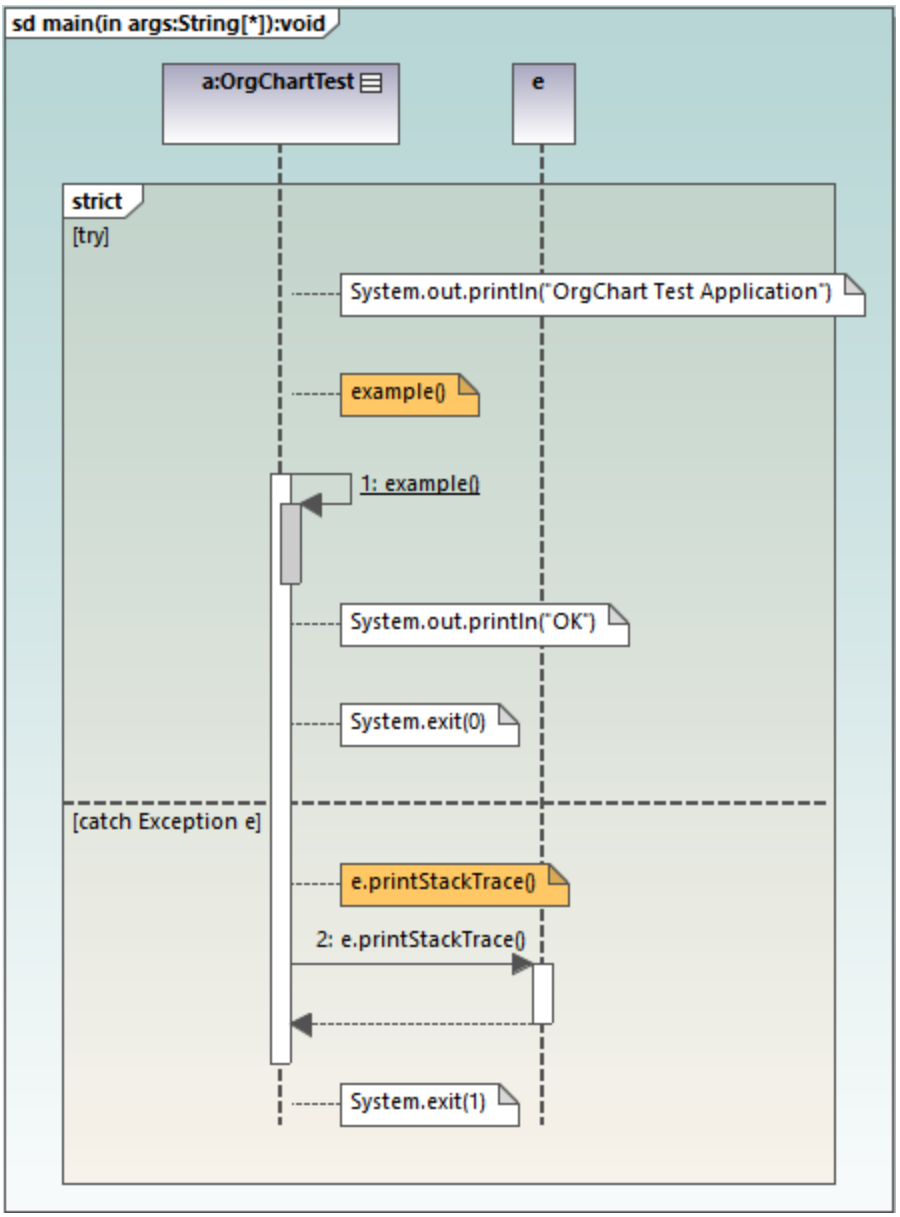
1. En el menú **Proyecto** haga clic en **Importar directorio de código fuente** y seleccione el directorio en el que descomprimió los archivos en el paso anterior.
2. Siga las instrucciones del asistente para importar el código fuente como proyecto de Java. Para más información sobre este paso consulte el apartado Ingeniería inversa (del código al modelo).
3. Una vez importado el código, haga clic con el botón derecho en el método `main` de la clase `OrgChartTest` en la *Estructura del modelo*. Ahora elija el comando **Generar diagrama de secuencia a partir del código** en el menú contextual.



Esto abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede configurar la generación.



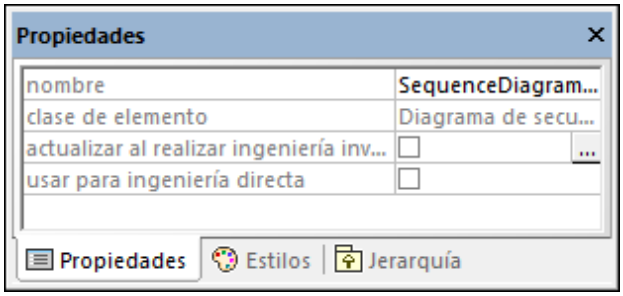
4. Seleccione las opciones de presentación y diseño y después haga clic en **Aceptar**.
   La imagen siguiente muestra el diagrama de secuencia que se genera con las opciones seleccionadas en la imagen anterior.

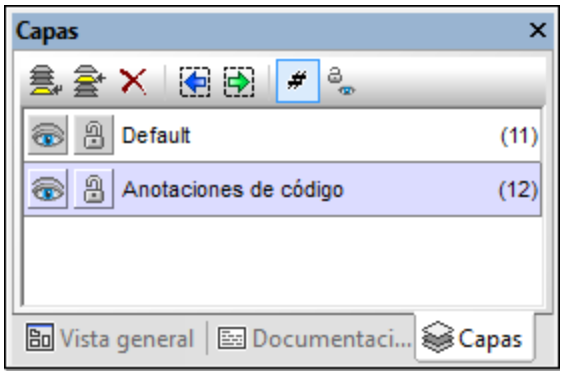


Opciones de los diagramas de secuencia

En esta tabla se explican las opciones de generación de los diagramas de secuencia.

Opción	Objetivo
Propietario del diagrama	Puede elegir esta opción al generar un diagrama por primera vez. En el caso de diagramas que ya existen, esta información es de solo lectura. Haga clic en el botón **Examinar** y navegue hasta el paquete propietario del diagrama. De lo contrario, la

Opción	Objetivo
	opción **[autoselect]** coloca el diagrama en el paquete predeterminado.
Actualizar el diagrama automáticamente cuando el modelo se actualice con el código	Al aplicar ingeniería inversa (del código al modelo), los diagramas de secuencia se vuelven a generar automáticamente en el modelo, siempre y cuando haya marcado esta opción al generar el diagrama por primera vez. En el caso de diagramas que ya existen: 1. Seleccione el diagrama de secuencia en la Estructura del modelo o en el Árbol de diagramas. 2. En la ventana Propiedades marque la casilla *actualizar al realizar ingeniería inversa*.  Si marca la casilla *usar para ingeniería directa*, la sincronización del modelo al código generará código basado en el diagrama de secuencia cuando aplique ingeniería directa (del modelo al código); véase también Generar código a partir de diagramas de secuencia. Si no marca ninguna de estas casillas es probable que este diagrama forme parte de un diagrama mayor o puede que haya creado el diagrama a partir de una operación a la que no ha aplicado ingeniería inversa.
Mostrar código en las notas	Marque esta casilla para generar el diagrama con notas que contienen código de programa.
Mostrar también el código de los mensajes que aparecen justo debajo	Incluso si es posible mostrar un extracto de código como mensaje UML en el diagrama, si marca esta opción el código de ese mensaje se mostrará también como una nota.
Agregar notas en una capa diferente	Asigna notas de código a la capa "Anotaciones de código".

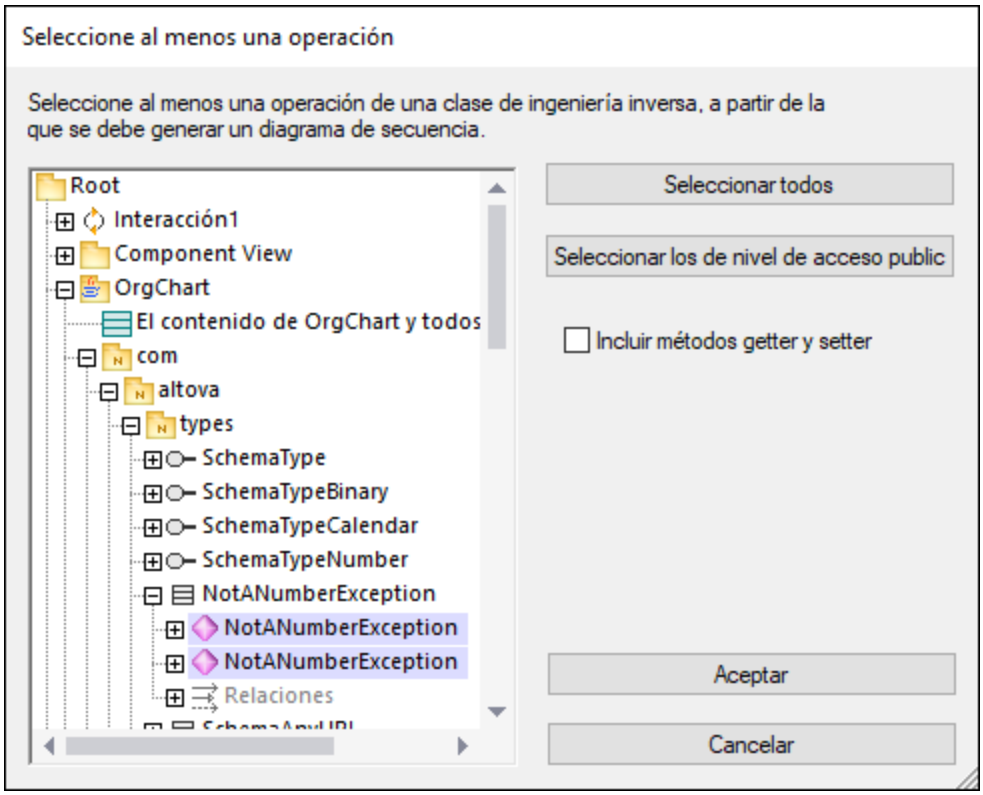
Opción	Objetivo
	
Usar color especial para invocaciones que no se puedan mostrar	Asigna el color que elija a las invocaciones que no se pueden mostrar.
Mostrar fragmentos combinados vacíos	Mantiene los bloques Fragmentos combinados en el diagrama, incluso aunque no tengan contenido.
Mostrar invocaciones desconocidas	Si marca esta opción se muestran los mensajes de las operaciones o de los constructores que no se pueden resolver (es decir, que no se encuentran en el modelo).
Dividir en diagramas más pequeños cuando proceda	Divide automáticamente los diagramas de secuencia en subdiagramas más pequeños y genera hipervínculos entre ellos para facilitar la navegación.
Nivel máximo de invocación	Define el nivel máximo de las llamadas en ese diagrama. Por ejemplo, si `method1()` llama a `method2()`, que llama a `method3()` y el nivel máximo de invocación es **2**, entonces se muestra `method2` pero no `method3`.
Omitir estos nombres de tipo	Permite establecer una lista de valores separados por comas de los tipos que no deben aparecer en el diagrama de secuencia al generarlo.
Omitir estos nombres de operación	Permite establecer una lista de valores separados por comas de las operaciones que no deben aparecer en el diagrama de secuencia generado. Las operaciones cuyos nombres añada a la lista se ignorarán. Si escribe un símbolo "+" antes del nombre de la operación (por ejemplo, **+InitComponent**), esa operación aparece en el diagrama pero sin contenido.
Usar una línea de vida especial para llamadas estáticas	Si hay llamadas estáticas a métodos y si ya hay una instancia de ese objeto en el diagrama, los mensajes suelen usar esa línea de vida. Si activa esta opción, el generador de diagramas usa una línea de vida

Opción	Objetivo
	distinta para las llamadas estáticas a métodos para ese clasificador.

9.1.7.2.1 Generar varios diagramas de secuencia a partir de propiedades

En Umodel también puede crear modelos de diagramas de secuencia a partir de operaciones:

1. Seleccione la opción de menú **Proyecto | Generar diagramas de secuencia a partir del código**. Aparece el cuadro de diálogo "Seleccione al menos una operación".



2. Seleccione las operaciones para las que desea generar un diagrama de secuencia y haga clic en **Aceptar** (si quiere puede usar los botones **Seleccionar todos** y **Seleccionar los de nivel de acceso public**).
3. Otra opción es marcar la casilla *Incluir métodos getter y setter* para generar diagramas de secuencia para los getter y setter de C#/VB.NET.
4. Al hacer clic en **Aceptar** se abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede elegir las opciones de generación.
5. Haga clic en **Aceptar** para generar los diagramas de secuencia. Por cada operación seleccionada se genera un diagrama de secuencia distinto.

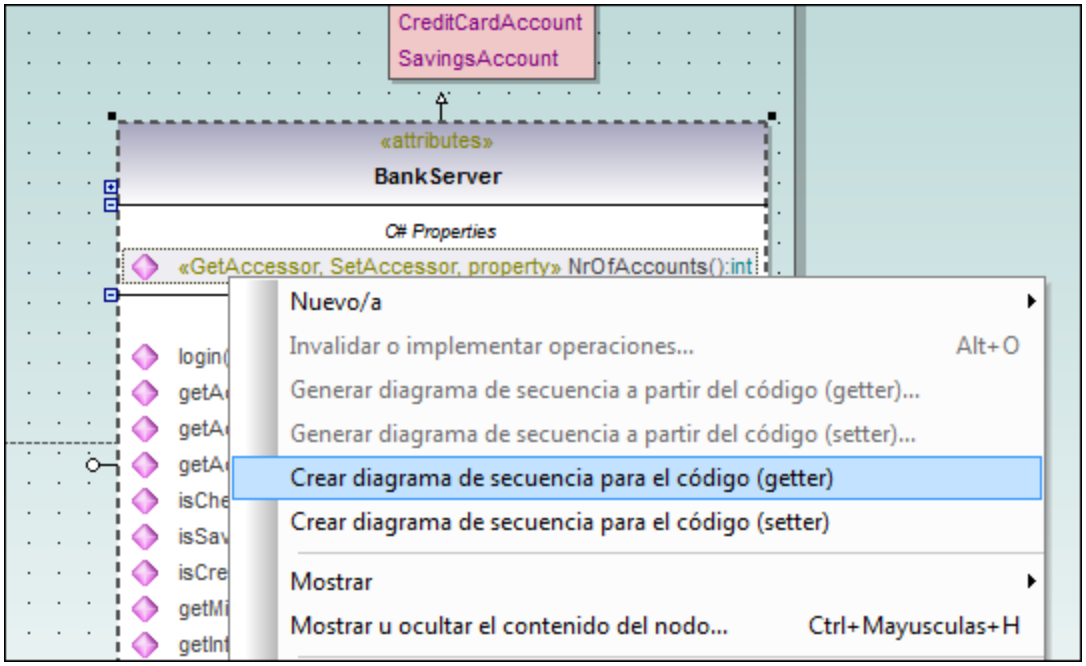
Si su proyecto es grande, es probable que crear varios diagramas lleve bastante tiempo. Tenga en cuenta que UModel solamente abre automáticamente los primeros 10 diagramas; el resto se generan pero no sin

abrirse.

9.1.7.2.2 Generar diagramas de secuencia a partir de propiedades getter/setter

También puede generar diagramas de secuencia a partir de las propiedades getter/setter (en C#, VB .NET):

1. Haga clic con el botón derecho en una operación que tenga el estereotipo GetAccessor/SetAccessor.



2. Seleccione la opción del menú contextual correspondiente (p. ej. **Generar diagrama de secuencia a partir del código... (getter/setter)**).
   Esto abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede configurar las Opciones del diagrama de secuencia.
3. Haga clic en **Aceptar** para generar el diagrama de secuencia.

9.1.7.3 Generar código a partir de diagramas de secuencia

UModel puede crear código a partir de un diagrama de secuencia que esté vinculado a una operación o a varias.

A partir de diagramas de secuencia puede generar código para:

- VB.NET, C# y Java.
- UModel y la edición Eclipse y Visual Studio de UModel.
- las tres ediciones de UModel.

Para generar código a partir de diagramas de secuencia:

Hay dos maneras de hacerlo:

- Comenzando con una operación creada con ingeniería inversa (consulte el apartado Generar diagramas de secuencia a partir de código fuente).
- Creando desde cero un diagrama de secuencia **nuevo** que esté vinculado a una operación (haciendo clic con el botón derecho en la *Estructura del modelo* y seleccionando **Crear diagrama de secuencia para el código**).

> Si usa como base un diagrama de secuencia al que ha aplicado ingeniería inversa, asegúrese de que selecciona la opción *Mostrar código en las notas* al aplicar la ingeniería inversa al código para no perder código al iniciar de nuevo el proceso de ingeniería. Esto se debe a que UML no puede mostrar todas las características de los lenguajes VB.NET, Java y C# en el diagrama de secuencia y las características que no puede mostrar se presentan como notas.

Para agregar texto sin formato como código durante la creación de diagramas de secuencia:

1. Anexe una nota a una línea de vida del diagrama de secuencia.
2. Escriba el código que se debe escribir en el código fuente final.
   Marque la casilla Es código (panel *Propiedades*) de la nota para poder acceder a ella.

Para ver un ejemplo consulte el apartado Agregar código a los diagramas de secuencia.

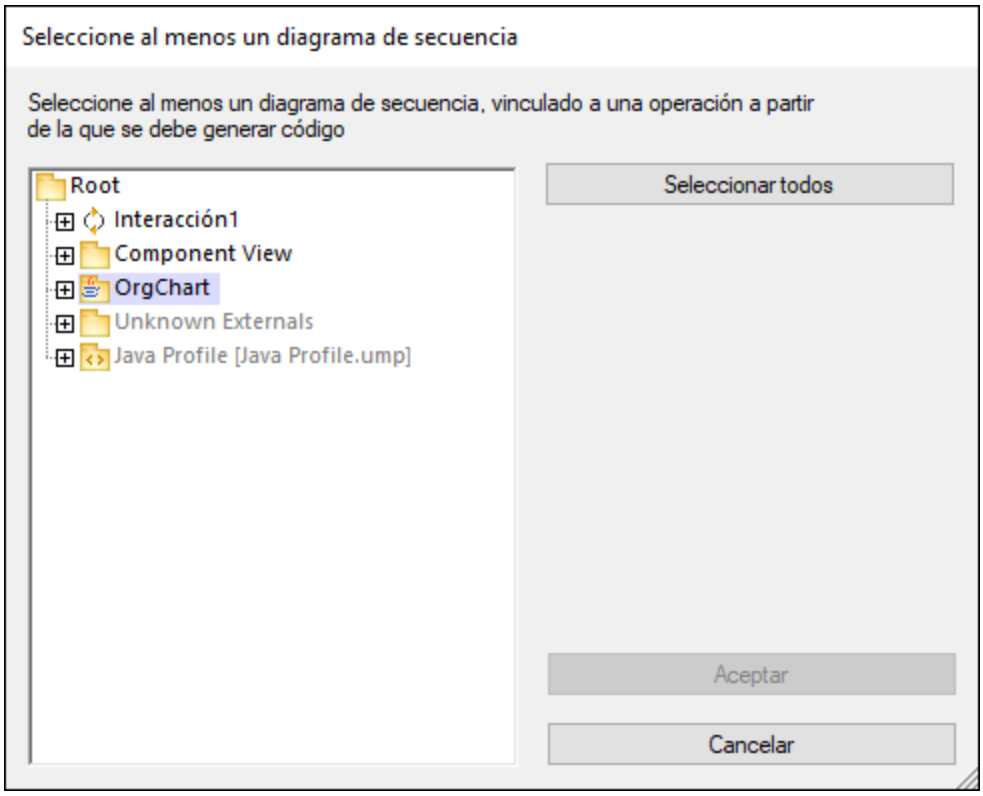
Si quiere que un diagrama de secuencia se utilice automáticamente para ingeniería de código cada vez que se inicie la ingeniería de código:

- Seleccione el diagrama en la Estructura del modelo o en el Árbol de diagramas.
- Active la casilla *Usar para ingeniería directa* de la ventana *Propiedades*.

Cuando se crea código por ingeniería directa a partir de un diagrama de secuencia, siempre se pierde código antiguo porque lo sobrescribe el código nuevo.

Para generar código usando el menú Proyecto:

1. Seleccione la opción de menú **Proyecto | Generar código a partir de diagramas de secuencia**.
   Aparece un diálogo donde debe seleccionar el diagrama de secuencia.

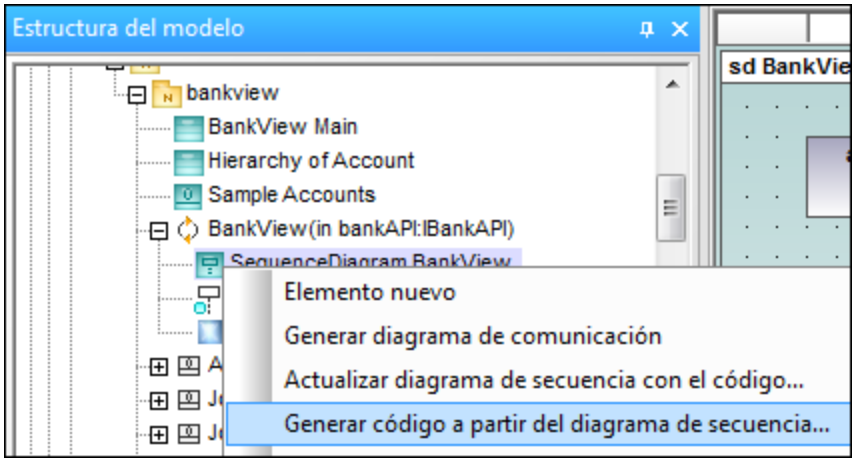


Con el botón **Seleccionar todos** se seleccionan todos los diagramas de secuencia del proyecto de UModel.

2. Haga clic en **Aceptar** para generar el código.
   La ventana Mensajes muestra el estado del proceso de generación de código.

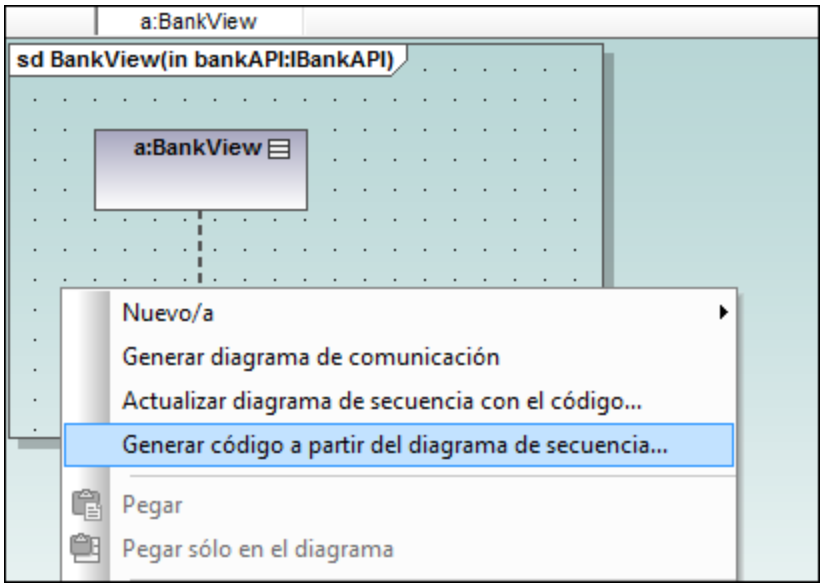
Para generar código usando el panel Estructura del modelo:

- Haga clic con el botón derecho en un diagrama de secuencia y elija **Generar código a partir del diagrama de secuencia** en el menú contextual.



Para generar un diagrama de secuencia con código de una operación:

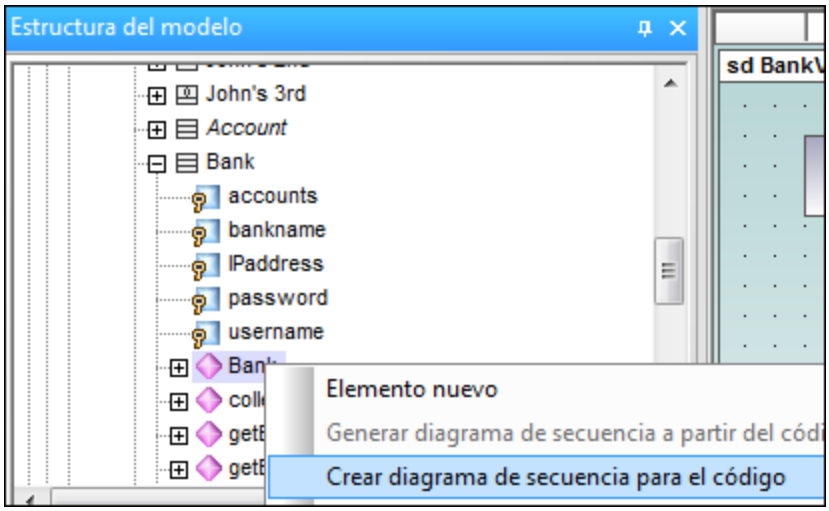
1. Haga clic con el botón derecho en el fondo del diagrama de secuencia que contiene el código de una operación.
2. Elija la opción **Generar código a partir del diagrama de secuencia**.



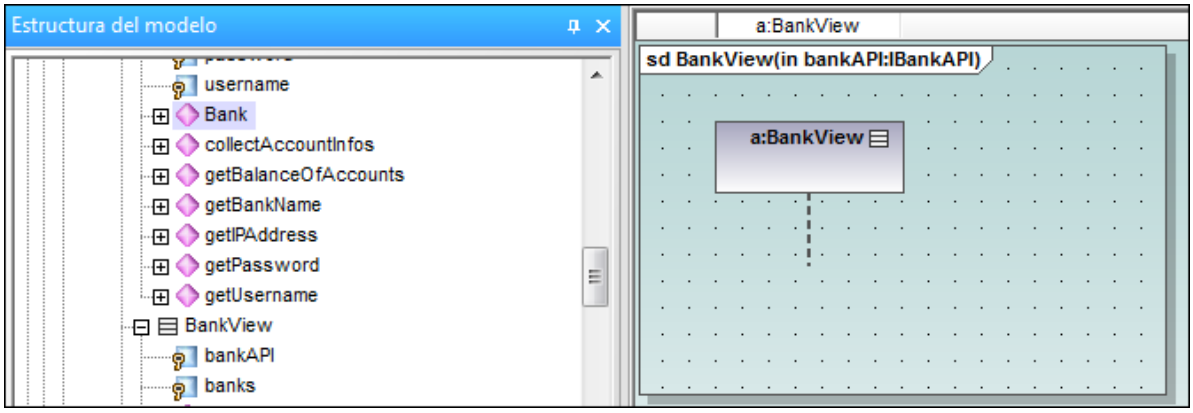
Este comando inicia el proceso de ingeniería directa.

Para crear un diagrama de secuencia para el código (ingeniería):

- En la *Estructura del modelo* haga clic con el botón derecho en una operación y elija la opción **Crear diagrama de secuencia para el código**.



UModel le pregunta si quiere usar el diagrama nuevo para la ingeniería directa.



El resultado es un diagrama de secuencia nuevo que contiene la línea de vida de esa clase.

9.1.7.3.1 Agregar código a diagramas de secuencia

En UModel puede generar código a partir de diagramas de secuencia nuevos y de diagramas de secuencia generados por ingeniería inversa, pero solo si el diagrama está vinculado a la operación principal.

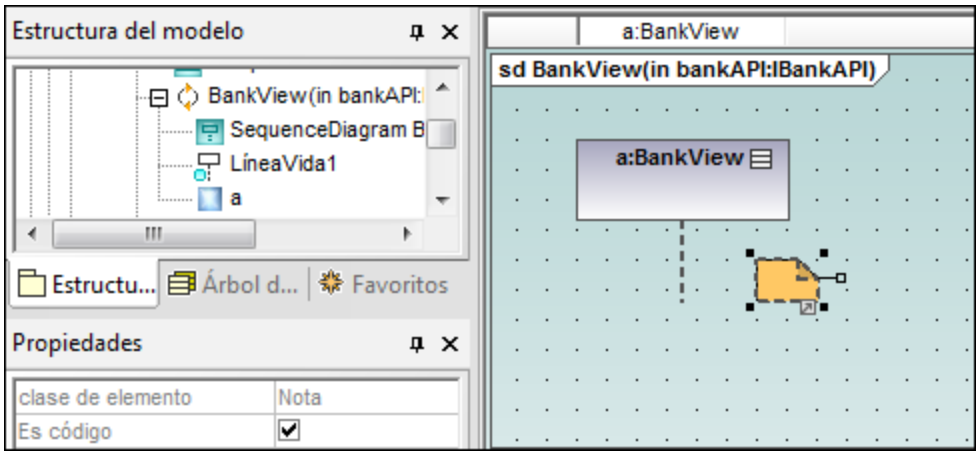
Cuando se aplica ingeniería inversa a código, los elementos estándar de los diagramas de secuencia (p. ej. los `FragmentosCombinados`) se asignan a los elementos del código (p. ej. instrucciones `if`, bucles, etc.).

Para las instrucciones de programación que no tengan elementos equivalentes en el diagrama de secuencia (p. ej. `i = i+1`), UModel utiliza las notas del código y añade código a los diagramas. Estas notas se deben vincular a la línea de vida.

Recuerde que UModel no revisa ni analiza estos fragmentos de código. Por eso es importante comprobar que los fragmentos de código son correctos y se podrán compilar.

Para agregar código a un diagrama de secuencia

1. Haga clic en el icono **Nota**  y después en el elemento de modelado donde desea insertar la nota (p. ej. `FragmentoCombinado`).
2 Escriba el fragmento de código dentro de la nota (p. ej. return).
3. Haga clic en el controlador Enlace de nota de la nota que acaba de insertar y arrastre el cursor hasta la línea de vida.
4. Marque la casilla Es código en la ventana *Propiedades* para incluir este fragmento de código cuando UModel genere código.



Cuando seleccione una nota de un diagrama de secuencia que se **pueda** usar para generación de código, la propiedad Es código aparece en la ventana *Propiedades*. Esta propiedad permite alternar entre notas normales y corrientes y notas para generación de código.

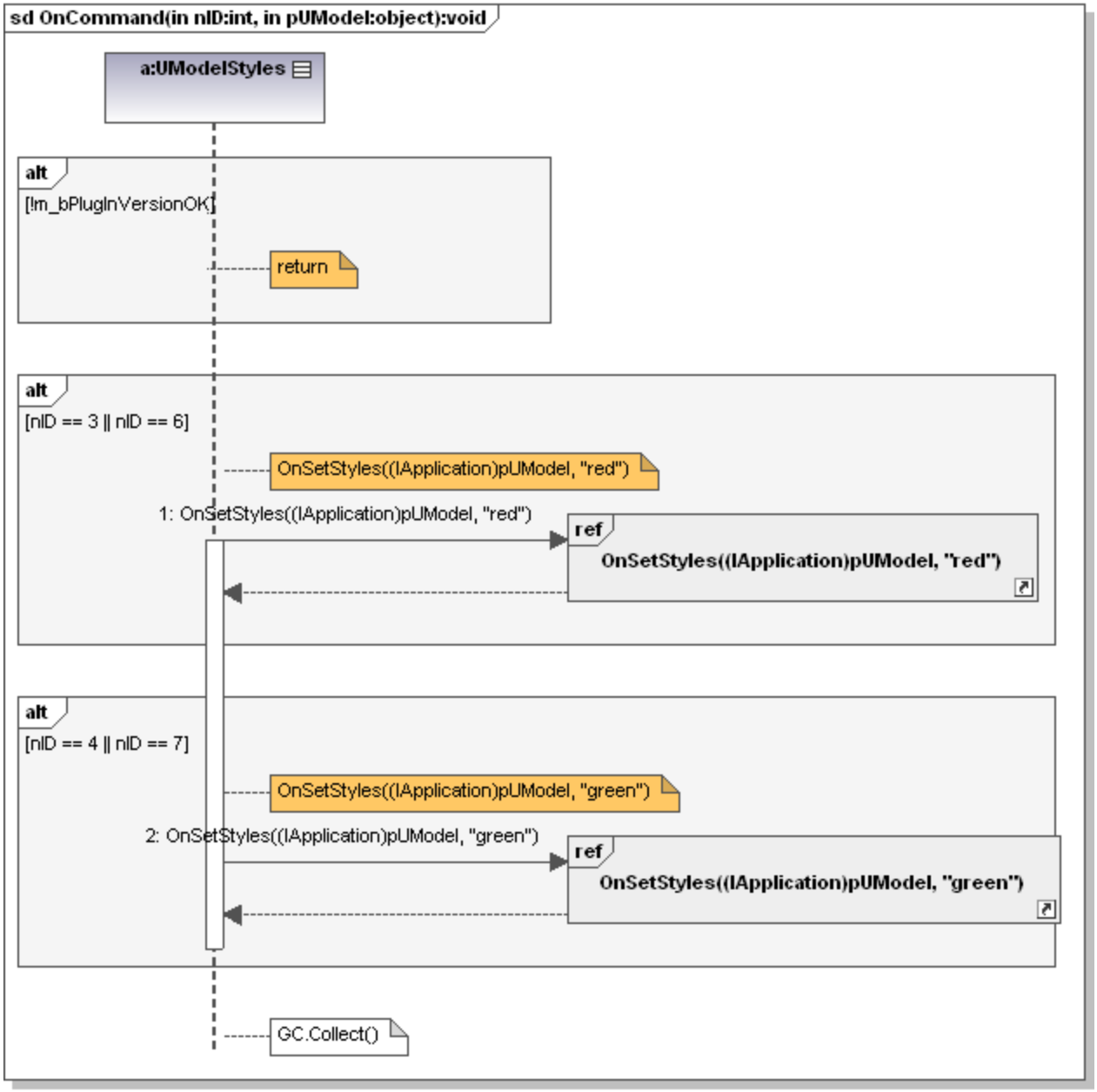
Notas normales y corrientes: 

Notas para generación de código: 

(la pestaña de la esquina superior derecha es más oscura)

Las actualizaciones de código tienen lugar automáticamente en cada proceso de ingeniería directa si está activa la casilla Usar para ingeniería directa. Si se realizaron cambios en el diagrama de secuencia, el código de la operación se sobrescribe siempre.

El diagrama de secuencia que aparece más abajo se generó haciendo clic con el botón derecho en la operación `OnCommand` y seleccionando la opción **Generar diagrama de secuencia a partir del código**. El código C# de este ejemplo está disponible en la carpeta c:\Usuarios\<usuario>\Documentos\Altova\UModel2017\UModelExamples\IDEPlugIn\Styles. Utilice la opción de menú **Proyecto | Importar proyecto de código fuente** para importar el proyecto.



El código que aparece a continuación se generó a partir del diagrama de secuencia.

```
Public void OnCommand(int nID, object pUModel)
{
   //Generated by UModel. This code will be overwritten when you re-run code generation.

   if (!m_bPlugINVersionOK)
   {
   return;
   }
```

```
    if (nID == 3 || nID == 6)
    {
    OnSetStyles((IApplication)pUModel, "red");
    }

    if (nID == 4 || nID == 7)
    {
    OnSetStyles((IApplication)pUModel, "green");
    }
    GC.Collect();

}
```

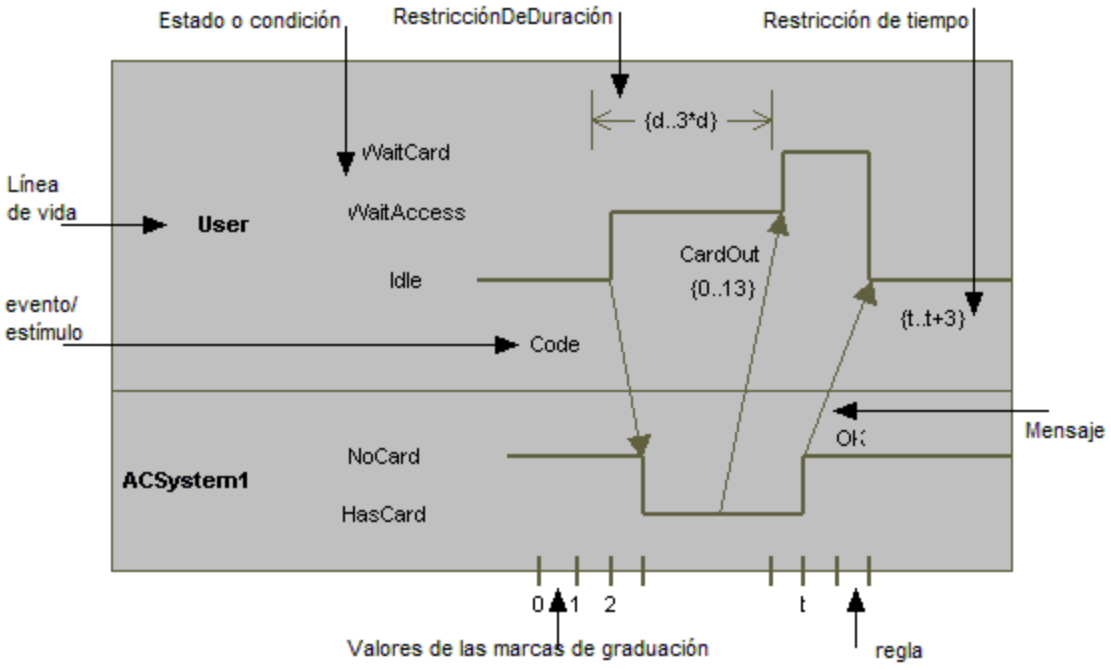
9.1.8 Diagrama de ciclo de vida

Sitio web de Altova: Diagramas de ciclo de vida UML

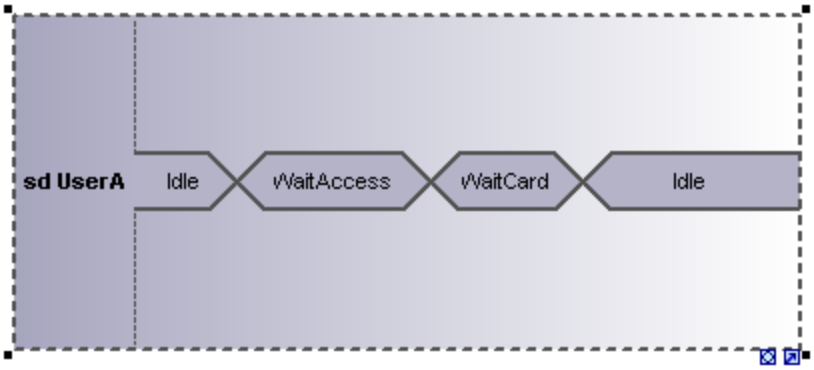
Los diagramas de ciclo de vida modelan los cambios de estado o la condición de los objetos que interactúan entre sí a lo largo de un período de tiempo. Los estados o condiciones se representan como escalas de tiempo que responden a eventos de mensaje y las líneas de vida representan instancias de clasificador y roles clasificador.

Los diagramas de ciclo de vida son un tipo especial de diagrama de secuencia. La diferencia es que los ejes están invertidos, es decir, el tiempo aumenta de izquierda a derecha, y las líneas de vida aparecen por separado en compartimentos apilados verticalmente.

Además, este tipo de diagramas suelen utilizarse para el diseño de software integrado o de sistemas en tiempo real.



Hay dos tipos de diagramas de ciclo de vida: los que contienen la escala de tiempo del estado/de la condición (*imagen anterior*) y los que muestran el ciclo de vida general (*imagen siguiente*).



9.1.8.1 Insertar elementos

Insertar elementos con los iconos de la barra de herramientas

1. Haga clic en un icono de la barra de herramientas Diagrama de ciclo de vida.

2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Para arrastrar elementos desde la Estructura del modelo hasta el diagrama de ciclo de vida

1. En la *Estructura del modelo* busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de ciclo de vida.

9.1.8.2 Línea de vida

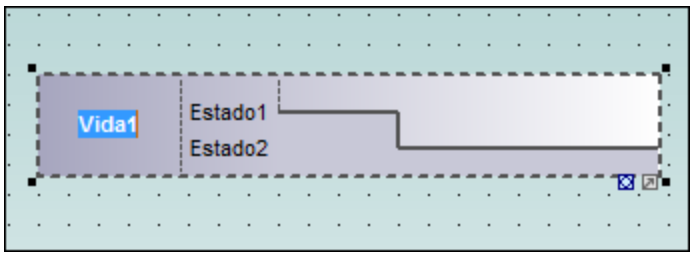
La **línea de vida** es un participante de la interacción y tiene dos representaciones distintas:

1. un **Estado/Condición** 
2. un **Valor general** 

Pulse **Ctrl+Entrar** para crear una línea nueva en el nombre de la línea de vida.

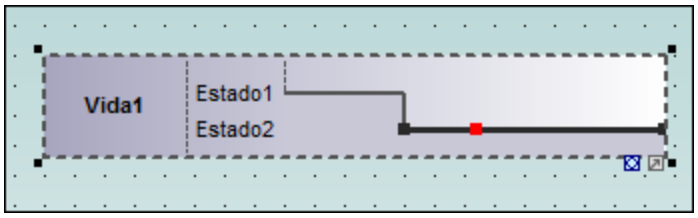
Para insertar un Estado/Condición (InvarianteDeEstado) y definir cambios de estado:

1. Haga clic en el icono **Línea de vida (Estado o Condición)**  de la barra de herramientas y después haga clic en el área de trabajo del diagrama.



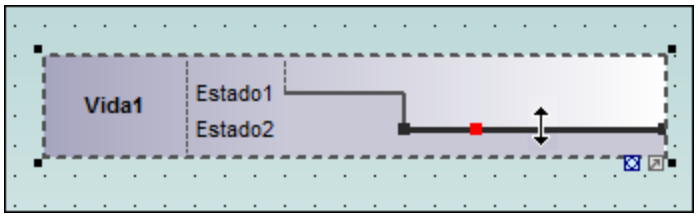
2. Escriba el nombre de la línea de vida o utilice el nombre predeterminado **`Líneadevida1`**.
3. Haga clic en una sección de la línea de tiempo para seleccionarla.
4. Haga clic en la posición de la línea de tiempo donde quiere que se produzca el cambio de estado.

   En este momento aparece una flecha con dos puntas.

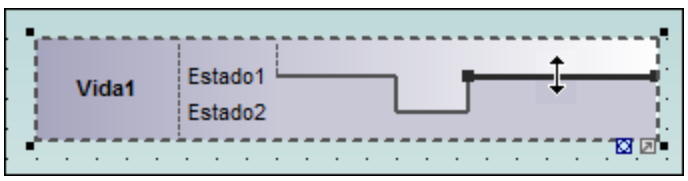


   Y además aparece un recuadro rojo en la posición donde hizo clic, que divide la línea por ese punto.

5. Ponga el cursor en la parte derecha de la línea y arrastre la línea hacia arriba.



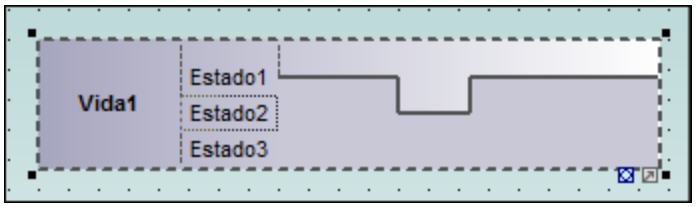
   No olvide que solo puede mover líneas entre estados de la línea de vida actual.



   Puede definir un número ilimitado de cambios de estado en una línea de vida. El recuadro rojo de la línea desaparece al hacer clic en otra parte del diagrama.

Para añadir un estado nuevo a la línea de vida:

1. Haga clic con el botón derecho en la línea de vida y seleccione **Nuevo/a | Estado o Condición (InvarianteDeEstado)**.
   El estado nuevo `Estado3` se añade a la línea de vida.



Para cambiar la posición de un estado (dentro de una línea de vida):

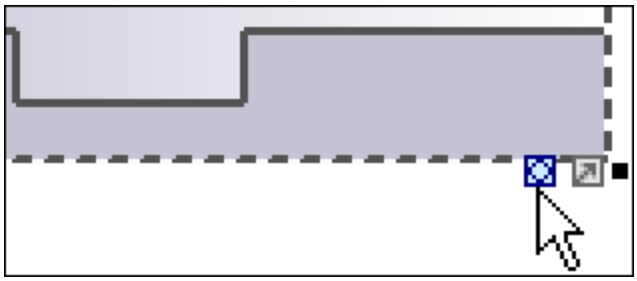
1. Haga clic en la etiqueta del estado.
2. Arrástrela hasta la posición nueva.

Para eliminar un estado de una línea de vida:

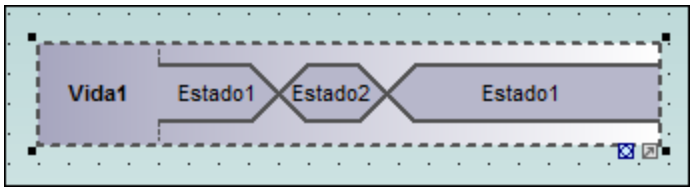
1. Haga clic en la etiqueta del estado y pulse la tecla **Supr**. También puede hacer clic con el botón derecho en el estado y seleccionar el comando **Eliminar**.

Para cambiar de tipo de diagrama de ciclo de vida:

1. Haga clic en el icono **Alternar estilo de notación** que aparece en la esquina inferior derecha de la línea de vida.



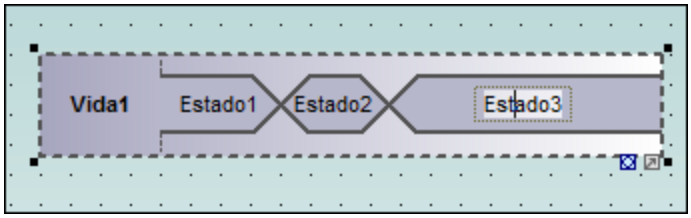
Ahora la línea de vida se presenta con su `Valor general`. Cada punto donde se cruzan las líneas es un cambio de estado/valor.



Nota: recuerde que al hacer clic en el icono **Línea de vida (Valor general)** se inserta una línea de vida como la de la imagen anterior. Puede cambiar a la otra presentación cuando quiera.

Para añadir un estado nuevo a la línea de vida Valor general:

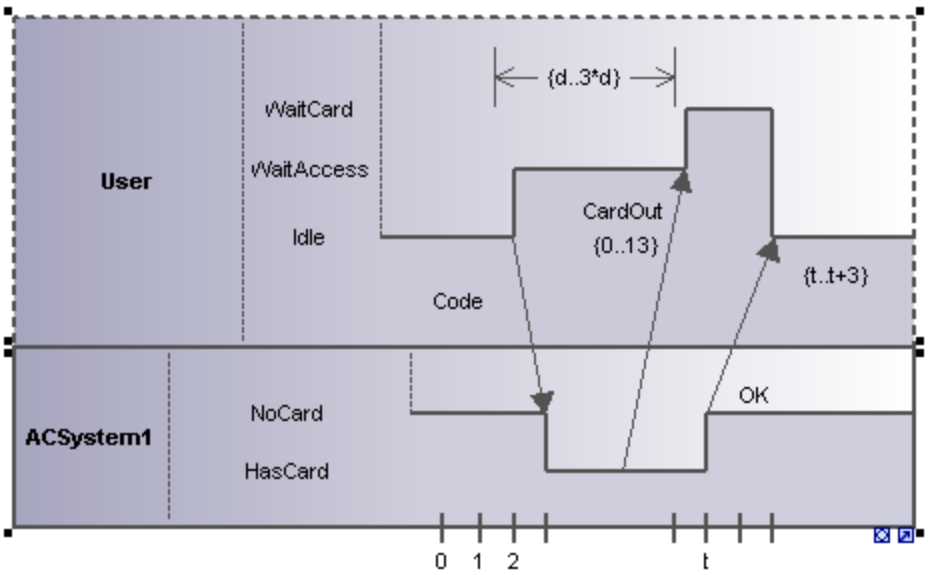
1. Haga clic con el botón derecho en la línea de vida y seleccione **Nuevo/a | Estado o condición (InvarianteDeEstado)**.
2. Edite el nombre del estado nuevo y pulse **Entrar** para confirmar.



Se añade un estado nuevo en la línea de vida.

Agrupar las líneas de vida

Si apila las líneas de vida, UModel las reorganiza automáticamente de forma correcta y conserva las marcas de graduación disponibles hasta ese momento. También puede crear mensajes entre las líneas de vida. Para ello arrastre el objeto de mensaje correspondiente hasta la posición deseada.

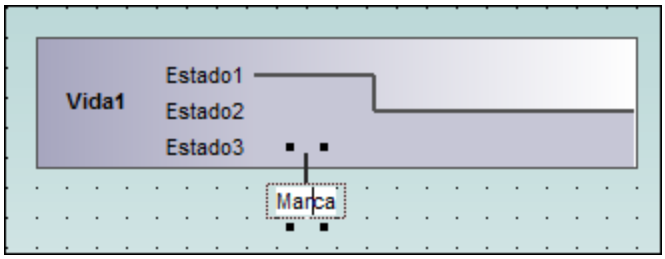


9.1.8.3 Marca de graduación

El icono **MarcaDeGraduación** permite insertar las marcas de graduación de una escala de tiempo en la línea de vida.

Para insertar una marca de graduación:

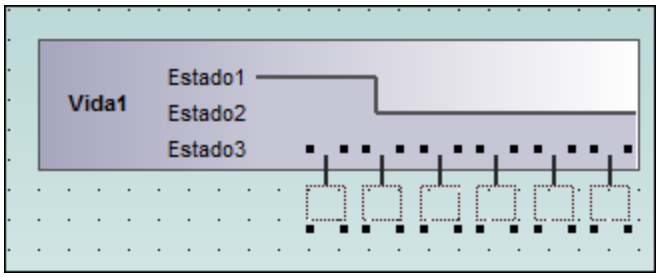
1. Haga clic en el icono de la marca de graduación y después en la línea de vida para insertarla.



2. Para insertar varias marcas, pulse la tecla **Ctrl** mientras hace clic en la línea de vida tantas veces como sea necesario.
3. Escriba el nombre de la marca.
   Si quiere cambiar la posición de una marca de graduación, simplemente arrástrela hasta la posición nueva.

Para espaciar las marcas de graduación uniformemente:

1. Seleccione todas las marcas de graduación.
2. Haga clic en el icono **Espaciar uniformemente en horizontal** de la barra de herramientas.

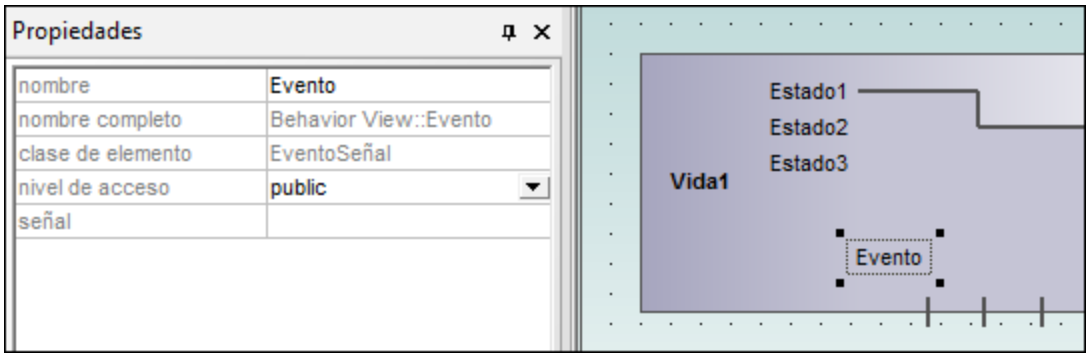


9.1.8.4 Evento/estímulo

El icono **Evento/ Estímulo** sirve para ilustrar el cambio de estado de un objeto causado por el correspondiente evento o estímulo. Los eventos recibidos se anotan para mostrar qué evento provoca el cambio en la condición o en el estado.

Para insertar un evento o estímulo:

1. Haga clic en el icono **Evento o estímulo** y después haga clic en la posición de la escala de tiempo donde tiene lugar el cambio de estado.



2. Escriba el nombre del evento.

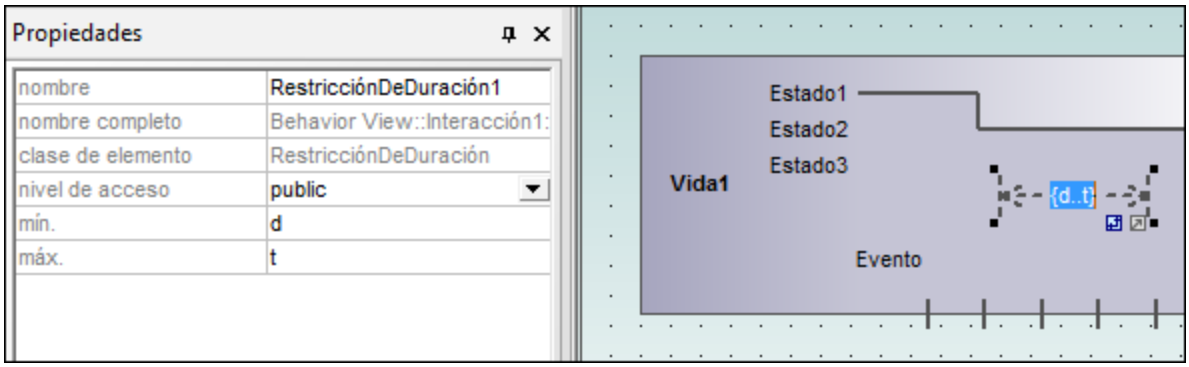
Observe que las propiedades del evento aparece en la ventana *Propiedades*.

9.1.8.5 Restricción de duración

Una **RestricciónDeDuración** define una `EspecificaciónDeValor` que denota el período de tiempo comprendido entre el punto de inicio y el punto final. Por lo general una duración es una expresión que representa el tiempo que puede pasar durante este período.

Para insertar una RestricciónDeDuración:

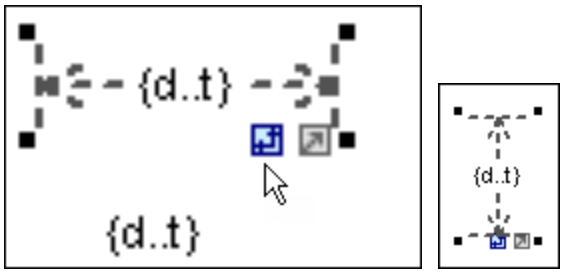
1. Haga clic en el icono **RestricciónDeDuración** y después haga clic en la posición de la línea de vida donde debe aparecer la restricción. El valor mínimo y máximo predeterminado ("`d..t`") aparece automáticamente. Para editar estos valores haga doble clic en la restricción o edite los valores en la ventana *Propiedades*.



2. Si quiere, use los controladores para cambiar el objeto de tamaño.

Para cambiar la orientación de la RestricciónDeDuración:

1. Haga clic en el icono rotación para poner la restricción en vertical.

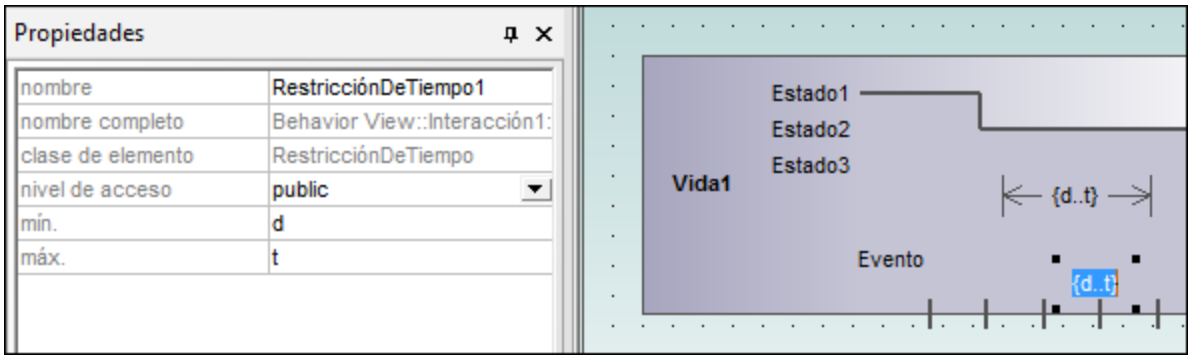


9.1.8.6 Restricción de tiempo

Una **RestricciónDeTiempo**  suele representarse como una asociación gráfica entre un `IntervaloDeTiempo` y la construcción que limita. Lo normal es que sea una asociación gráfica entre un evento y un intervalo de tiempo.

Para insertar una restricción de tiempo:

1. Haga clic en el icono **RestricciónDeTiempo**  en la barra de herramientas y después haga clic en la posición de la línea de vida donde debe aparecer la restricción.



El valor mínimo y máximo predeterminado ("`d..t`") aparece automáticamente. Para editar estos valores haga doble clic en la restricción o edite los valores en la ventana *Propiedades*.

9.1.8.7 Mensaje

Un mensaje es un elemento de modelado que define un tipo concreto de comunicación dentro de una interacción. Una comunicación puede lanzar una señal, invocar una operación, crear o destruir una instancia, etc. El mensaje especifica el tipo de comunicación definida por la `EspecificaciónDeEjecución` emisora, así como el remitente y el destinatario.

Use los siguientes botones de la barra de herramientas para añadir tipos de mensaje específicos:

 **Mensaje (Llamada)**

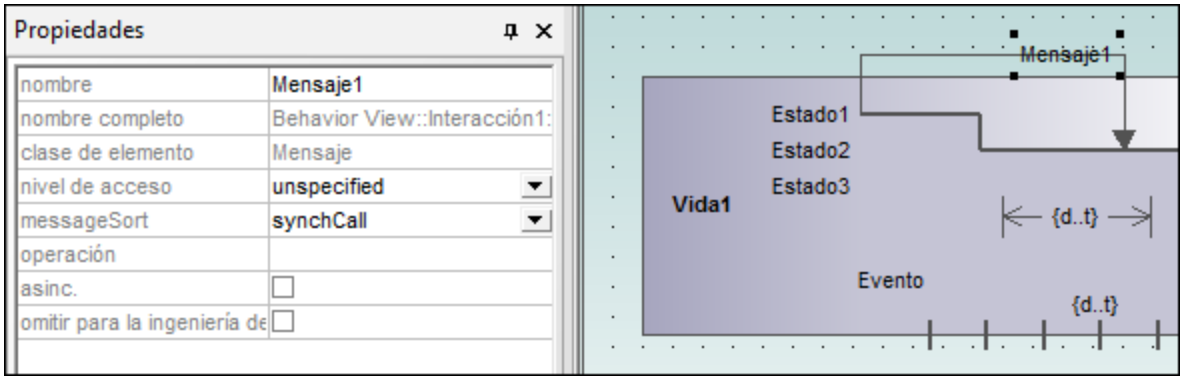
Mensaje (Respuesta)

Mensaje asincrónico (Llamada)

Los mensajes se transmiten entre la escala de tiempo remitente y la escala de tiempo destinataria y se representan en forma de flechas con etiqueta:

Para insertar un mensaje:

1. Haga clic en el icono del mensaje que desea insertar en la barra de herramientas.
2. Haga clic en el objeto remitente.
3. Arrastre la línea del mensaje y suéltela encima del objeto destinatario. La línea de vida se resalta cuando el mensaje se puede soltar.



Notas:

- La dirección en la que se arrastra la flecha define la dirección del mensaje. Los mensajes de respuesta pueden apuntar en ambas direcciones.
- Mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo para insertar varios mensajes.

Para eliminar un mensaje:

1. Haga clic en el mensaje que desea eliminar.
2. Pulse la tecla **Supr** para eliminarlo del modelo. También puede hacer clic con el botón derecho en el mensaje y elegir el comando **Eliminar solo en el diagrama**.

9.2 Diagramas de estructura

Los diagramas de estructura muestran qué elementos estructurales componen un sistema o una función. Pueden representar tanto las relaciones estáticas (p. ej. diagramas de clases) como las dinámicas (p. ej. diagramas de objetos).

Diagramas de clases
Diagramas de componentes
Diagramas de estructura compuesta
Diagramas de implementación
Diagramas de objetos
Diagramas de paquetes
Diagramas de perfil

9.2.1 Diagrama de clases

Esta sección se ocupa de describir las tareas y los conceptos relacionados con los diagramas de clases:

- Personalizar diagramas de clases
- Invalidar operaciones de clases base e implementar operaciones de interfaz
- Crear métodos getter y setter
- Notaciones de tipo bola (ball and socket)
- Agregar excepciones emitidas a los métodos de una clase
- Adding Receptions to a Class
- Generar diagramas de clases

Para obtener información general sobre los diagramas de clases, consulte el apartado *Diagramas de clases* del tutorial que acompaña a esta documentación.

9.2.1.1 Personalizar diagramas de clases

Expandir/ocultar los compartimentos de las clases en un diagrama UML

Hay varias maneras de expandir los compartimentos de los diagramas de clases.

- Haga clic en los botones **+** o **-** de la clase activa para expandir/contraer el compartimento correspondiente.
- Use el recuadro de selección (arrastrando el puntero por el diagrama) para marcar varias clases y después haga clic en el botón expandir/ocultar. También puede usar **Ctrl+clic** para seleccionar varias clases.
- Pulse **Ctrl+A** para seleccionar **todas las clases** y después haga clic en el botón expandir/ocultar en una de las clases para expandir/contraer los compartimentos correspondientes.

Expandir/ocultar los compartimentos de las clases en la Estructura del modelo

En la *Estructura del modelo*, las clases son subelementos de los paquetes y la acción expandir/ocultar se puede ejecutar en los paquetes o en las clases.

- Haga clic en el paquete / en la clase que desea expandir y:
  - Pulse la tecla * para expandir el paquete/la clase actual y todos los subelementos.
  - Pulse la tecla + para abrir el paquete/la clase actual.

Para **contraer** los paquetes/las clases, pulse la tecla del teclado **-**.

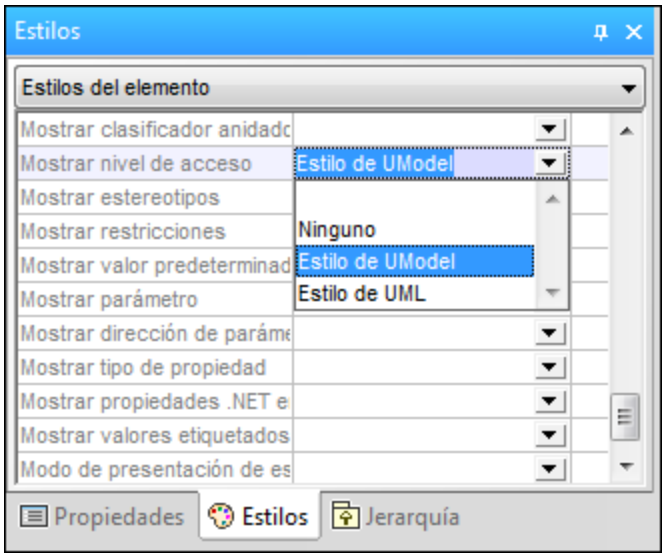
Recuerde que puede usar las teclas del teclado estándar o del teclado numérico para hacer esto.

Cambiar el icono de nivel de acceso

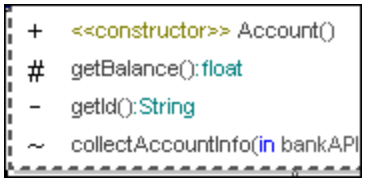
Haga clic en el **icono de nivel de acceso** situado a la izquierda de una operación o propiedad para abrir una lista desplegable en la que puede elegir el nivel de acceso del elemento.

En UModel también puede elegir qué tipo de símbolo se utiliza para identificar los niveles de acceso nivel de acceso:

- Haga clic en una clase del diagrama y abra la ventana **Estilos**. Desplácese hasta el estilo **Mostrar nivel de acceso**.



Aquí puede elegir entre usar el estilo de UModel, el estilo de UML (*imagen siguiente*) o no utilizar ninguno.



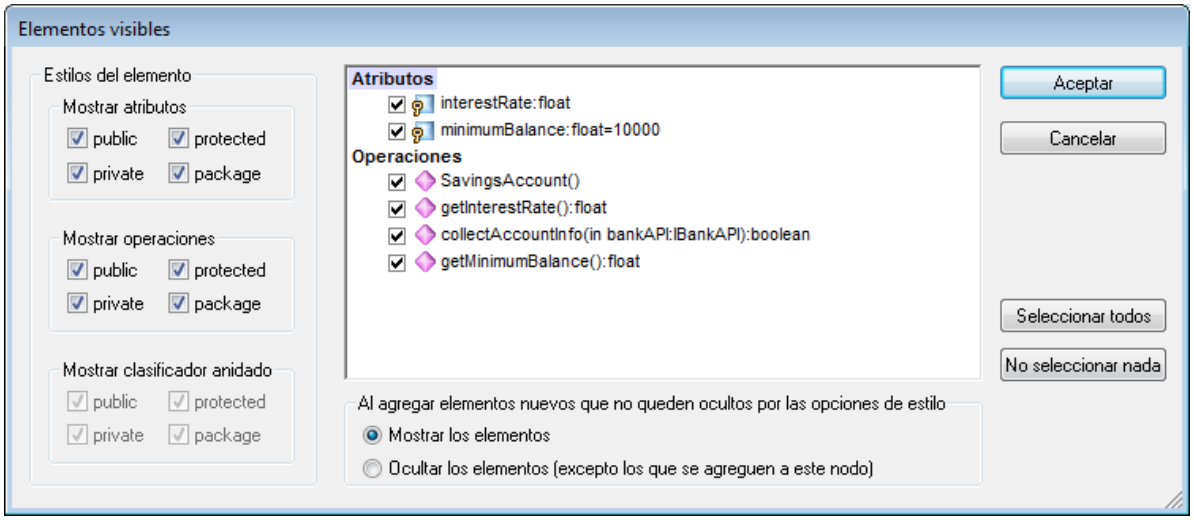
Mostrar/ocultar el contenido de los nodos (atributos de clase, operaciones, slots)

En los diagramas de clase puede mostrar u ocultar miembros específicos de una clase, como atributos u operaciones. También puede mostrar u ocultar miembros múltiples del mismo tipo según su nivel de visibilidad.

Por ejemplo, puede ocultar solamente los atributos de clase privados. En los diagramas de objeto también puede mostrar u ocultar slots de objetos (`InstanceSpecifications`).

Para mostrar y ocultar miembros de clase u slots de objetos:

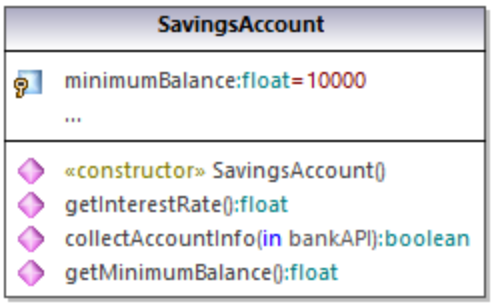
1. Haga clic con el botón derecho en una clase (por ejemplo `SavingsAccount` en el proyecto de ejemplo **Bank_MultiLanguage.ump**) y seleccione **Mostrar u ocultar contenido del nodo** en el menú contextual.
2. Marque o desmarque la casilla que hay junto a los miembros que quiere mostrar u ocultar.



Para mostrar u ocultar varios miembros en base a su visibilidad, use las casillas del grupo **Estilos del elemento**. Por ejemplo, desmarque la casilla **protected** del grupo **Mostrar atributos** si quiere ocultar todos los atributos protegidos de esa clase.

Nota: si elige ocultar elementos se ocultarán también sus valores etiquetados.

Una vez haya confirmado sus preferencias haciendo clic en Aceptar y haya cerrado el cuadro de diálogo los elementos ocultos del diagrama son reemplazados por puntos suspensivos. Para volver a abrir el cuadro de diálogo haga doble clic en los puntos suspensivos.



La opción **Al agregar elementos nuevos que no queden ocultos por las opciones de estilo** permite definir qué elementos de los que se añaden a la clase son visibles. Esto no solo afecta a los elementos que se agreguen manualmente al diagrama o a la Estructura del modelo, sino también a los que se añaden automáticamente durante el proceso de ingeniería de código. Estos son los valores válidos para esta opción:

Mostrar elementos	Con esta opción, cuando se añade un miembro nuevo a la clase, aparece en el diagrama. Sin embargo, si alguna de las opciones de "Estilos del elemento" indica que el elemento debe ocultarse, se oculta.
Ocultar los elementos (excepto los que se agreguen a este nodo)	El término "nodo" aquí se refiere a la instancia actual de la clase en el diagrama. (Recuerde que se puede añadir la misma clase varias veces al mismo diagrama; consulte Renombrar, mover y copiar elementos.) Cuando en el diagrama hay una o más instancias de la misma clase y si se añade un miembro nuevo a *esta instancia* de la clase, entonces esta opción oculta ese miembro en todas las instancias pero lo muestra para la instancia actual.

Para ver un ejemplo de cómo estas opciones pueden serle útiles, abra el proyecto de ejemplo **Bank_MultiLanguage.ump** y busque el diagrama "Hierarchy of Account".

A continuación, cree una instancia nueva de la clase `SavingsAccount`:

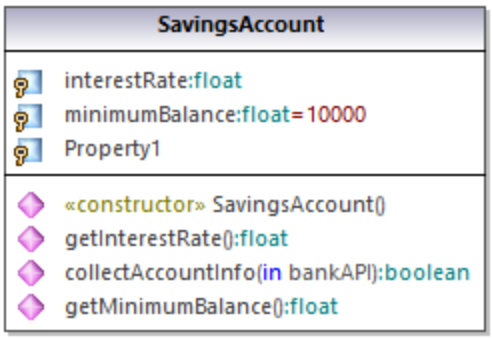
1. Haga clic con el botón derecho en la clase `SavingsAccount` del diagrama y seleccione **Copiar**.
2. Haga clic con el botón derecho en un área vacía del mismo diagrama y seleccione **Pegar sólo en el diagrama** en el menú contextual.

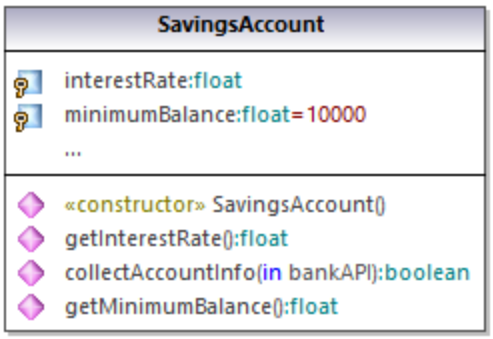
Ahora hay dos instancias de la clase `SavingsAccount` en el diagrama.

A continuación puede definir distintas opciones de visibilidad en cada una de las instancias:

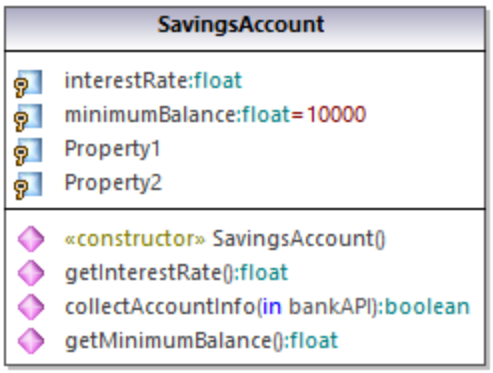
1. Haga clic con el botón derecho en la instancia izquierda de la clase, seleccione **Mostrar u ocultar contenido del nodo** y después seleccione la opción **Mostrar elementos**.
2. Haga clic con el botón derecho en la instancia derecha de la clase, seleccione **Mostrar u ocultar contenido del nodo** y después seleccione la opción **Ocultar los elementos (excepto los que se agreguen a este nodo)**.

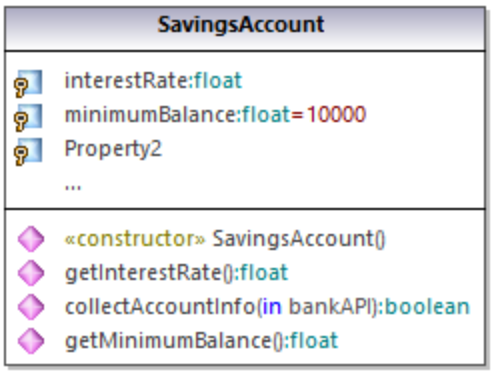
A continuación agregue una propiedad nueva a la instancia izquierda (seleccione la clase y pulse **F7**). Como se ve a continuación, la propiedad nueva (`Property1`) aparece en la instancia izquierda pero no en la derecha. Esto se debe a que en la instancia derecha hemos habilitado la opción **Ocultar los elementos (excepto los que se agreguen a este nodo)**.





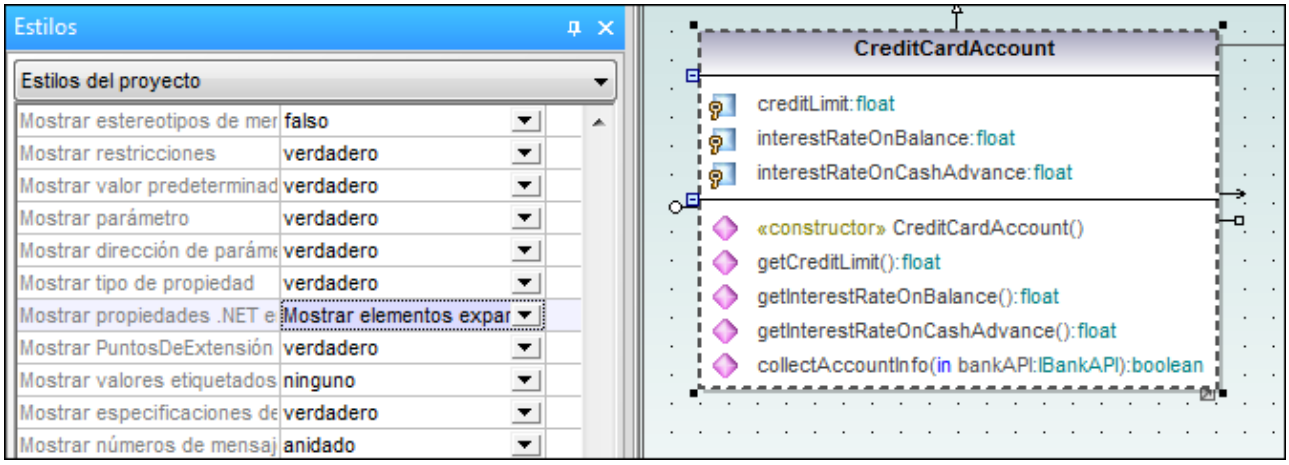
Por último, agregue una propiedad nueva a la instancia derecha de la clase. Como puede ver a continuación, esta propiedad nueva (`Property2`) aparece en las dos instancias. Esto se debe a que en la instancia izquierda la configuración permite mostrar elementos nuevos, mientras que la instancia derecha es la instancia *actual*, por lo que la propiedad aparece en cualquier caso.





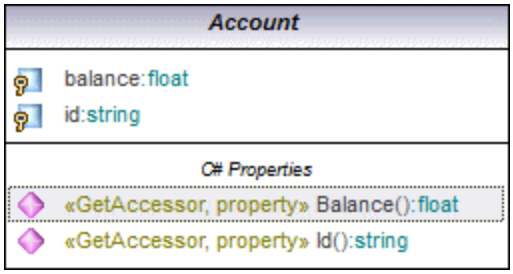
Mostrar/ocultar compartimentos VS .NET

Para mostrar las propiedades .NET en un compartimento separado, habilite el estilo Mostrar propiedades .NET en un compartimento propio de la ventana **Estilos**.



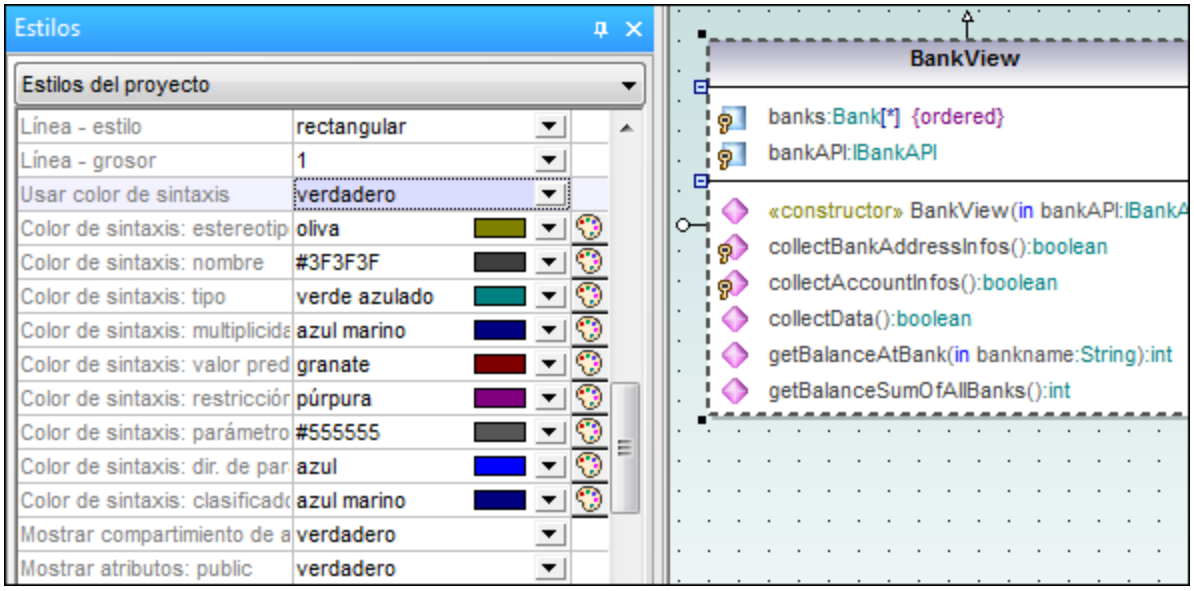
Mostrar las propiedades VS .NET como asociaciones

UModel también puede mostrar las propiedades .NET como asociaciones. Haga clic con el botón derecho en una propiedad C# y elija **Mostrar | Todas las propiedades .NET como asociaciones** en el menú contextual.



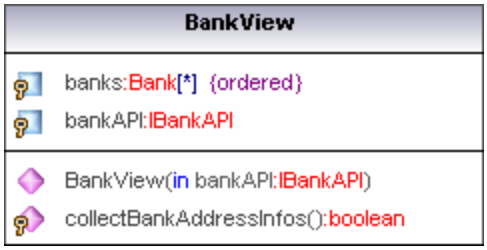
Cambiar el color de sintaxis de las operaciones y propiedades

UModel habilita automáticamente la función de color de sintaxis, pero esta función se puede personalizar. A continuación puede ver la configuración predeterminada.



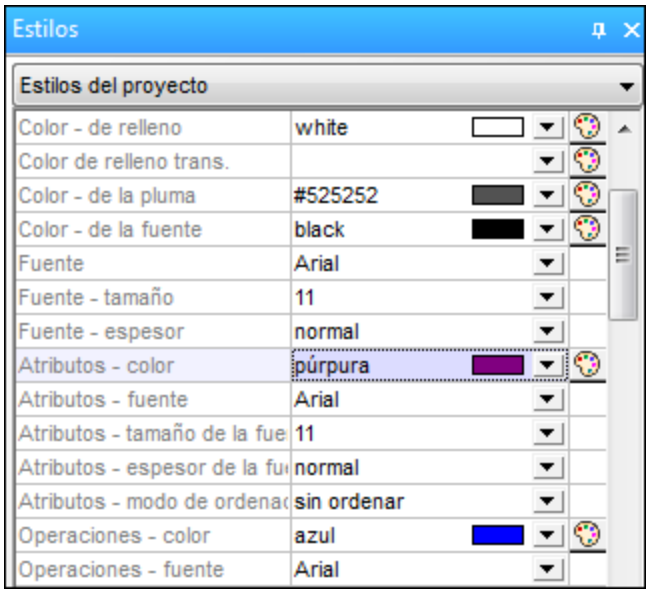
Para cambiar las opciones predeterminadas de color de sintaxis:

1. Abra la ventana *Estilos* y desplácese hasta los estilos que empiezan por `Color de sintaxis`.
2. Cambie el valor de uno de estos estilos. Por ejemplo: cambie el valor de `Color de sintaxis: tipo` a `red`.



Para deshabilitar el color de sintaxis:

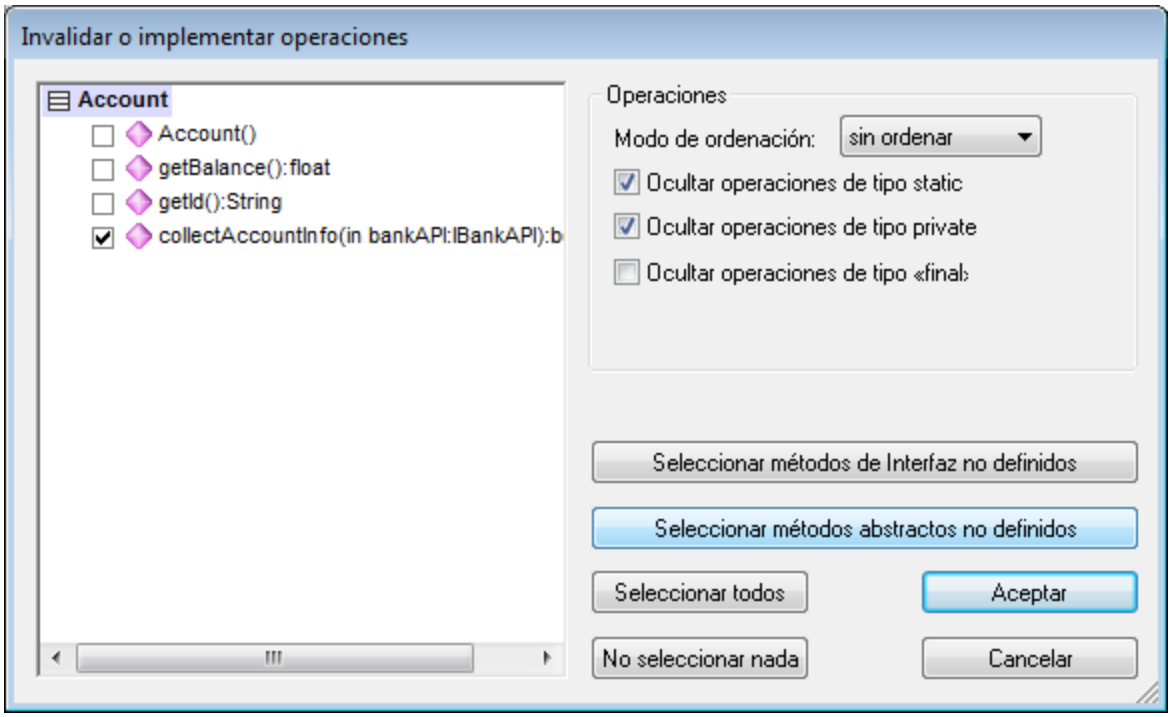
1. Abra la ventana *Estilos* y asigne el valor `false` al estilo `Usar color de sintaxis`.
2. Y ahora use los estilos `Atributos - color` o los estilos `Operaciones - color` para personalizar estos elementos en las clases.



9.2.1.2 Invalidar operaciones de clases base e implementar operaciones de interfaz

UModel ofrece la posibilidad de invalidar las operaciones de la clase base o implementar operaciones de interfaz de una clase. Esto se puede hacer desde la Estructura del modelo, desde *Favoritos* o desde diagramas de clases.

1. Haga clic con el botón derecho en una de las clases derivadas del diagrama (p. ej. **`CheckingAccount`**) y elija **Invalidar o implementar operaciones** en el menú contextual. Esto abre el cuadro de diálogo "Invalidar o implementar operaciones" (*imagen siguiente*).



2. Seleccione las operaciones que desea invalidar y haga clic en **Aceptar** para confirmar.
   Con los botones **Seleccionar métodos...** puede seleccionar esos tipos de métodos en la vista previa de la izquierda.

Nota: cuando se abre este cuadro de diálogo, se marcan (se activan) las operaciones de las clases base y de las interfaces implementadas que tiene la misma firma que las operaciones disponibles.

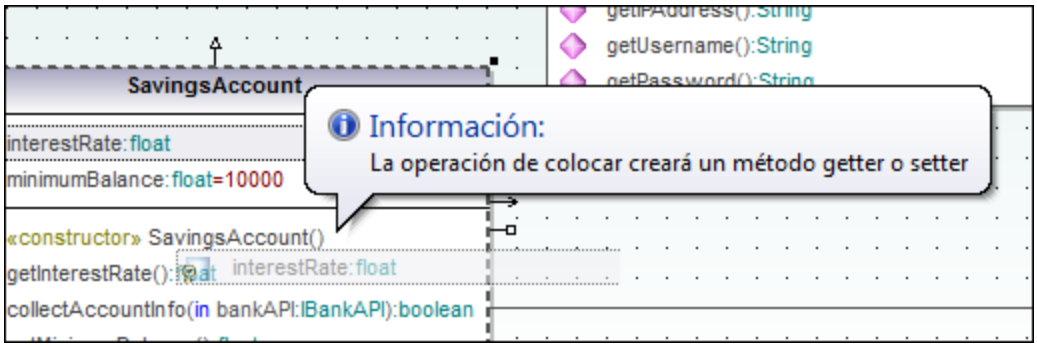
9.2.1.3 Crear métodos getter y setter

Durante el proceso de modelado a veces es necesario crear métodos getter/setter para los atributos existentes. UModel ofrece dos métodos para crear métodos getter/setter:

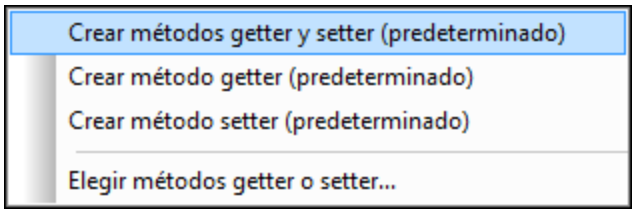
- Arrastrando y colocando un atributo en el compartimento de una operación.
- Usando el menú contextual para abrir un cuadro de diálogo donde puede gestionar los métodos getter/setter.

Para crear métodos getter/setter mediante operaciones arrastrar y colocar:

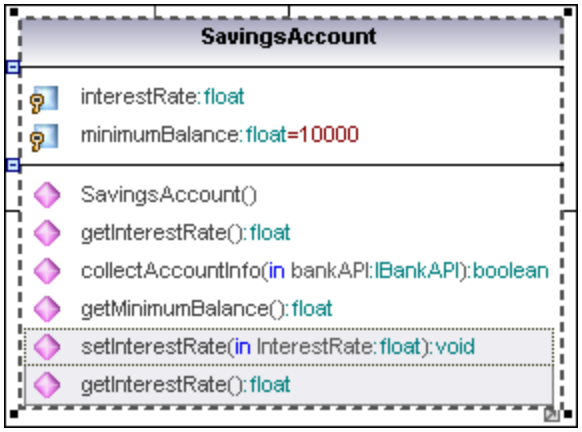
1. Arrastre un atributo desde el compartimento de atributos hasta el compartimento de operaciones.



Aparece un menú contextual donde puede elegir el tipo de método getter/setter que se debe crear.

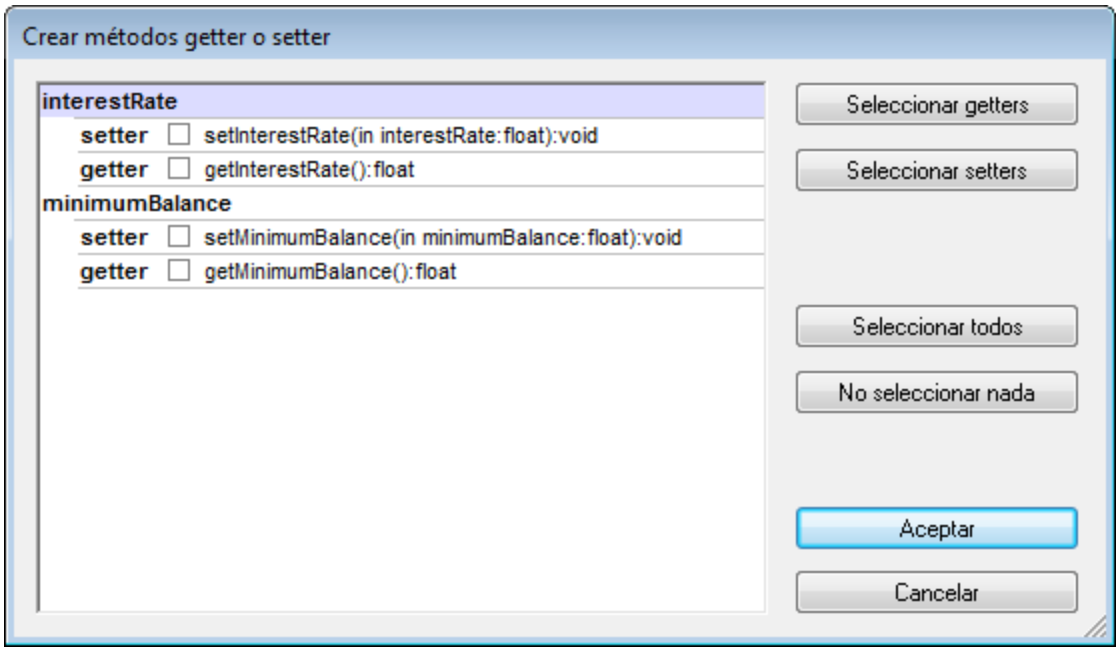


2. Seleccione el primer comando del menú para crear un método getter y setter para interestRate:float.



Para crear métodos getter/setter con el menú contextual:

1. Haga clic con el botón derecho en el título de una clase (p. ej. **`SavingsAccount`**) y elija la opción **Crear operaciones de método getter o setter...** del menú contextual. Esto abre el cuadro de diálogo "Crear métodos getter o setter", que enumera los atributos disponibles en la clase activa.

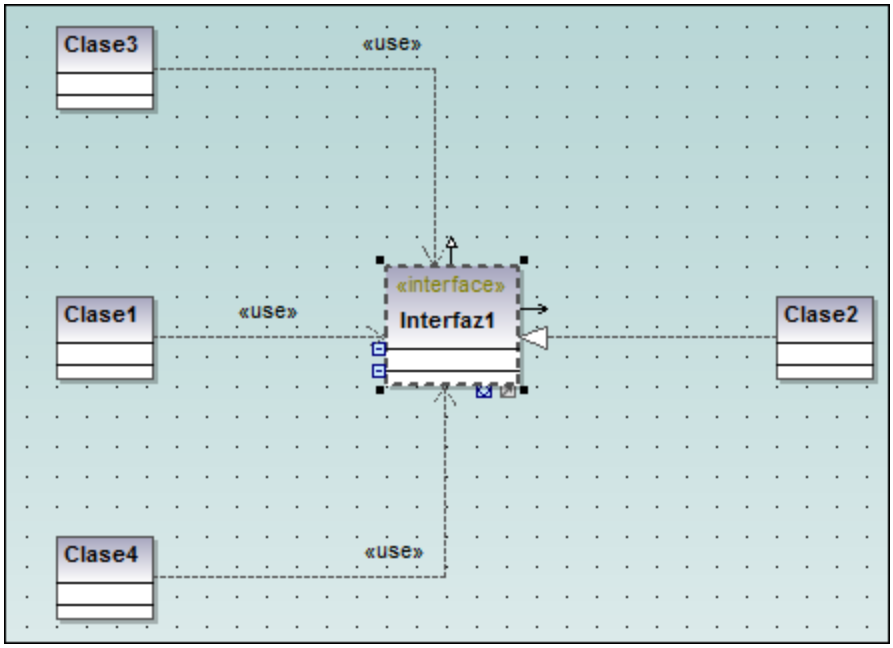


2. Use los botones o marque las casillas para seleccionar los métodos que desea crear.

Nota: también puede hacer clic con el botón derecho en un solo atributo y usar el mismo método para crear una operación para el atributo.

9.2.1.4 Notaciones de forma esférica (Ball and socket)

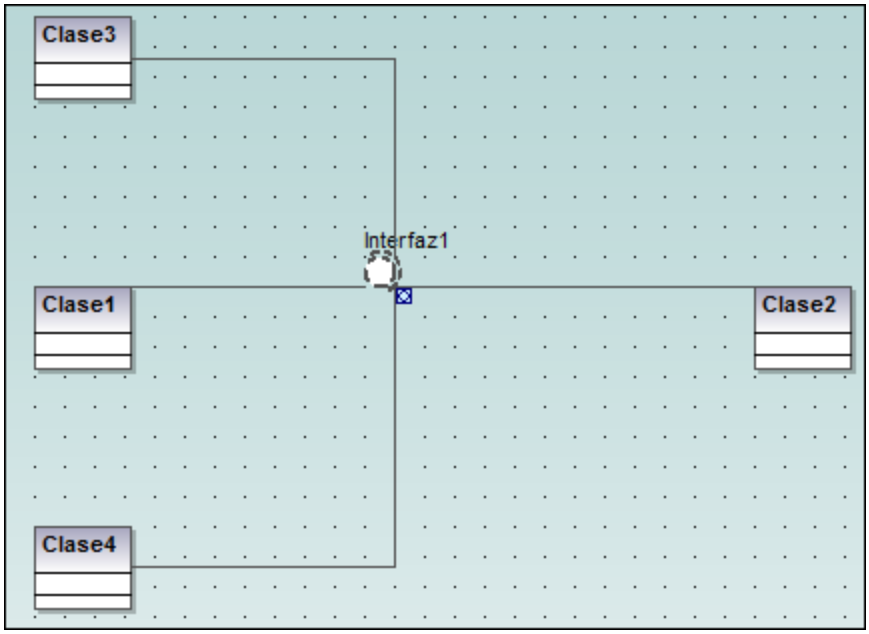
UModel es compatible con la notación UML de tipo bola-enchufe (*ball-socket*). Las clases que necesitan una interfaz muestran un círculo, que representa a la bola, y el nombre de la interfaz. Las clases que implementan la interfaz muestran "enchufe" cóncavo en el que encajar la bola.



En la imagen anterior, la `Clase2` realiza la `Interfaz1`, que es utilizada por las clases `clase1`, `clase3` y `clase4`. Para crear la relación de utilización entre las clases y la interfaz se usó el icono **Utilización** de la barra de herramientas *Diagrama de clases*.

Para cambiar entre la vista estándar y la vista de tipo bola:

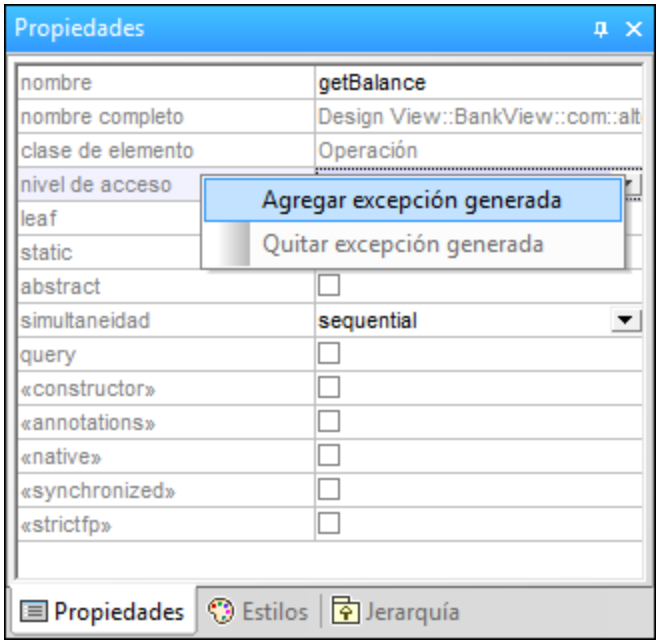
- Haga clic en el icono **Alternar estilo de notación de interfaz** situado en la parte inferior del elemento interfaz.



9.2.1.5 Agregar excepciones emitidas a los métodos de una clase

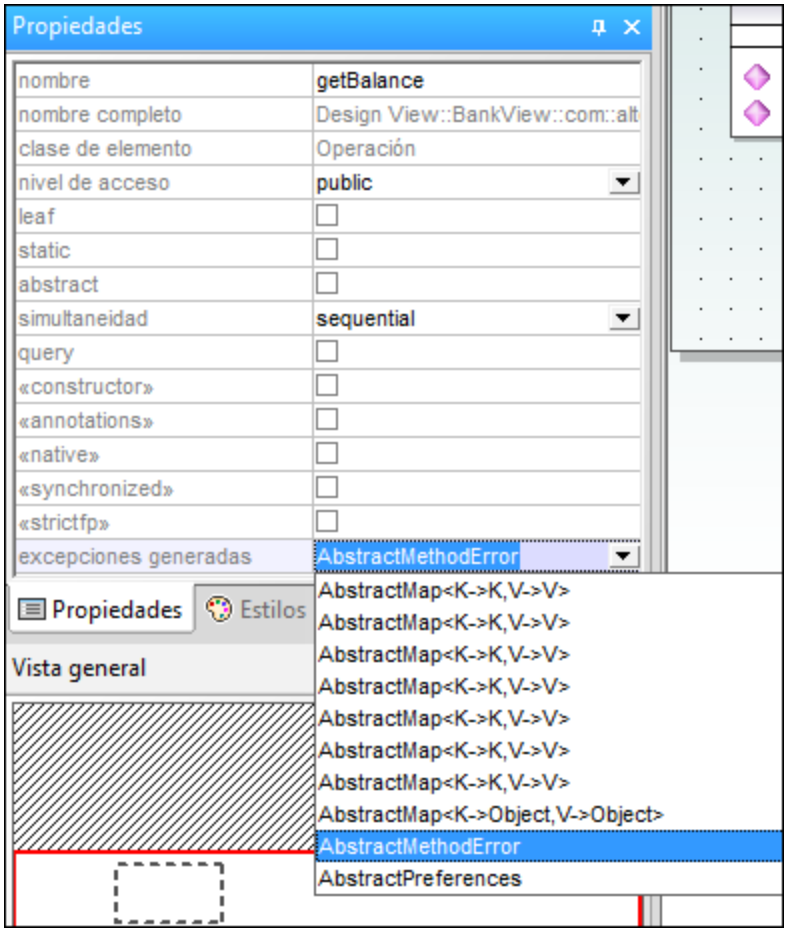
Para agregar excepciones generadas a los métodos de una clase:

1. En la *Estructura del modelo* haga clic en el método de la clase en la que desea agregar la excepción generada (p. ej. el método `getBalance` de la clase `Account`).
2. Ahora haga clic con el botón derecho dentro de la ventana *Propiedades* y elija la opción **Agregar excepción generada** del menú contextual.



Esto añade el campo Excepciones generadas al panel *Propiedades* y selecciona la primera opción de la lista desplegable.

3. Seleccione una opción de la lista despegable del campo Excepciones generadas o escriba el nombre que quiera.



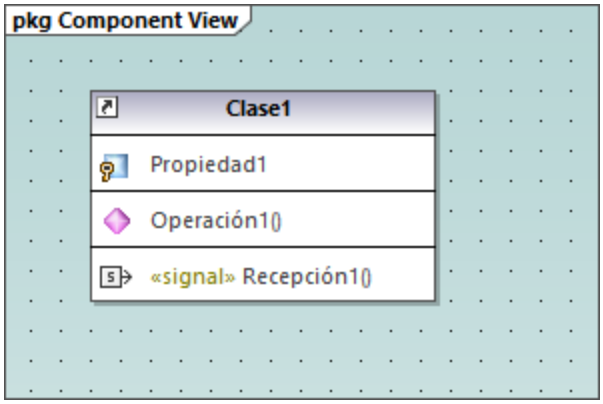
9.2.1.6 Agregar elementos recepciones a una clase

Además de operaciones y propiedades, también puede añadir elementos Recepción a una clase.

Para agregar un elemento Recepción a una clase:

- Haga clic con el botón derecho en el diagrama y seleccione **Elemento nuevo | Recepción** en el menú contextual.

Las recepciones aparecen en un compartimento distinto en el diagrama de clase, como ocurre con las propiedades y las operaciones:



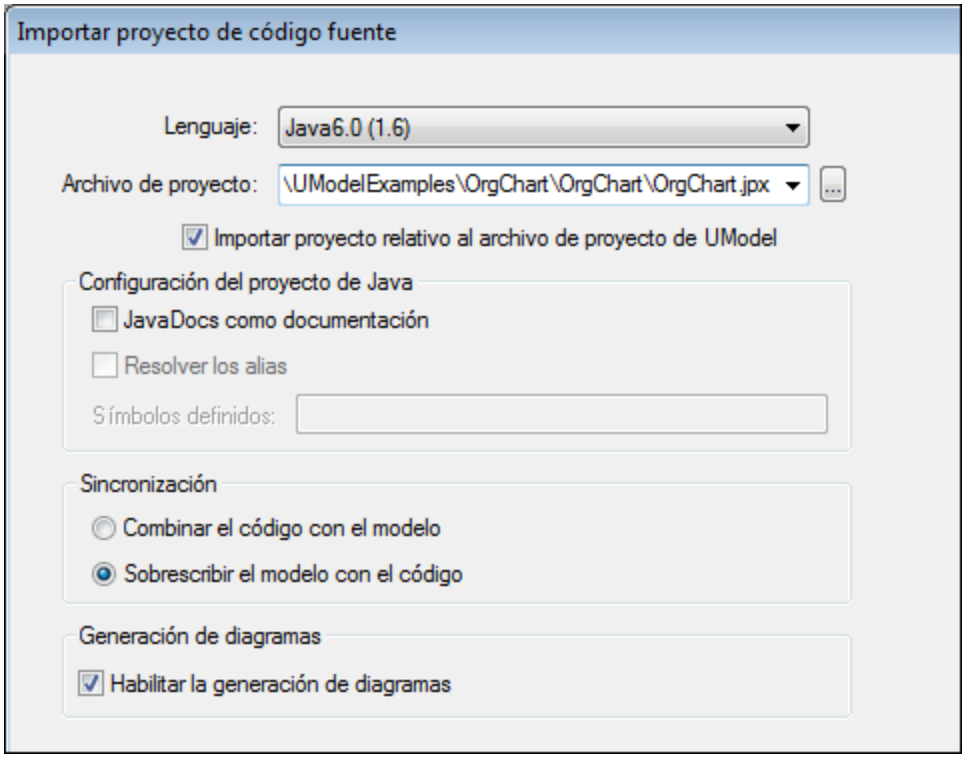
Las recepciones comparten estilos con las operaciones. Esto significa que siempre que cambie un estilo para las operaciones, esos cambios afectarán también a las recepciones. Para más información consulte Cambiar el estilo de los elementos de un diagrama.

9.2.1.7 Generar diagramas de clases

Si lo prefiere, en lugar de diseñar diagramas de clases en UModel directamente, también puede generarlos automáticamente cuando importe código fuente o binarios en sus proyectos de UModel (consulte los apartados Importar código fuente e Importar archivos binarios Java, C# y VB.NET).

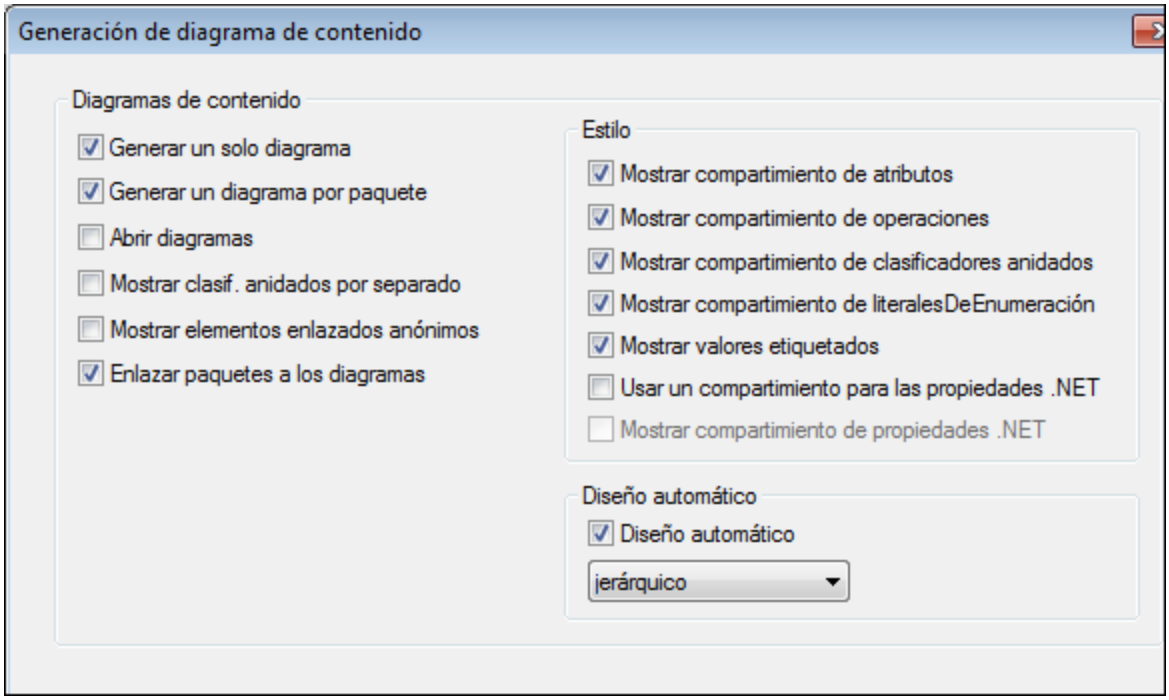
Siga las instrucciones del asistente para la importación y asegúrese de que:

1) La casilla *Habilitar generación de diagramas* está marcada en el cuadro de diálogo "Importar proyecto de código fuente", "Importar tipos binarios" o "Importar directorio de código fuente".



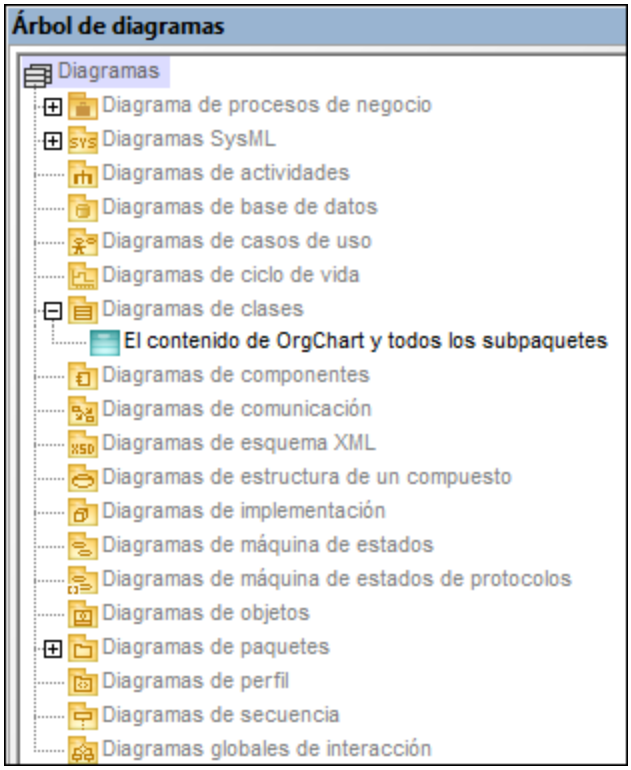
Cuadro de diálogo Importar proyecto de código fuente

2) Las opciones *Generar un solo diagrama* y *Generar un diagrama por paquete* están marcadas en el cuadro de diálogo "Generación de diagrama de contenido".



Cuadro de diálogo Generación de diagrama de contenido

Una vez finalizada la operación de importación, los diagramas de clases generados estarán disponibles bajo el nodo `Diagramas de clases` del *Árbol de diagramas*.

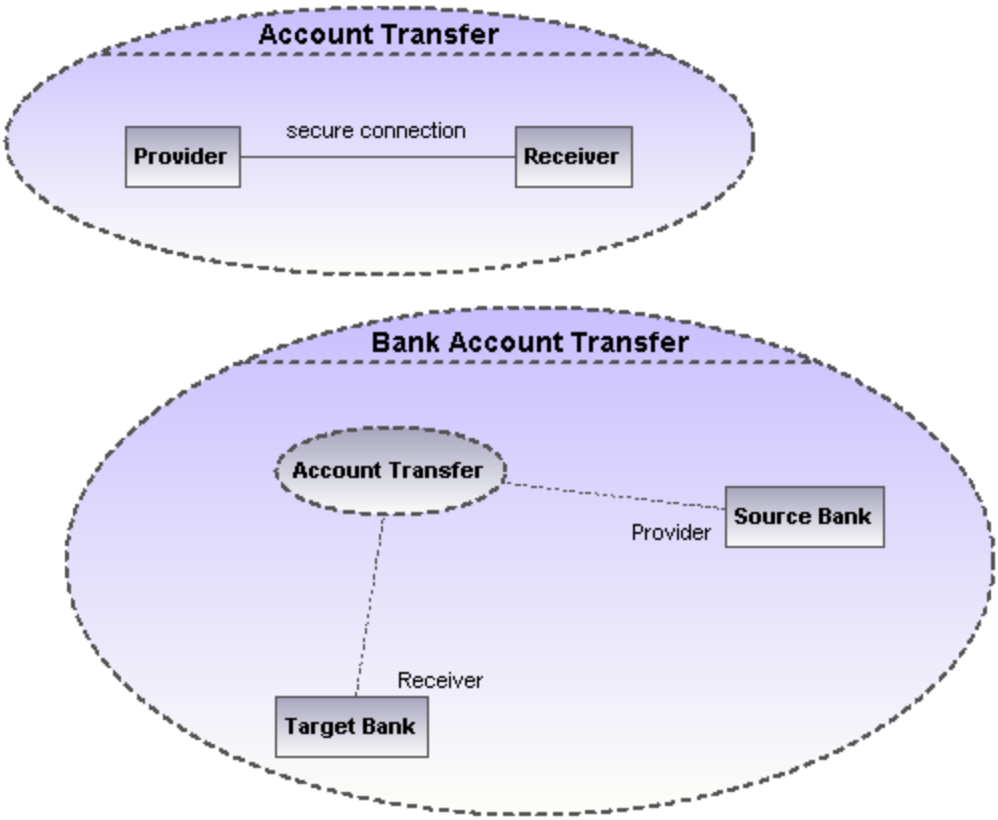


Árbol de diagramas

9.2.2 Diagrama de estructura compuesta

Sitio web de Altova: diagramas de estructura compuesta UML

Los diagramas de estructura compuesta son un tipo de diagrama añadido en la especificación UML 2.0 y sirven para mostrar la estructura interna (partes, puertos, conectores...) de un clasificador estructurado o colaboración.



9.2.2.1 Insertar elementos

Insertar elementos con los iconos de la barra de herramientas

1. Haga clic en un icono de la barra de herramientas Diagrama de estructura compuesta.

2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama de estructura compuesta

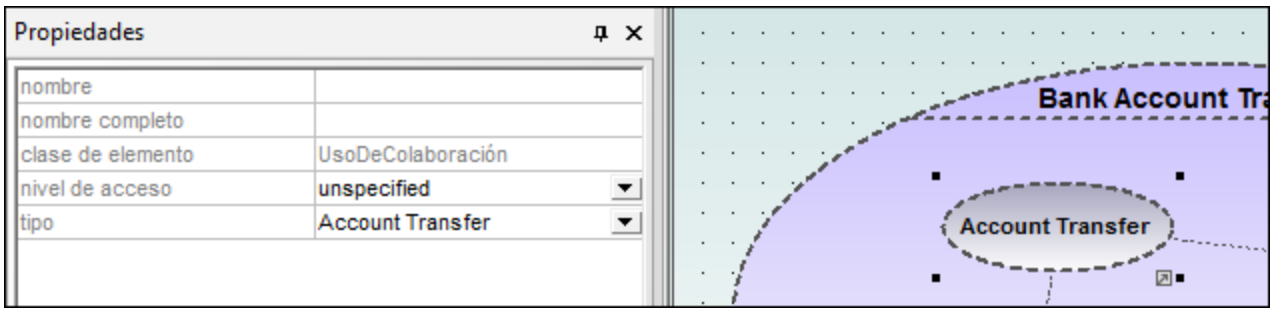
1. En la *Estructura del modelo* busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de estructura compuesta.

Colaboración

Este icono inserta un elemento `colaboración`, que es un tipo de clasificador/instancia que se comunica con otras instancias para producir el comportamiento del sistema.

UsoDeColaboración

Este icono inserta un elemento `usoDeColaboración`, que representa un uso determinado de una colaboración en la que participan determinadas clases o instancias que desempeñan el papel de la colaboración. Un uso de colaboración se representa por medio de una elipse con una línea de puntos. Dentro de la elipse aparece el nombre de la instancia, un punto y coma y el nombre del tipo de colaboración.



Cuando cree dependencias entre elementos uso de colaboración, es necesario rellenar el campo tipo para poder crear el enlace de roles y la colaboración de destino debe tener una parte/un rol como mínimo.

Parte (Propiedad)

Inserta un elemento `parte`, que representa un conjunto de instancias propiedad de un clasificador contenedor. Los elementos parte pueden añadirse a colaboraciones y a clases.

Puerto

Inserta un elemento `puerto`, que define el punto de interacción entre un clasificador y su entorno. Los elementos puerto pueden añadirse en partes con un tipo definido.

Clase

Inserta un elemento `clase`, que es el clasificador que tiene lugar en ese uso concreto de la colaboración.

Conector

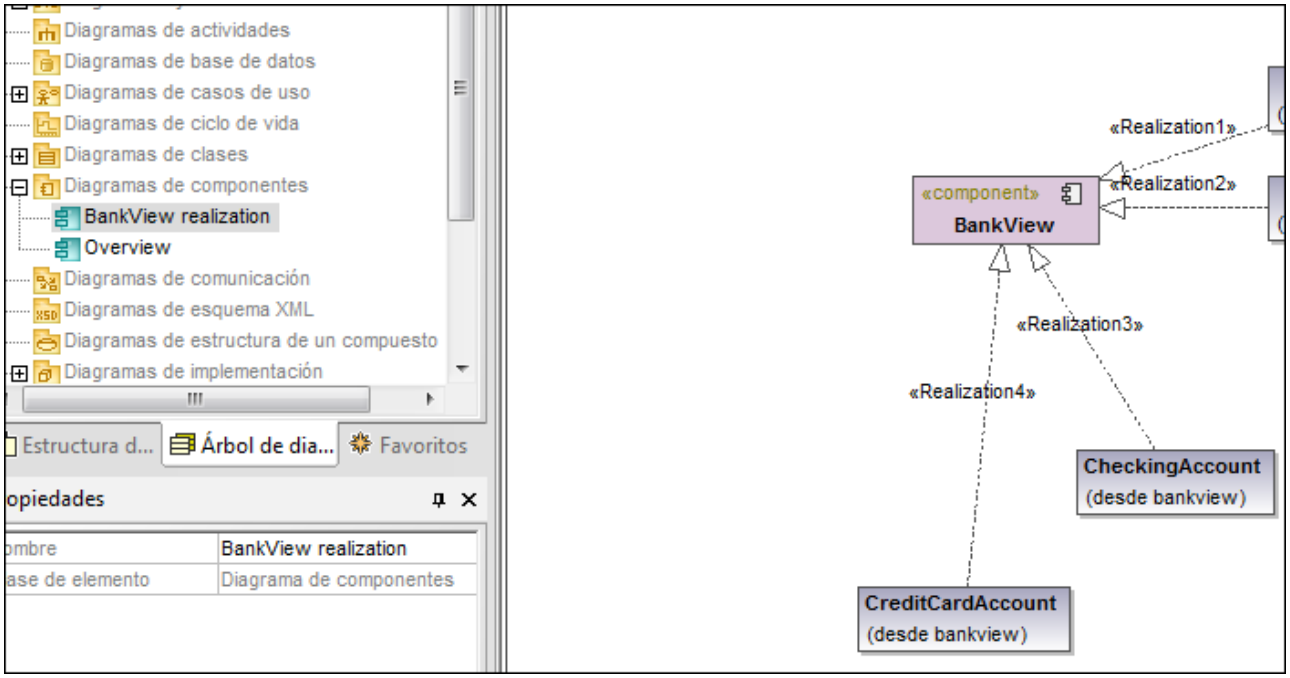
Inserta un elemento `conector`, que se puede usar para conectar dos o más instancias de una parte o de un puerto. El conector define la relación entre los objetos e identifica la comunicación entre los roles.

Dependencia (Enlace de roles)

Inserta el elemento `dependencia`, que indica qué rol desempeña cada elemento conectable del clasificador o de la operación en la colaboración.

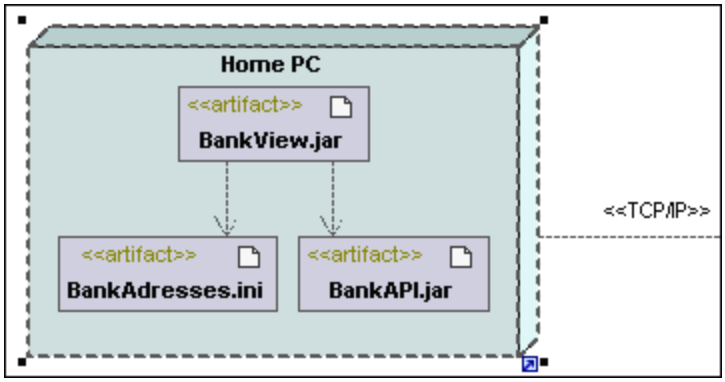
9.2.3 Diagrama de componentes

Para más información sobre cómo añadir componentes al diagrama consulte el apartado Diagramas de componentes.



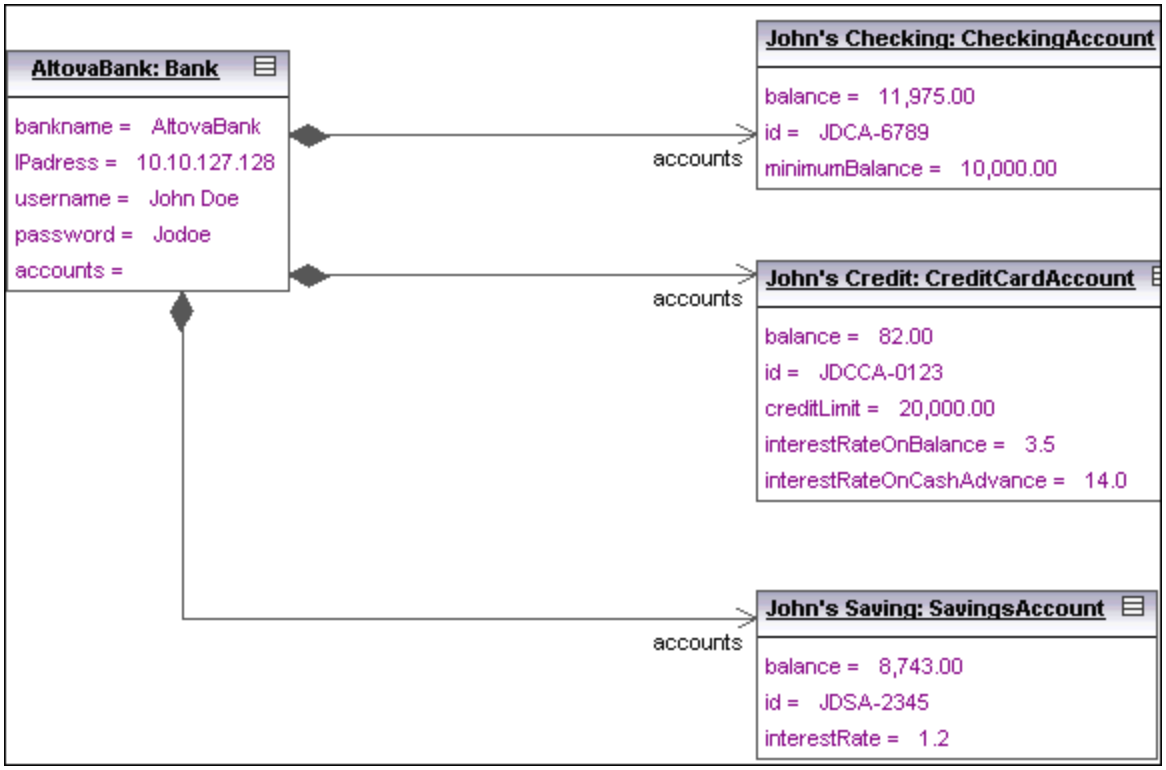
9.2.4 Diagrama de implementación

Para más información sobre cómo agregar nodos y artefactos al diagrama consulte el apartado Diagramas de implementación del tutorial.



9.2.5 Diagrama de objetos

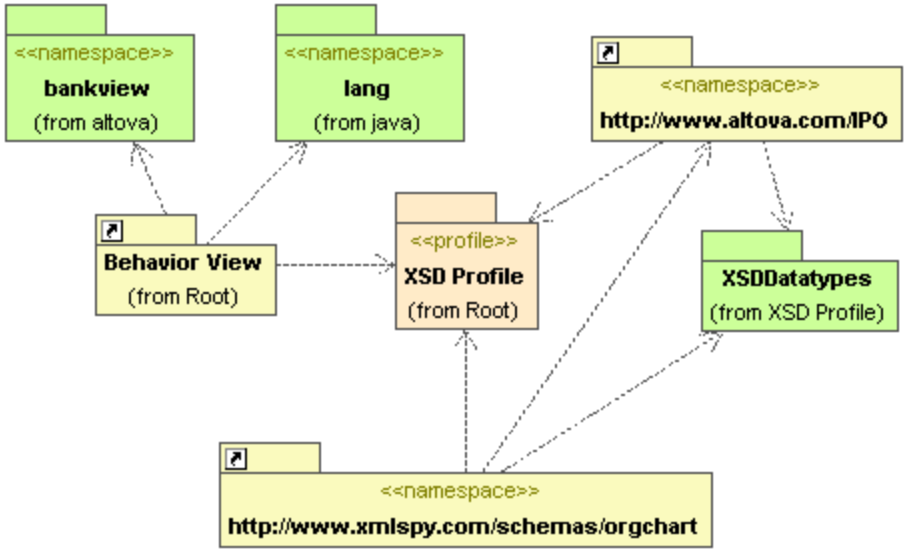
Para más información sobre cómo agregar objetos/instancias nuevos al diagrama consulte el apartado Diagramas de objetos del tutorial.



9.2.6 Diagrama de paquetes

Los diagramas de paquetes ilustran la organización de los paquetes y de sus elementos, así como sus correspondientes espacios de nombres. Además en UModel puede crear hipervínculos para navegar hasta el contenido del paquete correspondiente.

Los paquetes se dibujan en forma de carpetas y se pueden usar en cualquier diagrama UML, aunque se usan sobre todo en diagramas de casos de uso y de clases.



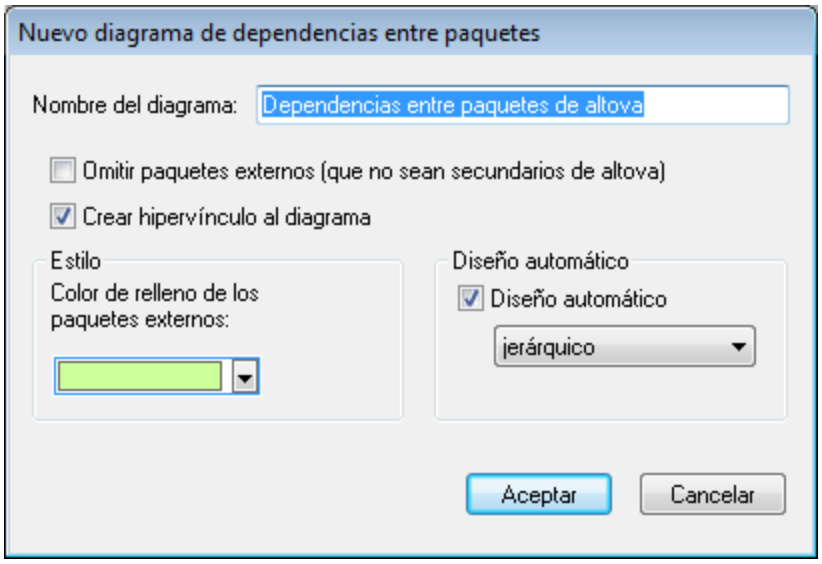
Generación automática de diagramas de dependencias entre paquetes

UModel tiene una función para generar diagramas de dependencias entre paquetes a partir de cualquier paquete de la ventana *Estructura del modelo*.

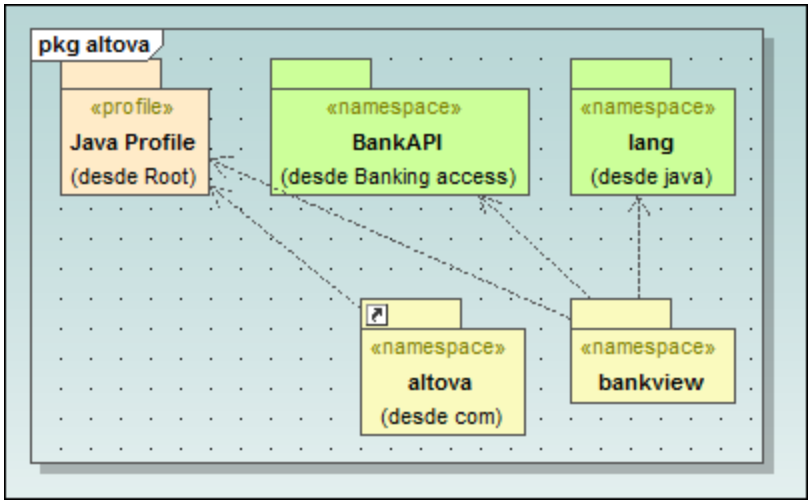
Esta función crea vínculos de dependencia entre los paquetes si existen referencias entre sus elementos de modelado. P. ej. si hay dependencias entre clases, si hay clases derivadas o si los atributos tienen tipos que están definidos en otro paquete.

Para generar un diagrama de dependencias entre paquetes:

1. En la *Estructura del modelo* haga clic con el botón derecho en un paquete (p. ej. `altova`) y elija **Mostrar en un diagrama nuevo de | Dependencias entre paquetes...**. Esto abre el cuadro de diálogo "Nuevo diagrama de dependencias entre paquetes".



2. Seleccione las opciones que necesite en el cuadro de diálogo y haga clic en **Aceptar**.



UModel genera un diagrama nuevo y muestra las dependencias entre los paquetes del paquete **`altova`**.

9.2.6.1 Insertar elementos

Insertar elementos con los iconos de la barra de herramientas

1. Haga clic en un icono de la barra de herramientas Diagrama de paquetes.

2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama de paquetes

1. En la *Estructura del modelo* busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de paquetes.

 Paquete

Inserta el elemento `paquete` en el diagrama. Los paquetes sirven para agrupar elementos y para aportar un espacio de nombres para los elementos agrupados. Al ser un espacio de nombres, un paquete puede importar elementos concretos de otros paquetes o todos sus elementos. Los paquetes también se pueden combinar con otros paquetes.

 Perfil

Inserta el elemento `perfil`, un tipo de paquete que se puede aplicar a otros.

El paquete `perfiles` se usa para extender el metamodelo UML. La construcción de extensión principal es el estereotipo, que a su vez es parte del perfil. Los perfiles siempre deben estar relacionados con un metamodelo de referencia como UML, es decir, no pueden existir por sí mismos.

 Dependencia

Inserta el elemento `dependencia`, que indica una relación de proveedor/cliente entre los elementos de modelado (en este caso paquetes o perfiles).

 ImportaciónDePaquete

Inserta una relación de importación <<import>> que muestra que los elementos del paquete incluido se importarán en el paquete de destino. El espacio de nombres del paquete de destino obtiene acceso al espacio de nombres incluido. El espacio de nombres del paquete incluido no se ve afectado.

Nota: los elementos private de un paquete no se pueden combinar ni importar.

 CombinaciónDePaquete

Inserta una relación de combinación <<merge>> que muestra que los elementos del paquete combinado (de origen) se importarán en el paquete de destino, incluido el contenido importado del paquete combinado (de origen).

Si en el paquete de destino existe el mismo elemento, las definiciones de estos elementos se expanden con las del paquete de destino. Los elementos actualizados o agregados se señalan por medio de una relación de generalización que apunta al paquete de origen.

Nota: los elementos private de un paquete no se pueden combinar ni importar.

AplicaciónDePerfil

Inserta una aplicación de perfil que muestra qué perfiles se aplicaron al paquete. Se trata de un tipo de importación de paquete que afirma que un perfil se aplicó a un paquete.

El perfil extiende el paquete al que se aplicó. Al aplicar un perfil (usando el icono **AplicaciónDePerfil**), todos los estereotipos que forman parte del perfil también estarán a disposición del paquete.

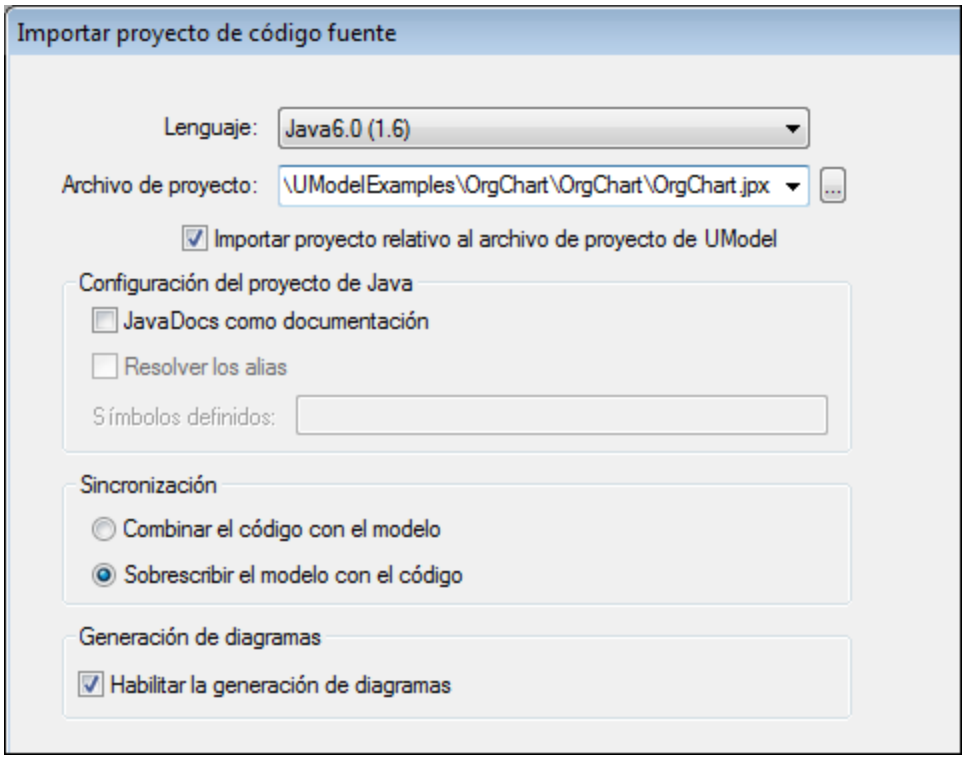
Los nombres de perfil aparecen en forma de líneas discontinuas con puntas de flecha que van del paquete hasta el perfil aplicado y llevan la etiqueta <<apply>>.

9.2.6.2 Generar diagramas de paquetes al importar código o binarios

Con UModel puede generar diagramas de paquetes al importar código fuente o binarios en el proyecto de UModel (consulte los apartados Importar código fuente e Importar archivos binarios Java, C# y VB.NET).

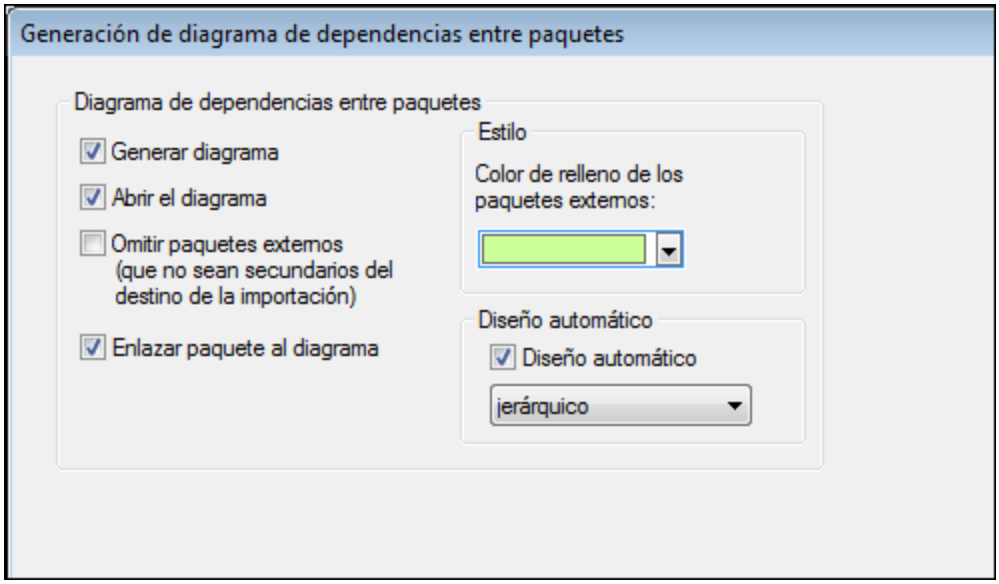
Siga las instrucciones del asistente para la importación y asegúrese de que:

1) La casilla *Habilitar generación de diagramas* está marcada en el cuadro de diálogo "Importar proyecto de código fuente", "Importar tipos binarios" o "Importar directorio de código fuente".



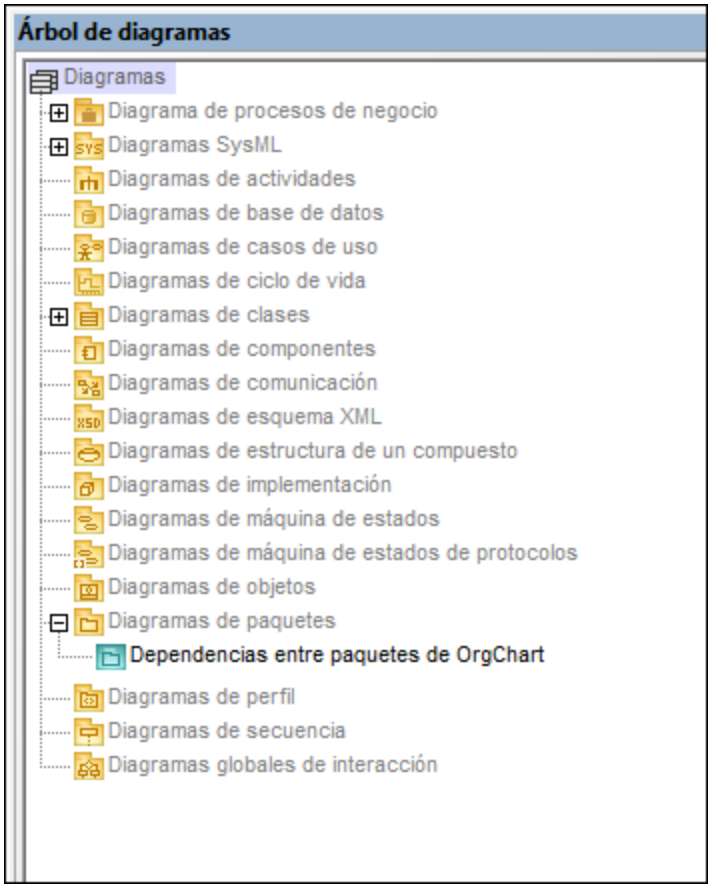
Cuadro de diálogo Importar proyecto de código fuente

2) La opción *Generar diagrama* está marcada en el cuadro de diálogo "Generación de diagrama de dependencias entre paquetes".



Cuadro de diálogo Generación de diagrama de dependencias entre paquetes

Una vez finalizada la operación de importación, los diagramas de clases generados estarán disponibles bajo el nodo `Diagramas de paquetes` del *Árbol de diagramas*.



Árbol de diagramas

9.2.7 Diagrama de perfil

Sitio web de Altova: Diagramas de perfil UML

En UML los perfiles son una forma de ampliar UML a una plataforma o un dominio. Al contrario que los paquetes, un perfil está en el metamodelo y consiste en bloques que amplían o limitan algo. Esto es posible con la ayuda de los siguientes mecanismos de extensión incluidos en un perfil: estereotipos, valores etiquetados y restricciones.

En el diagrama de perfil de UModel puede crear sus propios estereotipos, valores etiquetados y restricciones y guardarlos todos en un perfil personalizado. Este perfil permite ampliar o adaptar UML a un dominio específico o personalizar el aspecto de elementos de sus proyectos de modelado. Por ejemplo, puede que quiera personalizar estilos o iconos para elementos UML como clases, interfaces, etc.

El diagrama de perfil es donde puede aplicar un perfil a un paquete. Por ejemplo, el diagrama de perfil siguiente ilustra una relación **ProfileAppplication** entre el paquete **BankView** y el perfil Java integrado en UModel. Puede encontrar este diagrama en el siguiente proyecto de ejemplo: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\BankView_Java.ump**, que se llama "Apply Java Profile".

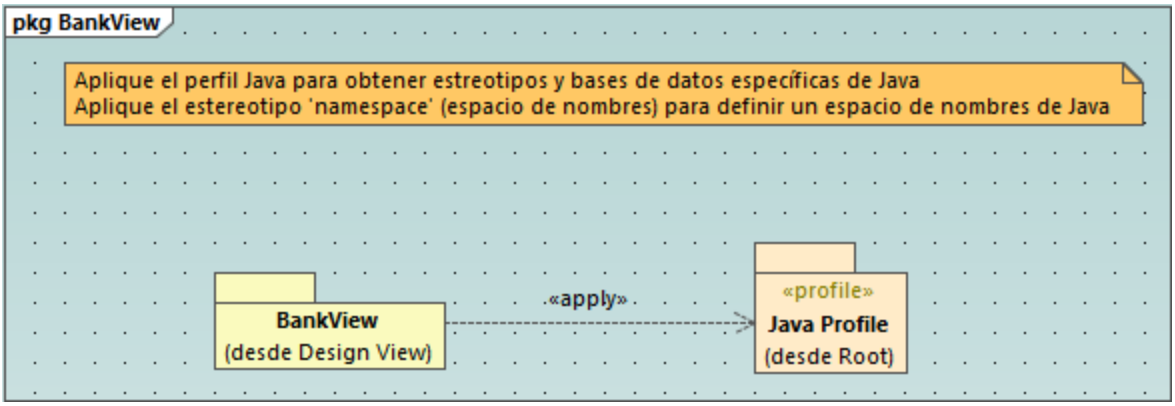


Diagrama de perfil

El perfil Java aplicado significa que cualquier clase o interfaz que forme parte del paquete **BankView** (o se añadirá a este paquete en el futuro) debe parecer una clase o interfaz Java y todos sus miembros deben tener el comportamiento típico de ese lenguaje. Por ejemplo:

- Todos los tipos de datos Java que existen en el perfil se pueden seleccionar en una lista desplegable al diseñar una clase en un diagrama de clases (*véase también Diagramas de clases)*.
- Todos los estereotipos específicos de Java definidos en el perfil, como `«annotations»`, `«final»`, `«static»`, `«strictfp»`, etc. son visibles como propiedades en la ventana *Propiedades* al seleccionar un elemento.

En este apartado explicamos cómo puede ampliar proyectos de UModel mediante perfiles y estereotipos personalizados. Para más información sobre cómo usar los perfiles integrados de UModel consulte Aplicar perfiles de UModel y Estereotipos y valores etiquetados.

9.2.7.1 Crear y aplicar perfiles personalizados

Las siguientes instrucciones explican cómo crear un perfil de UModel personalizado y aplicarlo a un paquete. Esto suele ser necesario si quiere crear y aplicar estereotipos además de los que ya vienen incluidos en los perfiles de UModel predeterminados. Para más información sobre cómo aplicar los perfiles predeterminados de UModel consulte Aplicar perfiles de UModel.

Para crear un perfil personalizado:

1. Haga clic con el botón derecho en el paquete en el que quiere crear el perfil nuevo (por ejemplo "Root") y seleccione **Elemento nuevo | Perfil** en el menú contextual.
2. Cree todos los elementos que deban formar parte de este perfil, como estereotipos, tipos de datos, etc. Puede hacerlo en la ventana *Estructura del modelo* o desde un diagrama de perfil. Por ejemplo, para crear un estereotipo nuevo en el modelo, haga clic en el perfil y seleccione **Elemento nuevo | Estereotipo** en el menú contextual. Consulte también el apartado Crear estereotipos.
3. También puede crear un diagrama de perfil (haga clic con el botón derecho en el perfil y seleccione **Diagrama nuevo | Diagrama de perfil** en el menú contextual). Para añadir todos los elementos necesarios al diagrama, use los comandos de menú y barras de herramientas estándar de UModel (*véase Cómo modelar*).

Si quiere crear el perfil a partir de un diagrama de perfil, asegúrese de que el diagrama pertenece a (se crea en) un perfil o a un paquete de dentro del perfil.

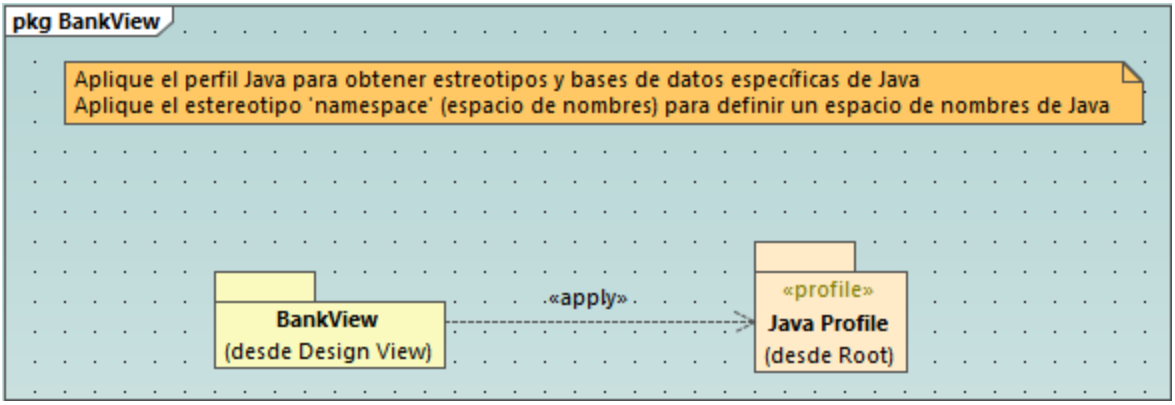
Además, si quiere reutilizar el perfil en varios proyectos de UModel, siga estos pasos:

1. Comparta los paquetes que quiera convertir en reutilizables. Para ello haga clic con el botón derecho en el paquete o en el perfil y seleccione **Subproyecto | Compartir paquete** en el menú contextual.
2. Guarde el proyecto en un directorio desde el que más tarde pueda incluirlo en un subproyecto (*véase Incluir subproyectos*).

De momento hemos creado un perfil pero no lo hemos añadido ni aplicado a ningún paquete. Al aplicar un perfil a un paquete todos los mecanismos de extensión de ese perfil (como estereotipos, tipos de datos, etc.) pasan a estar disponibles para elementos del paquete.

Para aplicar un perfil personalizado a un paquete:

1. Cree un proyecto de UModel nuevo o abra uno que ya existe.
2. Elija una de estas opciones:
   a. Cree un perfil personalizado en el proyecto que ya existe como se explica más arriba.
   b. Incluya un perfil personalizado desde un proyecto que ya existe usando el comando de menú **Proyecto | Incluir subproyecto**. Recuerde que debe compartir el perfil entero o sus paquetes para que sean reutilizables (véase Compartir paquetes y diagramas).
3. Haga clic con el botón derecho en el perfil y seleccione **Diagrama nuevo | Diagrama de perfil** en el menú contextual.
4. Añada paquetes y el perfil personalizado al diagrama.
5. Dibuje una relación **ProfileApplication** desde el paquete hasta el perfil. Por ejemplo, el diagrama de perfil siguiente ilustra una relación **ProfileApplication** entre el paquete **BankView** y el perfil Java creado en UModel. Como se ilustra a continuación, las aplicaciones del perfil se muestran como flechas discontinuas desde el paquete hasta el perfil aplicado, junto con la palabra clave `<<apply>>`.



9.2.7.2 Crear estereotipos

Al modelar proyectos con cualquiera de los perfiles integrados de UModel (como C#, Java, VB.NET, esquema XML, base de datos, etc.) en principio no debería tener que crear estereotipos personalizados, sino que puede

aplicar los estereotipos existentes a los elementos de su modelo, como se describe en el apartado Aplicar estereotipos.

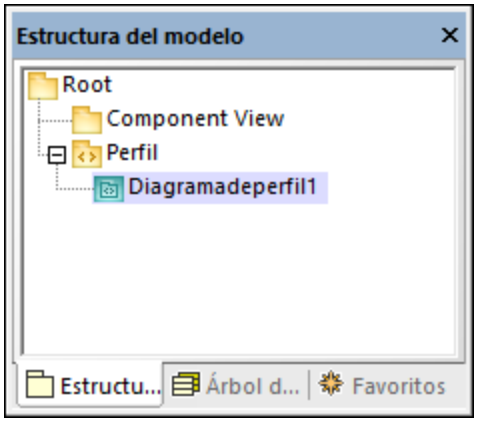
También puede añadir iconos personalizados o personalizar su aspecto en base al estereotipo aplicado creando estereotipos personalizados. Para ello debe cumplir estas condiciones:

- Los estereotipos deben pertenecer a un perfil o un paquete dentro del perfil. Por tanto, para crear un estereotipo primero debe crear un perfil (o un paquete dentro de un perfil existente).
- Después de crear el perfil debe aplicarlo al paquete en el que necesita usar los estereotipos personalizados, como se describe en el apartado Crear y aplicar perfiles personalizados.

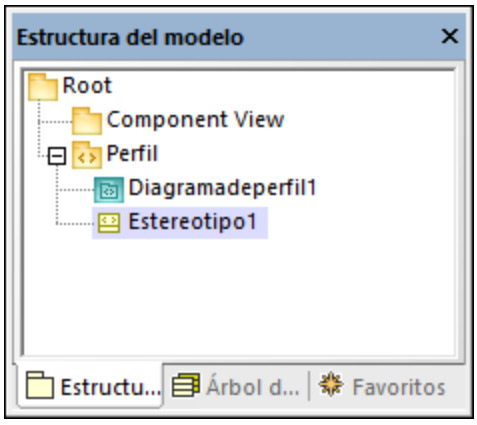
Una vez haya creado un perfil puede empezar a añadir estereotipos al mismo. Puede hacerlo directamente en la ventana *Estructura del modelo* o desde un diagrama de perfil. Si quiere crear estereotipos desde un programa de perfil, asegúrese de que el diagrama pertenece a (se crea en) un perfil o un paquete dentro del paquete, como se muestra a continuación.

Para crear un estereotipo:

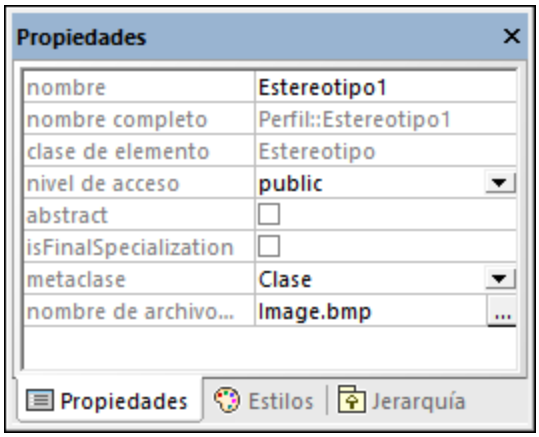
1. Si no lo ha hecho ya, cree un perfil (véase *Creating and Applying Custom Profiles*).
2. También puede hacer clic con el botón derecho en el perfil y seleccionar **Diagrama nuevo | Diagrama de perfil** en el menú contextual para crear un diagrama de perfil nuevo bajo el perfil actual. Así podrá visualizar en un mismo sitio todos los estereotipos, tipos de datos y otros elementos que añada más adelante al perfil.



3. Haga clic con el botón derecho en la ventana *Estructura del modelo* y seleccione **Elemento nuevo | Estereotipo** en el menú contextual.



4. Otra opción es configurar las propiedades del estereotipo en la ventana *Propiedades*. Por ejemplo, si cambia la **metaclase** del estereotipo a "Clase", entonces el estereotipo solo se aplicará a las clases. También puede definir un icono personalizado para el estereotipo haciendo clic en el botón de **puntos suspensivos** ... que hay junto al **nombre de archivo del icono**.



Notas

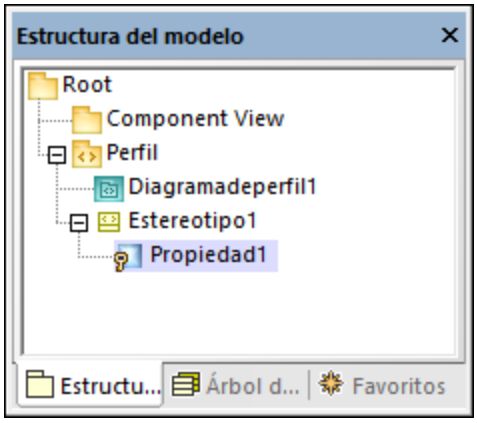
- Si la ruta de la imagen es relativa, debe ser relativa a la carpeta del proyecto de UModel.
- Para usar iconos personalizados con fondo transparente, establezca el color de fondo al valor RGB 82,82,82.
- Para mostrar estereotipos para relaciones de asociación debe establecer la propiedad **Mostrar estereotipos de memberEnd** en "true", en la ventana *Estilos*.

Añadir atributos a un estereotipo (propiedades)

El estereotipo que hemos creado en el paso anterior es muy simple y no tiene ningún atributo (propiedades) asociado, pero se le pueden añadir. Esas propiedades se convertirán en valores etiquetados cuando aplique el estereotipo a algún elemento en el futuro.

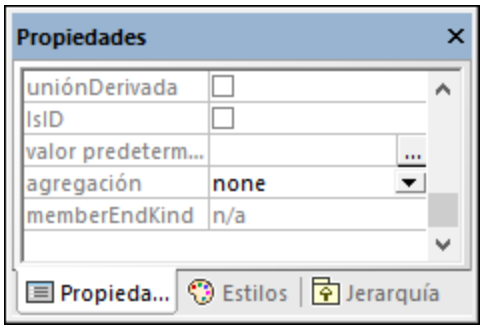
Para añadir atributos (propiedades) a un estereotipo:

1. Haga clic en el estereotipo, en la ventana *Estructura del modelo* o en el diagrama.
2. Elija una opción
   a. Haga clic con el botón derecho en el estereotipo y seleccione **Nuevo | Propiedad** en el menú contextual.
   b. Pulse la tecla **F7**.



Puede definir el tipo de datos de cada propiedad desde la ventana *Propiedades* seleccionando un valor de la lista **tipo**. Se puede seleccionar cualquier tipo de datos que se haya definido previamente en el mismo perfil como estereotipo. Si el perfil todavía no contiene ningún tipo de datos puede definir uno haciendo clic con el botón derecho en el diagrama de perfil y seleccionando **Nuevo | Tipo de datos** en el menú contextual.

Para definir el valor predeterminado de una propiedad, introduzca ese valor en el campo *predeterminado* de la ventana *Propiedades*. Por ejemplo, la propiedad del estereotipo de la imagen siguiente tiene como valor predeterminado "0":



El tipo de datos de un atributo de estereotipo (propiedad) también puede ser una enumeración (*véase* *Ejemplo: crear y aplicar estereotipos*).

9.2.7.3 Ejemplo: crear y aplicar estereotipos

En este ejemplo explicamos paso a paso el proceso de creación de estereotipos, que permite:

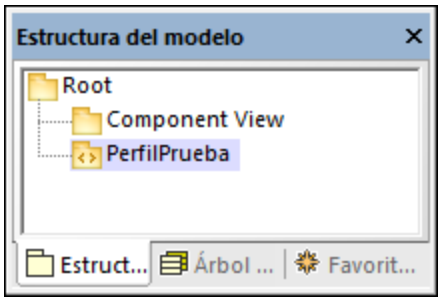
- Crear un estereotipo
- Crear atributos de estereotipo (propiedades) que se convierten en valores etiquetados al aplicarlos a un elemento
- Definir un atributo de estereotipo en una enumeración
- Definir un valor predeterminado para un atributo de estereotipo
- Aplicar el estereotipo a elementos del modelo.

En este ejemplo también se incluye un proyecto de ejemplo llamado **StereotypesDemo.ump**, que está disponible en **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Tutorial**. Si sigue estas instrucciones al pie de la letra creará un proyecto parecido.

Crear un perfil nuevo

Como ya hemos mencionado, un estereotipo debe pertenecer a un perfil, por lo que primero vamos a crear ese perfil.

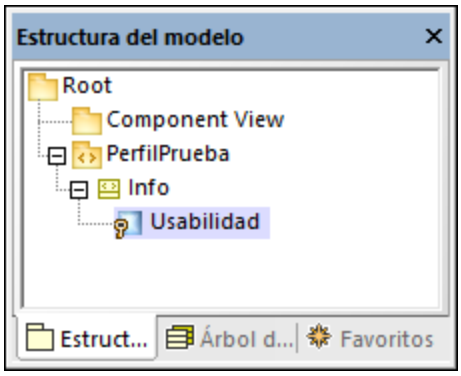
1. Cree un proyecto de UModel nuevo.
2. Haga clic con el botón derecho en el paquete "Root" y añada un perfil nuevo seleccionando **Elemento nuevo | Perfil** en el menú contextual.
3. Cambie el nombre del perfil nuevo a "PerfilPrueba".



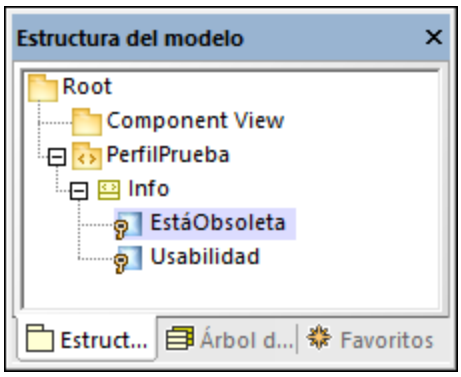
Crear un estereotipo

Para este tutorial vamos a crear un estereotipo con dos atributos: "Usabilidad" y "EstáObsoleta". Vamos a definir el atributo "EstáObsoleta" como enumeración. La enumeración consiste en dos valores, "Sí" y "No"; el valor predeterminado es "No".

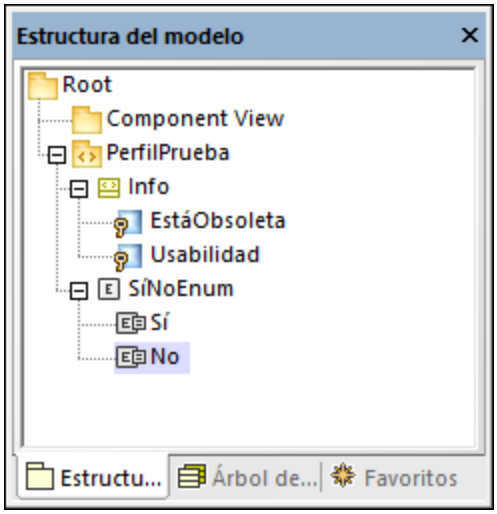
1. Haga clic con el botón derecho en el perfil y seleccione **Elemento nuevo | Estereotipo** en el menú contextual. Así añadimos un estereotipo nuevo al perfil.
2. Cambie el nombre del estereotipo nuevo a "Info".
3. Haga clic con el botón derecho en el estereotipo y seleccione **Elemento nuevo | Propiedad** en el menú contextual. Así añadimos una propiedad nueva.
4. Cambie el nombre de la propiedad a "Usabilidad".



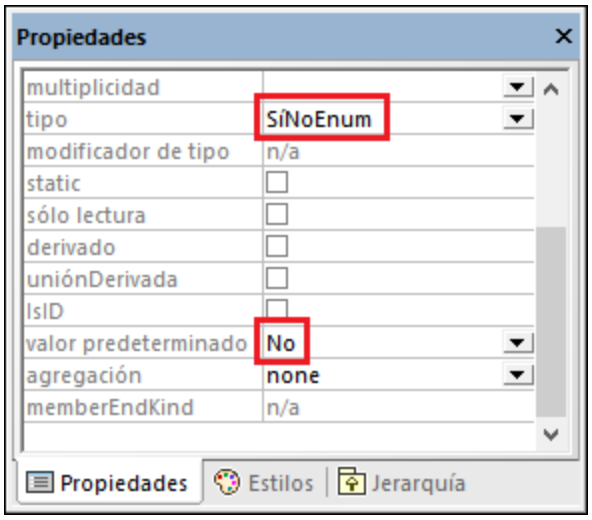
5. Repita los pasos anteriores para crear otro proyecto nuevo llamado "EstáObsoleta".



6. Haga clic con el botón derecho en "PerfilPrueba" y seleccione **Elemento nuevo | Enumeración** en el menú contextual. Cambie el nombre de la enumeración a "SíNoEnum".
7. Haga clic con el botón derecho en la enumeración y seleccione **Elemento nuevo | LiteralDeEnumeración** en el menú contextual. Cambie el nombre del literal de enumeración a "Sí".
8. Repita el paso anterior y cree un literal de enumeración llamado "No".



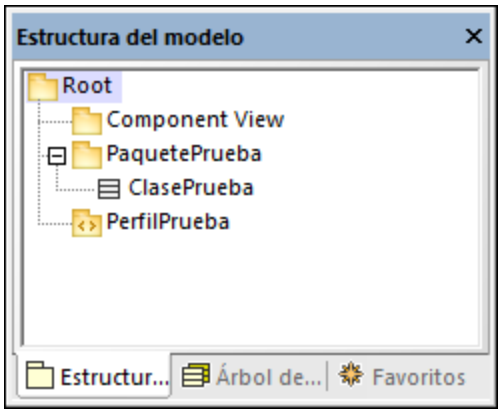
9. Haga clic en la propiedad "EstáObsoleta" y cambie su tipo a `SíNoEnum`. Ahora establezca la propiedad **predeterminada** a "No".



Crear un paquete nuevo

Para ilustrar cómo se pueden usar los estereotipo personalizados vamos a crear un paquete simple que contenga solamente una clase.

1. Haga clic con el botón derecho en el paquete "Root" y añada un paquete nuevo seleccionando **Elemento nuevo | Paquete** en el menú contextual.
2. Cambie el nombre del paquete nuevo a "PaquetePrueba".
3. Añada una clase al paquete (en este ejemplo, "ClasePrueba").

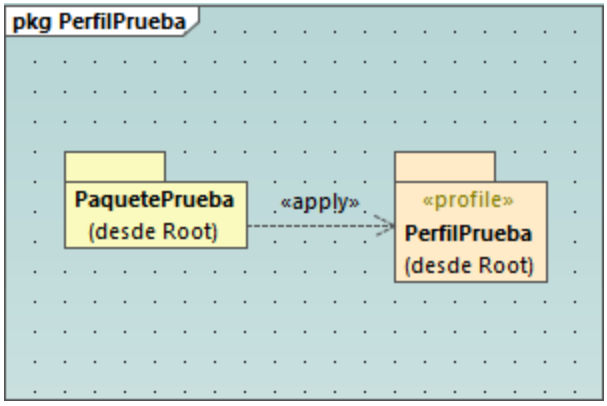


Aplicar el perfil a un paquete

Como explicábamos en el paso 1, el estereotipo se creó dentro de un perfil. En este paso vamos a aplicar el perfil a un paquete para que el estereotipo se vuelva "visible" para el paquete.

1. En la ventana *Estructura del modelo* haga clic con el botón derecho en "PerfilPrueba" y seleccione **Diagrama nuevo | Diagrama de perfil** en el menú contextual.
2. Arrastre los paquetes "PaquetePrueba" y "PerfilPrueba" desde la ventana *Estructura del modelo* hasta el diagrama.

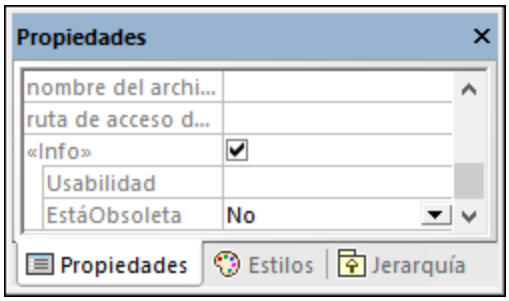
3. Haga clic en el botón de la barra de herramientas y dibuje una relación **ProfileApplication** desde el paquete hasta el perfil.



Aplicar el estereotipo a clases

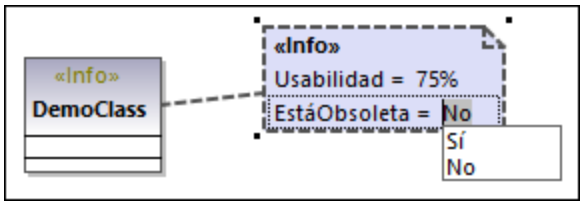
Ahora puede aplicar el estereotipo a una clase.

1. Haga clic con el botón derecho en "PaquetePrueba" y seleccione **Diagrama nuevo | Diagrama de clases** en el menú contextual.
2. Arrastre la clase "ClasePrueba" hasta el diagrama.
3. Haga clic en la clase y seleccione el estereotipo «Info» en la ventana *Propiedades*. Observe que la propiedad "EstáObsoleta" ya contiene su valor predeterminado.



4. Introduzca un valor para la propiedad "Usabilidad" ("75%" en este ejemplo).

Ahora la clase del diagrama tiene una sección "Valores etiquetados" en la que aparecen los atributos del estereotipo y sus valores. Puede cambiar estos valores desde la ventana *Propiedades* o directamente desde el diagrama.



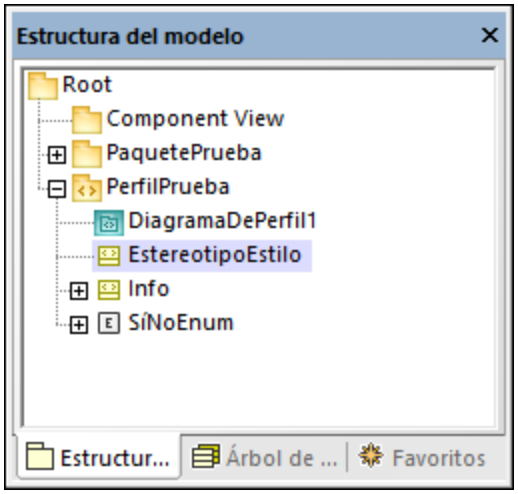
9.2.7.4 Ejemplo: personalizar iconos y estilos

En este ejemplo explicamos cómo personalizar el aspecto de una clase en UModel con la ayuda de estereotipos. Aprenderá a añadir iconos personalizados a elementos y a cambiar el estilo de todos los elementos que usen el mismo estereotipo.

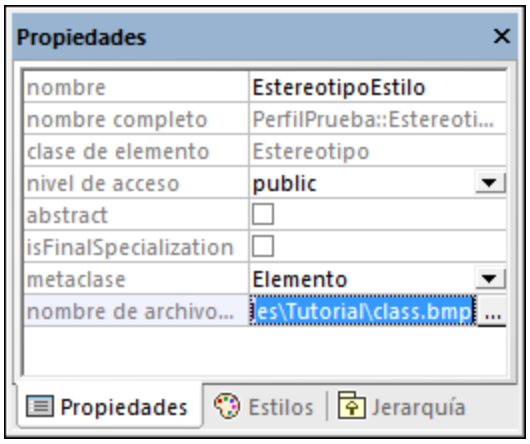
La clase que vamos a personalizar en este ejemplo está en el proyecto **StereotypesDemo.ump**, que puede encontrar en **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Tutorial**. Este es un simple proyecto de ejemplo que incluye un perfil personalizado dentro del cual vamos a crear el estereotipo. Para ver un ejemplo de cómo crear perfiles y estereotipos desde cero consulte Ejemplo: crear y aplicar estereotipos.

Primero vamos a crear el estereotipo que necesitamos para los estilos:

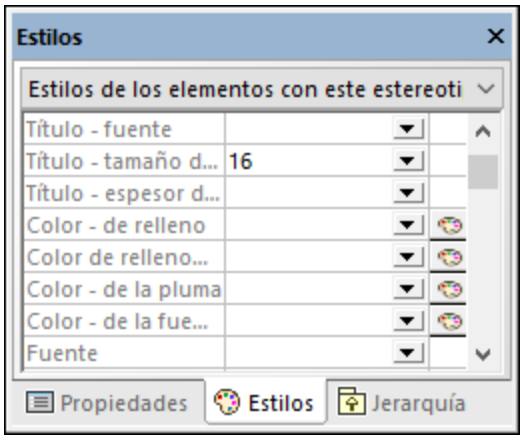
1. Abra el proyecto **StereotypesDemo.ump**.
2. En la *Estructura del modelo*, haga clic con el botón derecho en el perfil "PerfilPrueba" y seleccione **Elemento nuevo | Estereotipo** en el menú contextual.
3. Cambie el nombre del estereotipo a "EstereotipoEstilo".



Para añadir una imagen personalizada al estereotipo, haga clic en el estereotipo y después en el botón de **puntos suspensivos** ... que hay junto a la propiedad del **nombre de archivo del icono** en la ventana *Propiedades*. Seleccione la siguiente imagen de ejemplo: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Tutorial\class.bmp**.

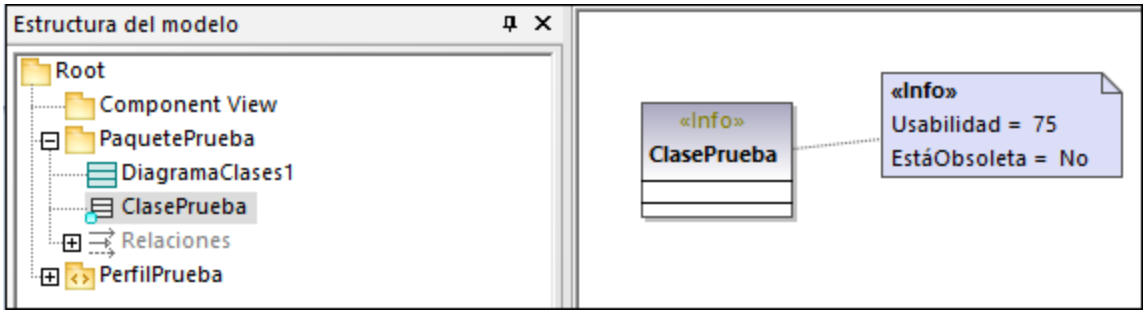


A continuación, haga clic en la pestaña *Estilos* de la ventana *Propiedades*. Seleccione **Estilos de los elementos con este estereotipo** de la lista superior y cambie la propiedad **Tamaño de la fuente de encabezado** a "16".

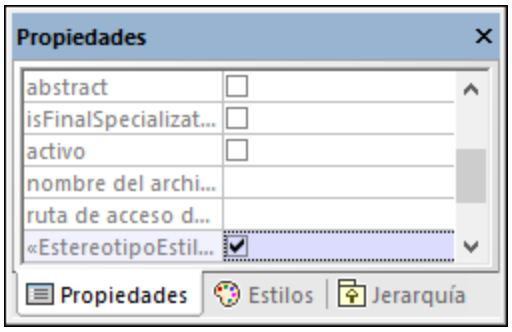


Por último, aplique el estereotipo a una clase.

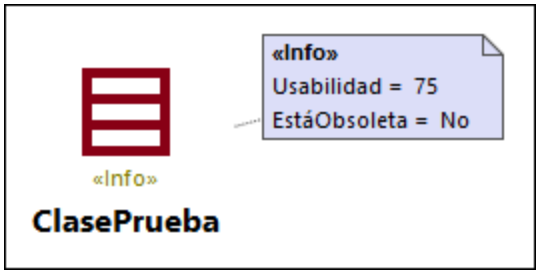
1. Abra el diagrama de clases "DiagramaClases1". Lo encontrará bajo el paquete "DemoPackage", en la vista *Estructura del modelo*.



2. Haga clic en la clase "ClasePrueba" y, en la ventana *Propiedades*, marque la casilla **«EstereotipoEstilo»**.

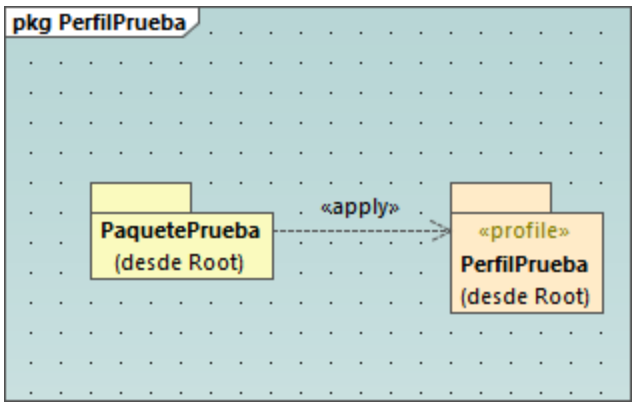


El aspecto de la clase en el diagrama cambia conforme al estereotipo aplicado:



Observaciones

El proyecto de ejemplo contiene el diagrama de perfil "DiagramaPerfil1". En este diagrama, observe que "PerfilPrueba" se aplica a "PaquetePrueba" con la relación **ProfileApplication**  . Esto hace que el estereotipo esté disponible para el paquete (*véase también el apartado Crear y aplicar perfiles personalizados*).



En este ejemplo hemos explicado cómo cambiar el aspecto de los elementos usando estereotipos. Puede usar la misma técnica en otros proyectos. Recuerde que debe aplicar al paquete de destino el perfil en el que haya creado el estereotipo, como se indica más arriba.

9.3 Otros diagramas

Estos son los demás tipos de diagramas compatibles con **UModel Enterprise Edition**:

Esquemas XML
BPMN (Business Process Modeling Notation)
Diagramas SysML
Diagramas de base de datos

9.3.1 Diagramas de esquema XML

Sitio wen de Altova: Esquemas XML en UML

UModel permite importar y generar esquemas W3C XML, así como su ingeniería de código e ingeniería inversa. En el caso de los esquemas XML, "ingeniería de código e ingeniería inversa" significa que puede importar un esquema (o varios esquemas de un directorio) en UModel, ver o modificar el model y escribir los cambios en el archivo del esquema. Si sincroniza datos del modelo con el archivo de esquema, el modelo siempre sobrescribe al archivo de esquema.

Nota: el esquema XML debe ser válido antes de importarlo en UModel. los esquemas XML no se validan cuando los crea o importa e UModel, ni cuando ejecuta una comprobación de sintaxis en un proyecto. Sin embargo, al importar un esquema XML, UModel comprueba si tiene el formato correcto.

Los diagramas de esquema XML presenta componentes del esquema en notación UML. Por ejemplo, los tipos simples aparecen en UModel como tipos de datos con el estereotipo «`simpleType`». Los tipos complejos aparecen como clases con el estereotipo «`complexType`». Los detalles del esquema se representan como Valores etiquetados, mientras que las anotaciones aparecen como comentarios. Para ver en una tabla cómo todos los elementos del esquema XML corresponden a elementos UML, consulte Correspondencias con XML Schema.

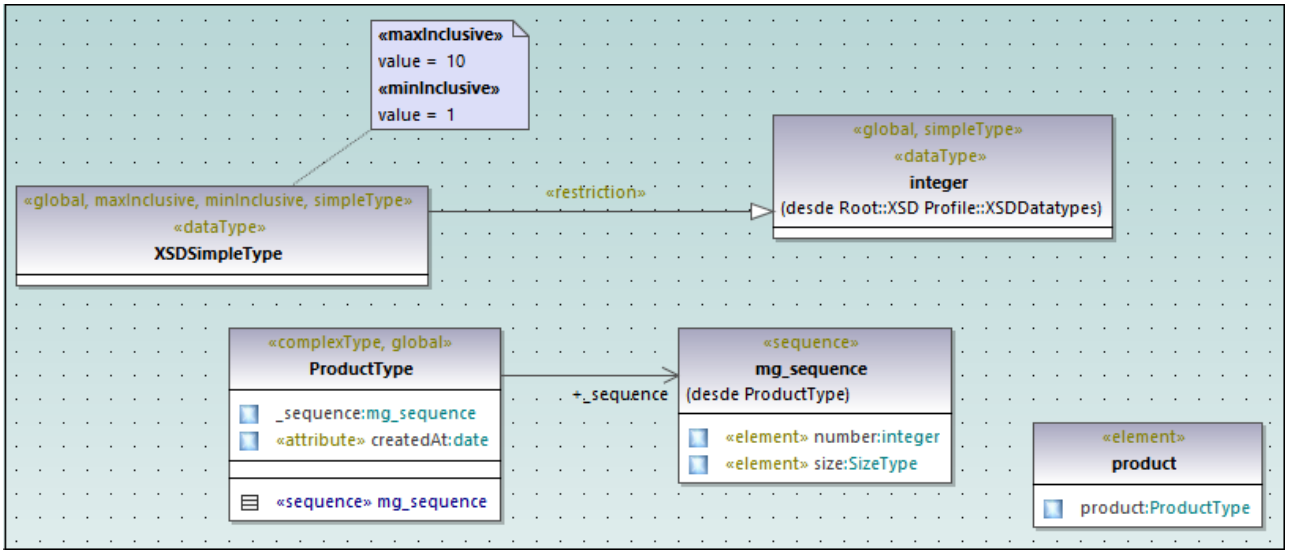


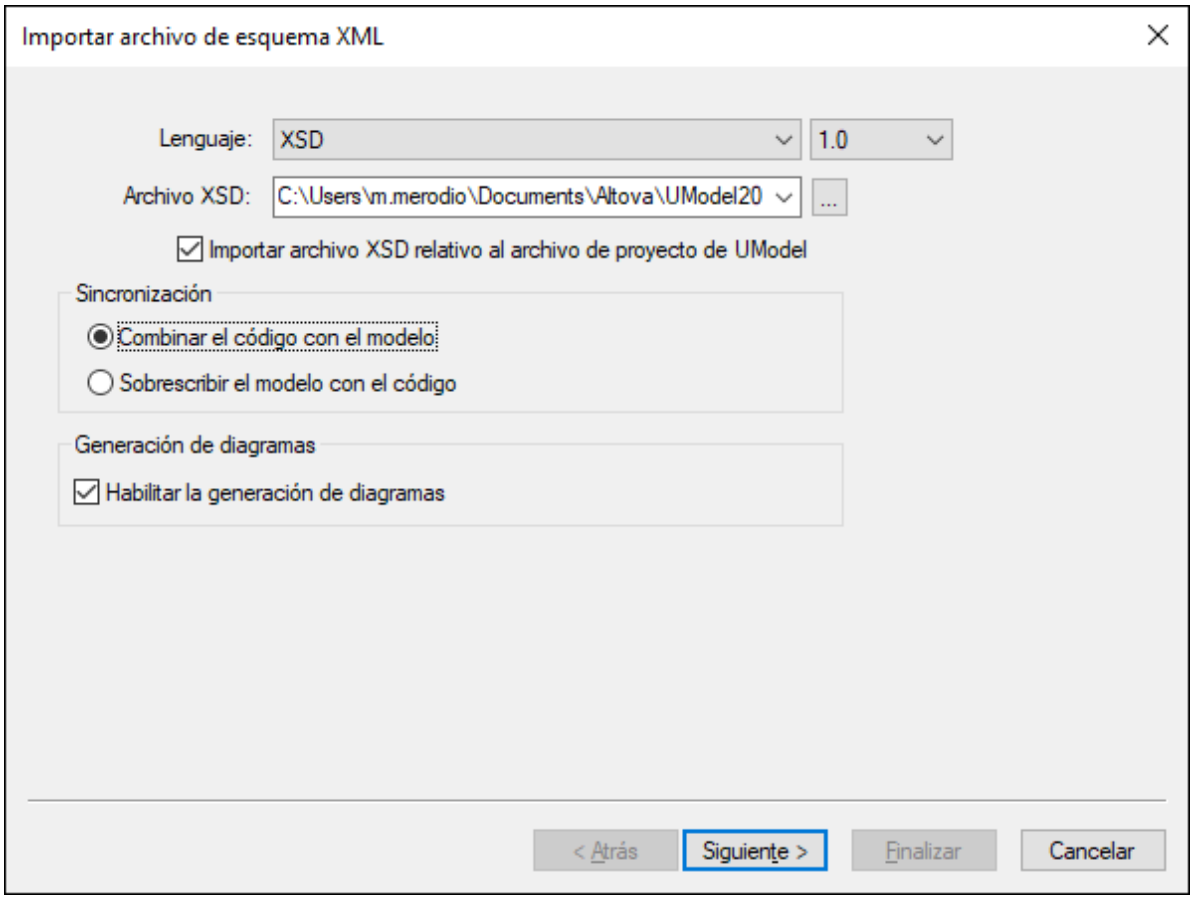
Diagrama XML de ejemplo

9.3.1.1 Importar esquemas XML

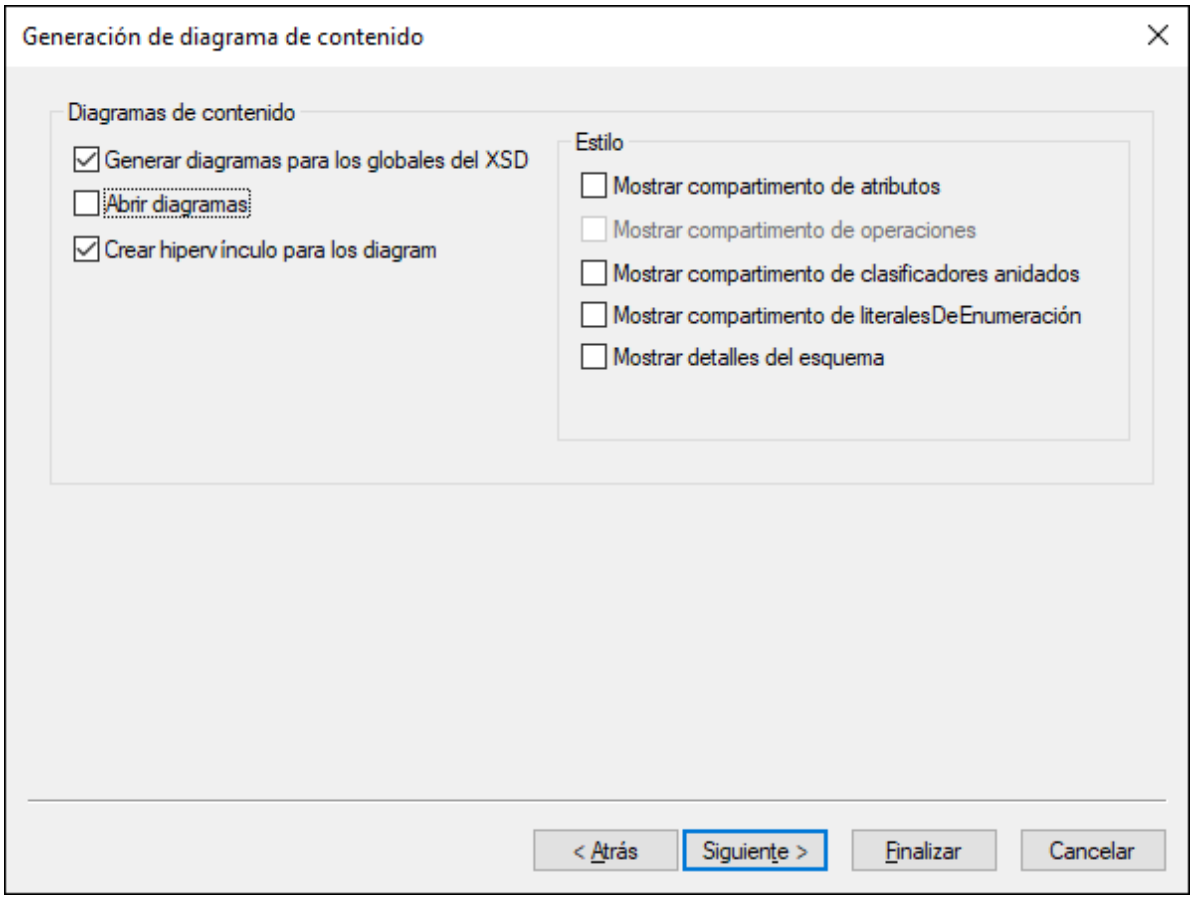
Puede escoger entre importar un solo archivo de esquema en UModel o todos los archivos de esquema de un directorio. Si un esquema incluye o importa otros esquemas, estos también se importan en el modelo.

Para importar un único esquema XML:

1. Seleccione el comando de menú **Proyecto | Importar archivo de esquema XML**.
2. Haga clic en **Examinar** y seleccione el esquema de origen que quiere importar. En este ejemplo usaremos el esquema: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Tutorial\OrgChart.xsd**.

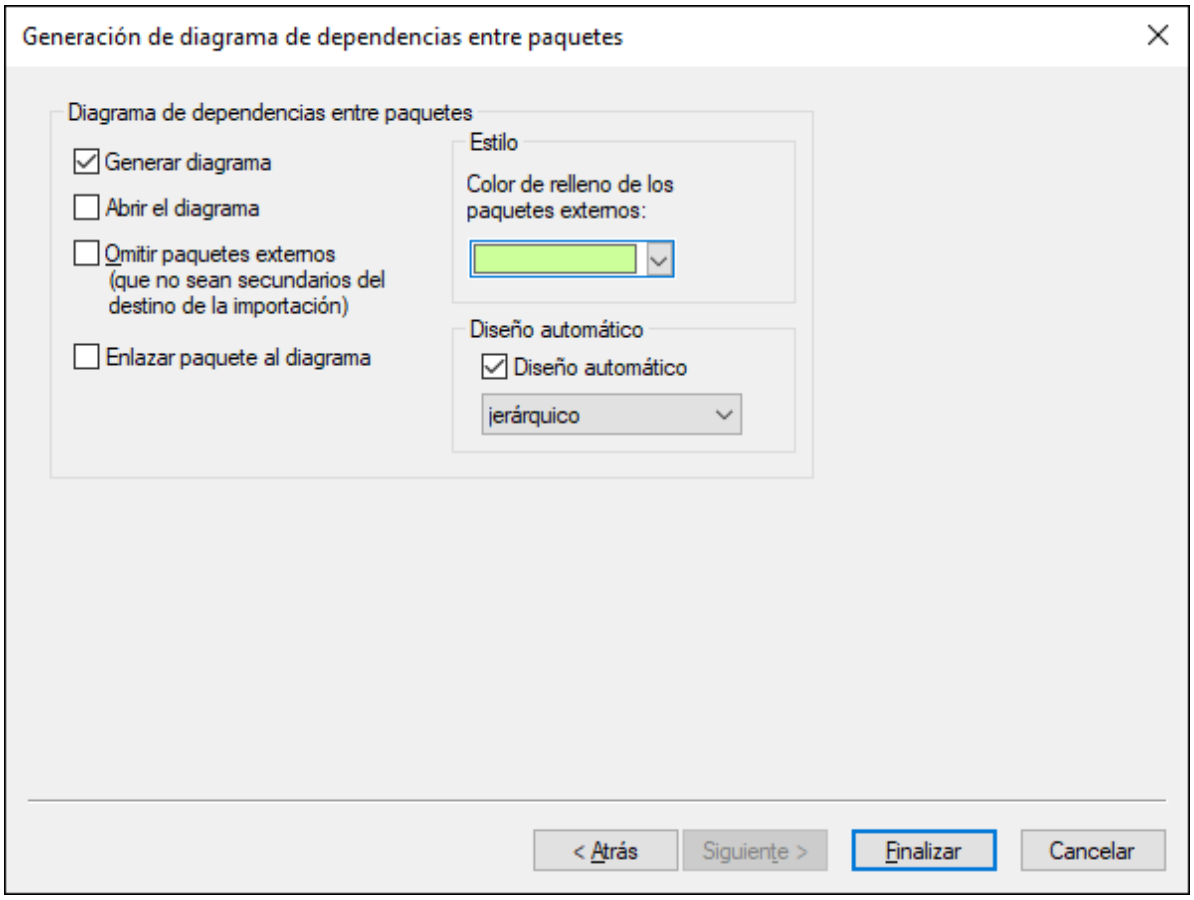


3. Para generar diagramas a partir del esquema debe marcar la casilla **Habilitar la generación de diagramas**; después haga clic en **Siguiente**.



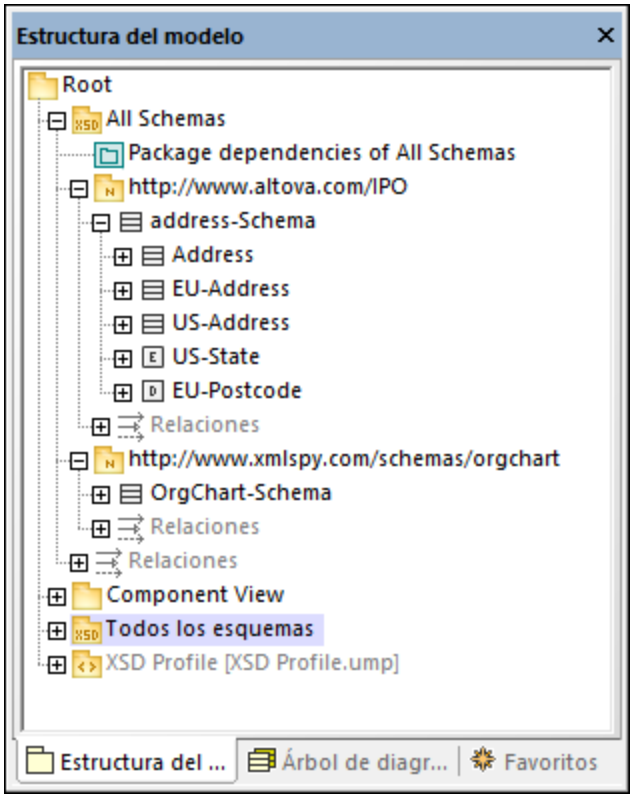
4. Para crear un diagrama aparte para cada uno de los componentes globales del esquema, como en este ejemplo, marque la opción *Generar diagramas para los globales del XSD*. Para abrir todos los diagramas generados después de la importación marque la opción *Abrir diagramas*. Las opciones del apartado "Estilo" permiten definir los compartimentos que aparecen por defecto en diagramas de cada componente del esquema. La opción *Mostrar detalles del esquema como valores etiquetados* muestra los detalles del esquema como Valores etiquetados.

5. Haga clic en **Siguiente**. Para generar un diagrama de dependencias de paquetes como el de este ejemplo, marque la casilla *Generar diagrama*.



6. Haga clic en **Finalizar**.

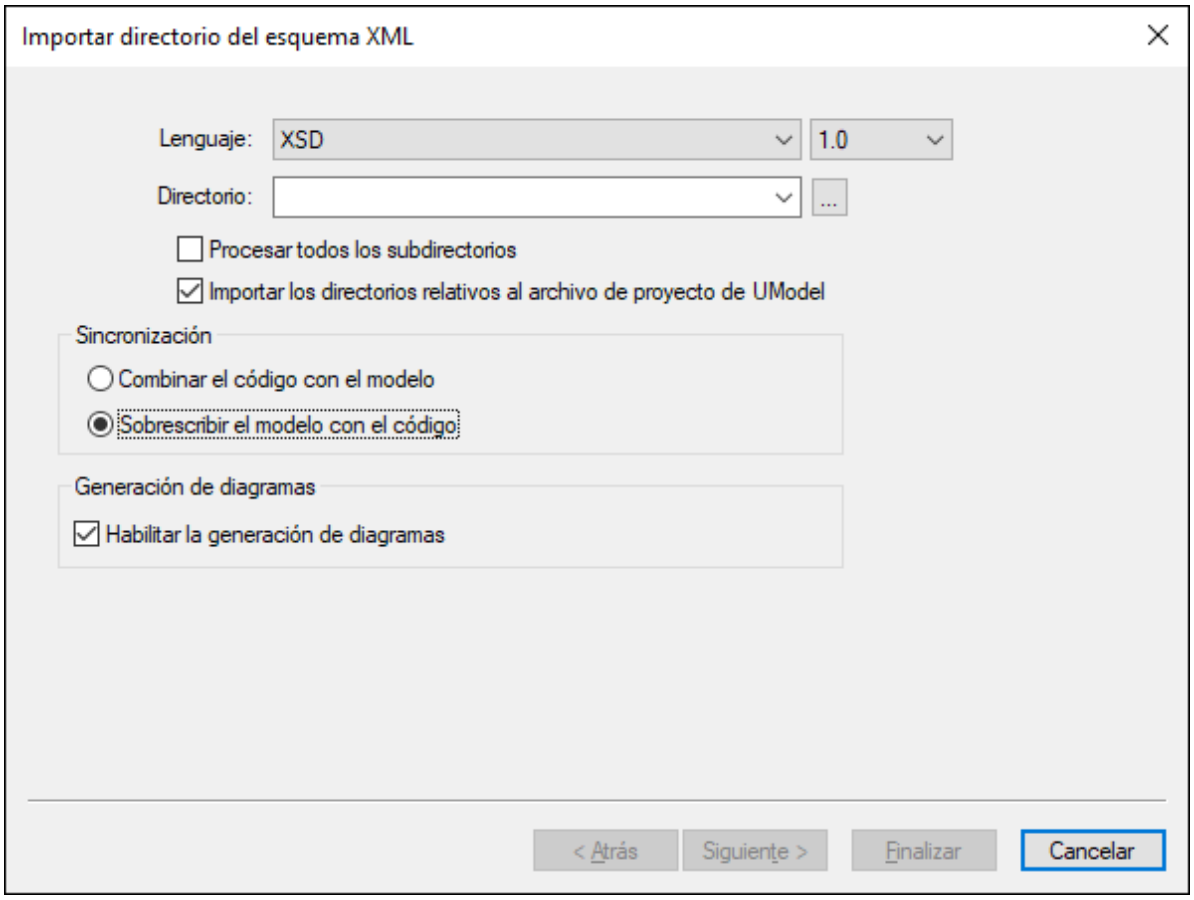
Una vez haya terminado de importar el esquema se crea un paquete nuevo llamado `Todos los esquemas`, que se convierte automáticamente en la raíz de espacio de nombres XSD. El esquema `OrgChart.xsd` de este ejemplo importa los tipos desde otro espacio de nombres, en concreto del esquema `ipo.xsd`. En consecuencia, después de la importación los dos esquemas aparecen en la ventana Estructura del modelo, cada uno bajo su propio espacio de nombres:



Si marcó la casilla *Generar diagramas para los globales del XSD*, todos los componentes globales XSD generan un diagrama de esquema XML y todos esos diagramas aparecen bajo los paquetes de espacio de nombres respectivos, como el diagrama "Address (complexType)" de la imagen anterior.

Para importar varios esquemas XML:

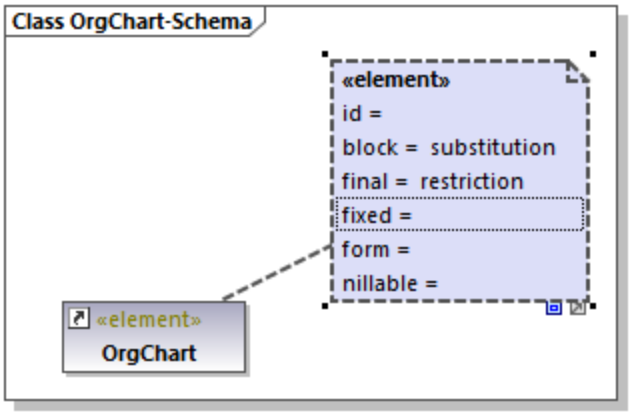
1. Seleccione el comando de menú **Proyecto | Importar directorio del esquema XML**.



2. Para importar esquemas de todos los subdirectorios del directorio seleccionado, marque la casilla *Procesar todos los subdirectorios*. El resto del proceso de importación es idéntico al proceso de importar un único esquema XML.

Cambiar la visualización de los valores etiquetados

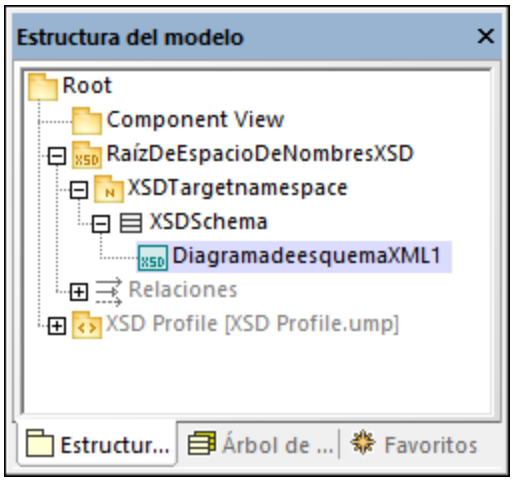
Una vez haya importado un esquema XML, hay detalles que pueden aparecer como valores etiquetados en el diagrama si marcó la casilla *Mostrar detalles del esquema como valores etiquetados* durante la importación.



Puede configurar estos detalles para que aparezcan en el diagrama o permanezcan ocultos. Para ello haga clic con el botón derecho en el elemento en cuestión y seleccione **Valores etiquetados | <opción>** en el menú contextual. Puede configurar la visualización de los valores etiquetados tanto a nivel individual como a nivel de todo el proyecto. Para más información consulte Mostrar u ocultar valores etiquetados.

9.3.1.2 Modelar esquemas XML

En UModel, los proyectos nuevos de esquema XML tienen la estructura de la imagen siguiente. Esta estructura se crea automáticamente la primera vez que añade un diagrama de esquema XML a un proyecto nuevo de UModel.



Los paquetes "Root" y "Component View" son comunes a todos los proyectos de UModel y no se pueden eliminar. "Root" es el nivel más alto bajo el que se añaden el resto de paquetes y "Component View" se usa para ingeniería de código (en este caso para importar o generar archivos de esquema).

El paquete "XSDNamespaceRoot" incluye todos los espacios de nombres usados en los esquemas. Para convertir un paquete en una raíz de espacio de nombres XSD haga clic con el botón derecho en él y seleccione **Ingeniería | Establecer como raíz de espacio de nombres XSD** en el menú contextual. Si importa en el proyecto un esquema XML que ya existe, este paquete se llamará "Todos los esquemas" por defecto.

El paquete "XSDTargetnamespace" es un espacio de nombres de esquema XML. Pueden existir muchos espacios de nombres de este tipo bajo una misma raíz de espacio de nombres XSD. Para convertir un paquete en un espacio de nombres primero seleccione el paquete y después la propiedad «namespace» (estereotipo) en la ventana Propiedades.

"XSDSchema" es un esquema o, en términos UML, una clase para la que se ha seleccionado la propiedad «schema» (estereotipo) en la ventana Propiedades.

XMLSchemaDiagram1 es el diagrama que describe el modelo del esquema. Puede crear diagramas de esquema XML bajo una raíz de espacio de nombres XSD, un espacio de nombres de esquema ML o un esquema XML. En el ejemplo de la imagen anterior el diagrama se creó bajo un esquema XML.

El **Perfil XSD** admite todos los tipos y las estructuras requeridos para trabajar con XML Schema en el proyecto. Si el proyecto no tiene este perfil, la aplicación le pedirá que lo incluya siempre que cree un

diagrama de esquema XML nuevo. También puede añadir el perfil XSD a un proyecto de forma explícita, consulte Aplicar perfiles de UModel.

Crear diagramas de esquema XML

Para crear un diagrama de esquema XML nuevo:

1. Elija una opción:
   a. Haga clic con el botón derecho en un paquete en la ventana Estructura del modelo y seleccione **Diagrama de esquema XML** en el menú contextual.
   b. Haga clic con el botón derecho en "Diagramas" o Diagramas de esquema XML" en la ventana Estructura del modelo y seleccione **Diagrama nuevo | Diagrama de esquema XML** en el menú contextual. Se abre un cuadro de diálogo donde debe seleccionar el propietario del diagrama. Seleccione el paquete donde se debe guardar el diagrama y haga clic en **Aceptar**.
2. Si el proyecto actual de UModel no incluye el perfil XSD, se abre un cuadro de diálogo que le pide que lo incluya. Haga clic en **Aceptar** para incluir el perfil XSD en el perfil actual; consulte también Aplicar perfiles de UModel.

Añadir elementos de esquema XML nuevos

Para añadir elementos de esquema XML a un diagrama:

- Haga clic en un botón concreto de la barra de herramientas y después haga clic dentro del diagrama de esquema XML.

Para insertar varios elementos del mismo tipo mantenga pulsada la tecla **Ctrl** y haga clic en los elementos del diagrama que quiera insertar.

> Como hemos explicado, los diagramas de esquema XML se pueden crear a varios niveles en la estructura del proyecto. Si el diagrama está en un nivel en el que no se puede colocar un elemento en concreto, hay botones de la barra de herramientas que no se pueden usar y que mostrarán información en vez de añadir el elemento.

Estos son todos los botones de la barra de herramientas y su función.

	targetMamespace XSD	Añade un espacio de nombres XSD de destino. Es útil si el diagrama se creó directamente bajo una raíz de espacio de nombres XSD.
	schema XSD	Añade una definición de esquema XML (XSD). Es útil si el diagrama se creó directamente bajo un espacio de nombres XSD de destino.
	Element (global) XSD	Añade un elemento global al diagrama. Al añadir un elemento se genera automáticamente una propiedad con el mismo nombre que el elemento en el compartimento del atributo. Defina el tipo de la propiedad para definir el tipo del elemento.

	group XSD	Añade un grupo de modelo al diagrama.
	complexType XSD	Añade un tipo complejo global al diagrama. En términos UML, se trata de una clase a la que se han aplicado los estereotipos «global» y «complexType».
	complexType XSD (simpleContent)	Añade un tipo complejo global con contenido simple. En términos UML, se trata de un tipo de datos al que se han aplicado los estereotipos «global», «complexType» y «simpleContent».
	simpleType XSD	Añade un tipo simple global.
	list XSD	Añade un tipo de lista.
	union XSD	Añade un tipo de unión.
	enumeración XSD	Añade una enumeración.
	Attribute (global) XSD	Añade un atributo.
	AttributeGroup XSD	Añade un grupo de atributo.
	notation XSD	Añade un tipo de notación.
	import XSD	Añade una relación de importación.
	include XSD	Añade una relación inclusión.
	redefine XSD	Añade una relación de redefinición.
	restriction XSD	Añade una relación de restricción.
	extension XSD	Añade una relación de extensión.
	substitution XSD	Añade una relación de sustitución.
	Comentario	Añade un comentario. Los comentarios se convierten en anotaciones cuando se genera el archivo de esquema a partir del modelo. Para indicar el tipo de anotación seleccione el estereotipo correspondiente en la ventana Propiedades.
	Nota	Añade una nota explicativa.
	Comentario/Enlace de nota	Vincula una nota a otros elementos del diagrama.

Para ver unas instrucciones de modelado paso a paso consulte Ejemplo: crear y generar un esquema XML.

9.3.1.3 Ejemplo: crear y generar un esquema XML

Este ejemplo aprenderá a modelar un esquema XML nuevo con UModel paso a paso. Una vez haya modelado el esquema de forma visual con UML podrá generar el archivo del esquema. Más concretamente, aprenderá a crear y generar el esquema **product.xsd** que se ve a continuación.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.altova.com/umodel"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:prod="http://www.altova.com/umodel">
   <xs:simpleType name="SizeType">
      <xs:restriction base="xs:integer">
         <xs:maxInclusive value="10"/>
         <xs:minInclusive value="1"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="ProductType">
      <xs:sequence>
         <xs:element name="number" type="xs:integer">
         </xs:element>
         <xs:element name="size" type="prod:SizeType">
         </xs:element>
      </xs:sequence>
      <xs:attribute name="createdAt" type="xs:date">
      </xs:attribute>
   </xs:complexType>
   <xs:element name="product" type="prod:ProductType">
   </xs:element>
</xs:schema>
```

product.xsd

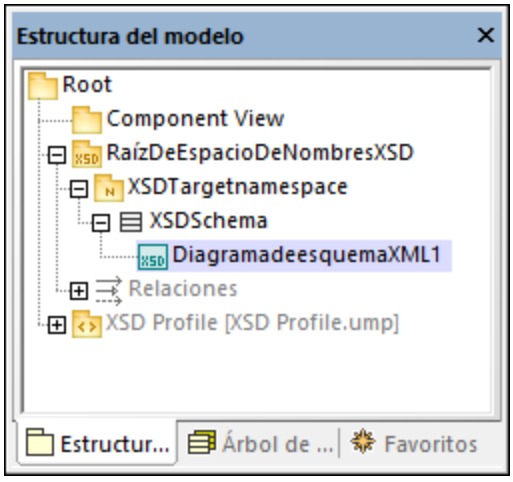
Como se aprecia en el código anterior, el esquema **product.xsd** tiene dos declaraciones de espacio de nombres:

1. El espacio de nombres de esquema XML predeterminado `http://www.w3.org/2001/XMLSchema`, que está asignado al prefijo "xs".
2. El espacio de nombres secundario `http://www.altova.com/umodel`, que está asignado al prefijo "prod", que también es el espacio de nombres de destino.

Asimismo, el esquema XML tiene un elemento global `product`, un tipo complejo `ProductType` y un tipo simple `SizeType`.

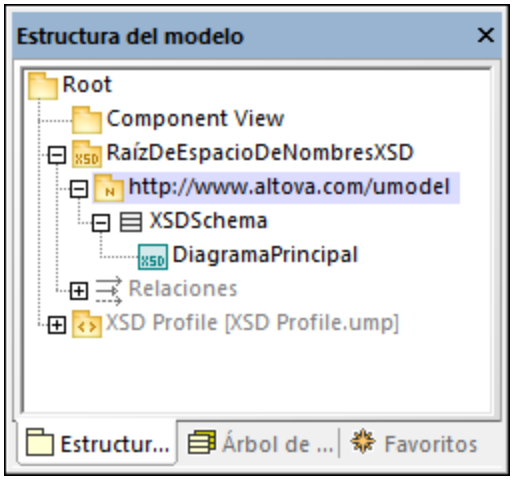
Declarar espacios de nombres y cifrar archivos

Primero debe crear un proyecto nuevo de UModel. Haga clic con el botón derecho en el paquete **Root** y seleccione **Diagrama nuevo | Diagrama de esquema XML** en el menú contextual. Cuando la aplicación le pida que incluya el perfil UModel XSD haga clic en **Aceptar**.



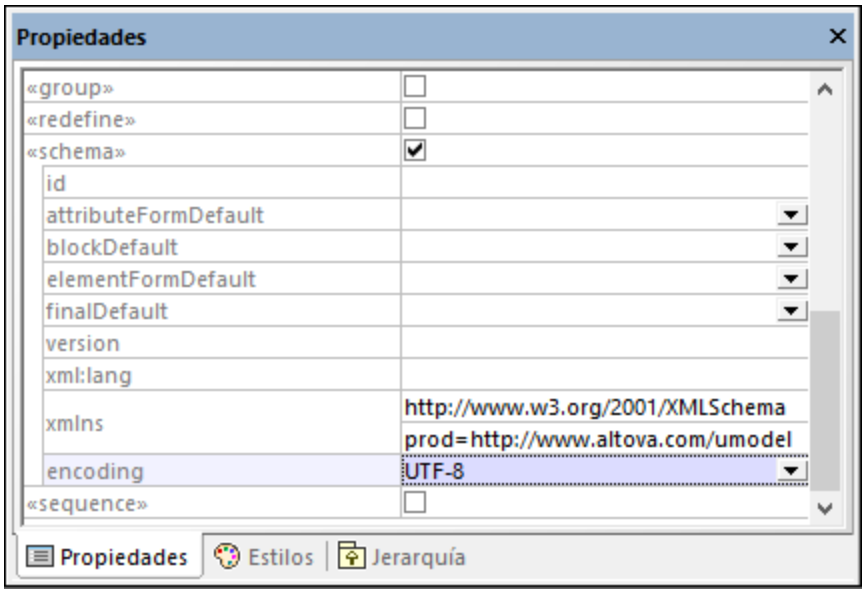
En la ventana Estructura del modelo cambie el nombre de "XMLSchemaDiagram1" a "DiagramaPrincipal". Este es el diagrama en el que crearemos la mayoría de los componentes del esquema, salvo las declaraciones de espacio de nombres.

A continuación cambie el nombre de "XSDTargetNamespace" a "http://www.altova.com/umodel" (recuerde que este es el espacio de nombres de destino requerido). De esta forma declara el espacio de nombres de destino del esquema nuevo.



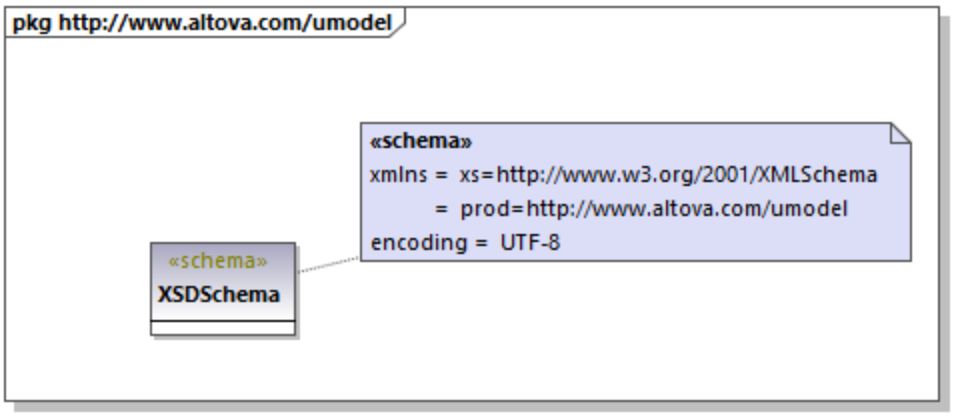
Para definir los dos espacios de nombres "xmlns" y el cifrado en UTF-8:

1. Seleccione el esquema **XSDSchema** en la Estructura del modelo.
2. En la ventana Propiedades haga clic con el botón derecho en la propiedad `xmlns` y seleccione **Agregar valor etiquetado | xlmns**.
3. Edite las propiedades `xmlns` y `encoding` como se muestra más abajo.



También puede generar rápidamente un diagrama de esquema XML nuevo a nivel del espacio de nombres que presente la misma información pero de forma visual:

1. En la Estructura del modelo haga clic con el botón derecho en el espacio de nombres "http://www.altova.com/umodel" y seleccione **Diagrama nuevo | Diagrama de esquema XML** en el menú contextual.
2. Cuando aparezca una caja de texto con el mensaje: "¿Desea agregar el "Diagrama de esquema XML" a un "Esquema XSD" nuevo?" haga clic en **No**.
3. Arrastre el esquema XML desde la Estructura del modelo hasta el diagrama.

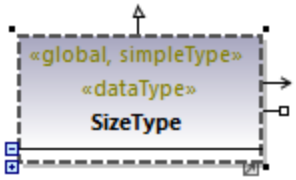


Como se ve más arriba, el espacio de nombres y el cifrado de almacenan como valores etiquetados y se pueden editar también desde la ventana del diagrama.

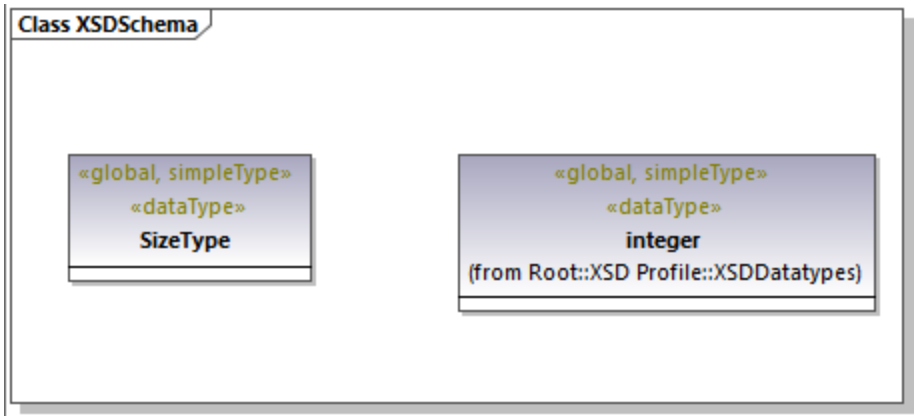
Agregar un tipo simple

Siga estos pasos para crear un tipo simple `SizeType` en el esquema XML. Este tipo restringe el tipo base `xs:integer`; por lo tanto, vamos a añadir también el tipo base al diagrama y a crear una relación de restricción.

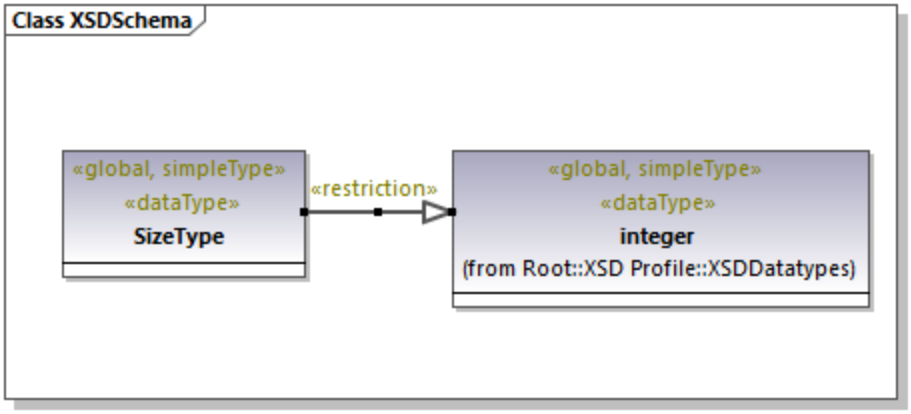
1. En la Estructura del modelo haga doble clic en **DiagramaPrincipal** para abrirlo.
2. Haga clic en el botón **simpleType XSD** de la barra de herramientas y después haga clic en el diagrama.
3. Cambie el nombre del tipo simple que acaba de añadir a `SizeType`.



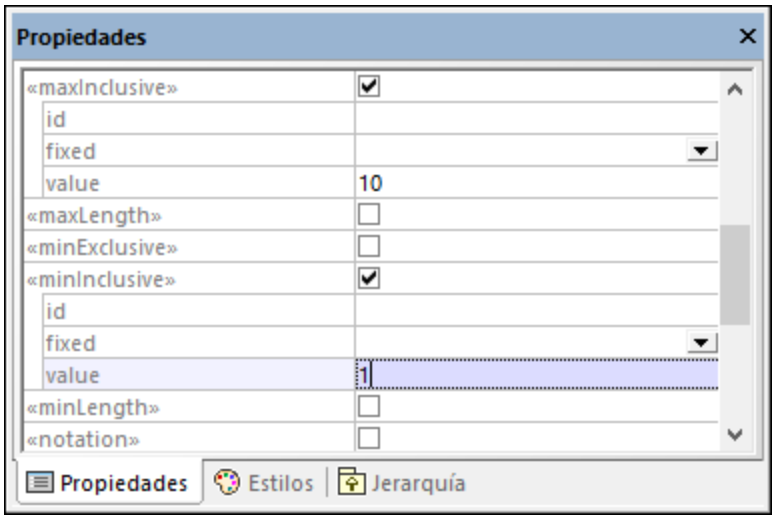
4. Haga clic dentro de la Estructura del modelo y pulse **Ctrl+F**. Esto abre el cuadro de diálogo "Buscar". Empiece a escribir "integer" para encontrar el tipo `integer` del paquete "XSDDataTypes" del perfil XSD.
5. Arrastre el tipo `integer` hasta el diagrama.



6. Haga clic en el botón **Restricción** de la barra de herramientas y arrastre el cursor desde `SizeType` hasta `integer`. Con esto se crea una relación de restricción; véase también Crear relaciones entre elementos.



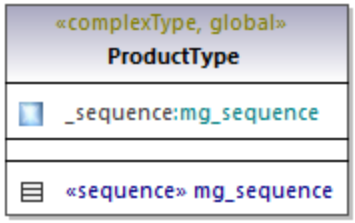
7. Para definir los valores `minInclusive` y `maxInclusive` seleccione el tipo simple y edite las propiedades con el mismo nombre en la ventana Propiedades.



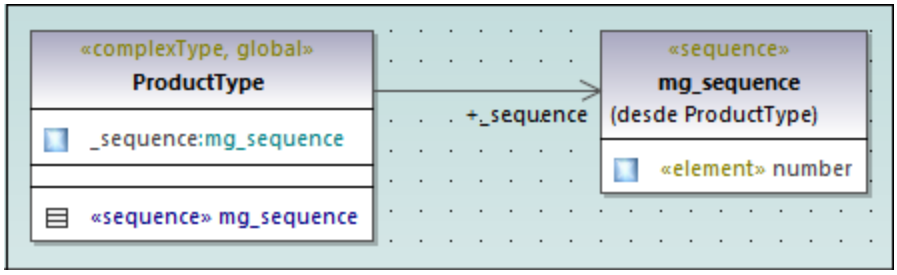
Agregar un tipo complejo

Siga estos pasos para añadir el tipo complejo `ProductType` al esquema XML. Todos estos pasos se reflejan también en **DiagramaPrincipal**.

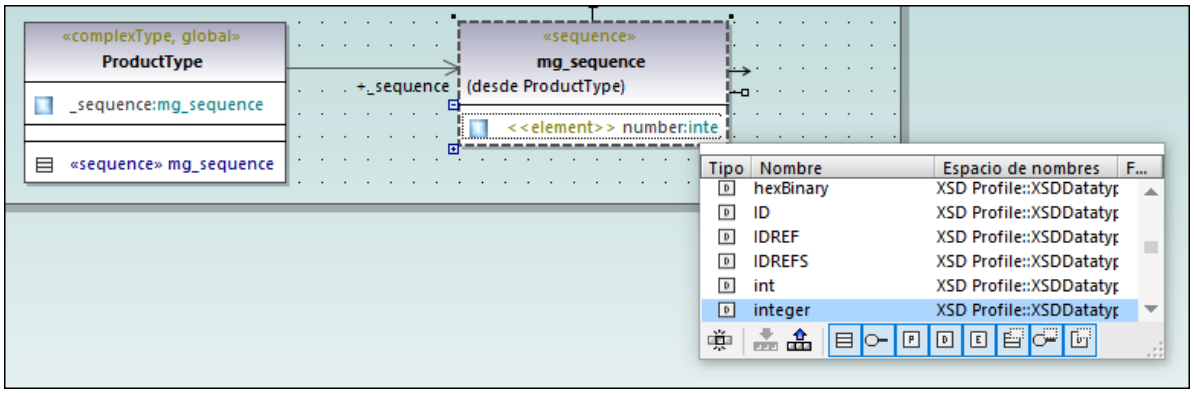
1. Haga clic en el botón **complexType XSD** de la barra de herramientas y después haga clic en el diagrama.
2. Cambie el nombre del tipo complejo a `ProductType`.
3. Haga clic con el botón derecho en el tipo complejo y seleccione **Nuevo | XSD sequence** en el menú contextual.



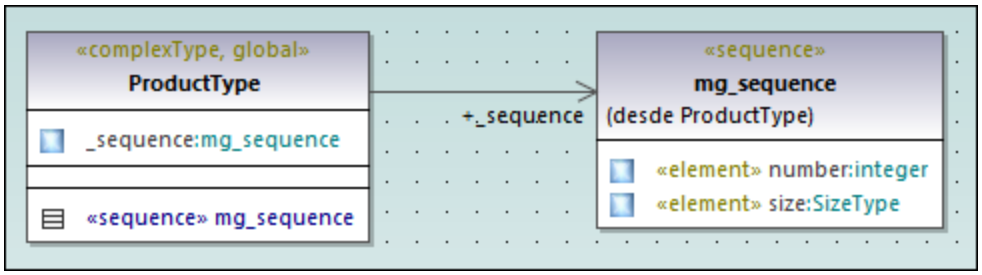
4. Arrastre la clase «sequence» desde el tipo complejo hasta el diagrama.



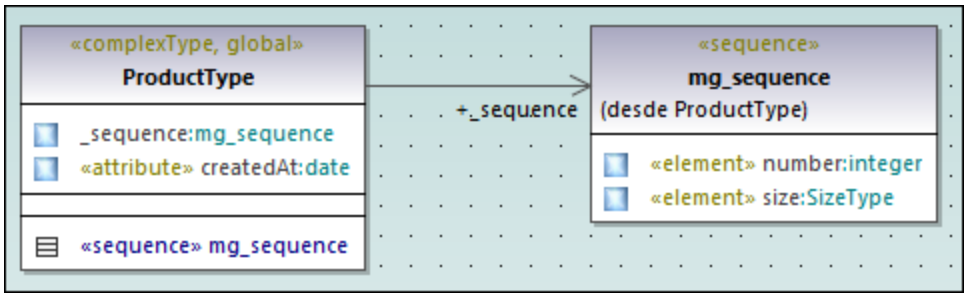
5. Haga clic con el botón derecho en la secuencia y seleccione **Nuevo | XSD Element (local)**. El tipo `integer` es un tipo base de esquema XML del perfil XSD. Para ver las instrucciones de cómo definir el tipo de un elemento consulte Finalización automática en clases.



6. Siga los mismos pasos que acabamos de explicar para crear un elemento **size** de tipo `SizeType`. Recuerde que `SizeType` es el tipo simple que creamos anteriormente.



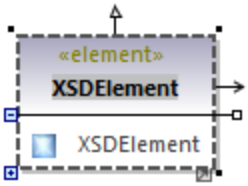
7. En el diagrama haga clic con el botón derecho en el tipo complejo y seleccione **Nuevo | XSD Attribute (local)** en el menú contextual.
8. Cambie el nombre del atributo a **createdAt** y el tipo a `date`.



Agregar un elemento

Ahora que ha definido todos los tipos obligatorios del esquema puede añadir un elemento de producto de tipo `ProductType`:

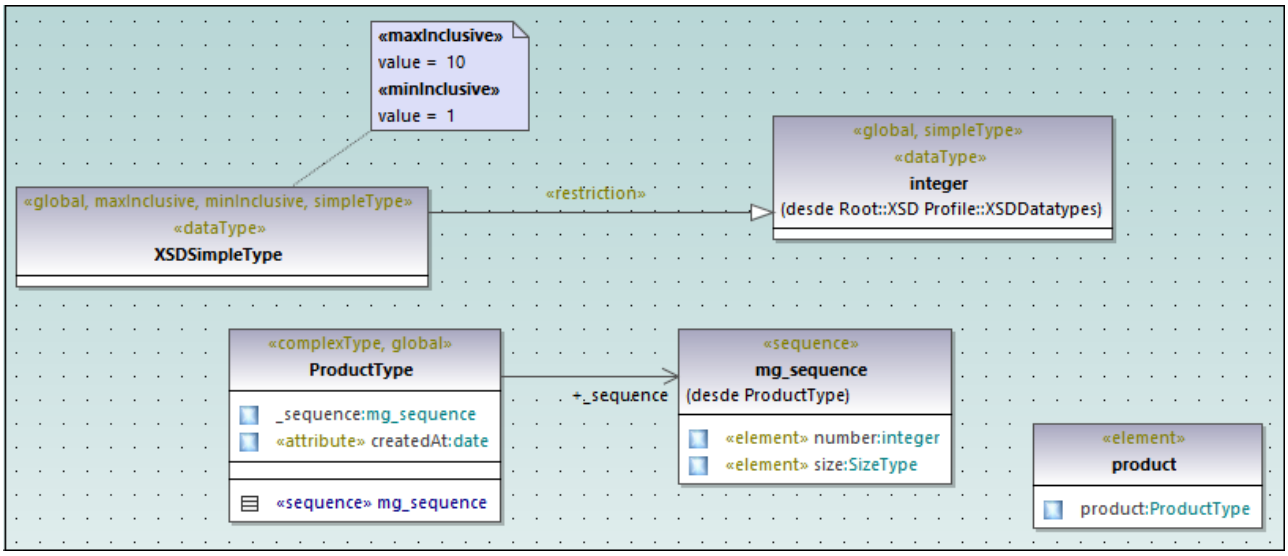
1. Haga clic en el botón **Element (global) XSD** de la barra de herramientas y después haga clic en el diagrama. Verá que se añaden una clase con el estereotipo y una propiedad única.



2. Cambie el nombre de la propiedad a **product** y su tipo a `ProductType`.

Diseño completo

Con los pasos del último punto concluye la parte de diseño del esquema. Ahora el esquema que ha diseñado debería tener este aspecto:



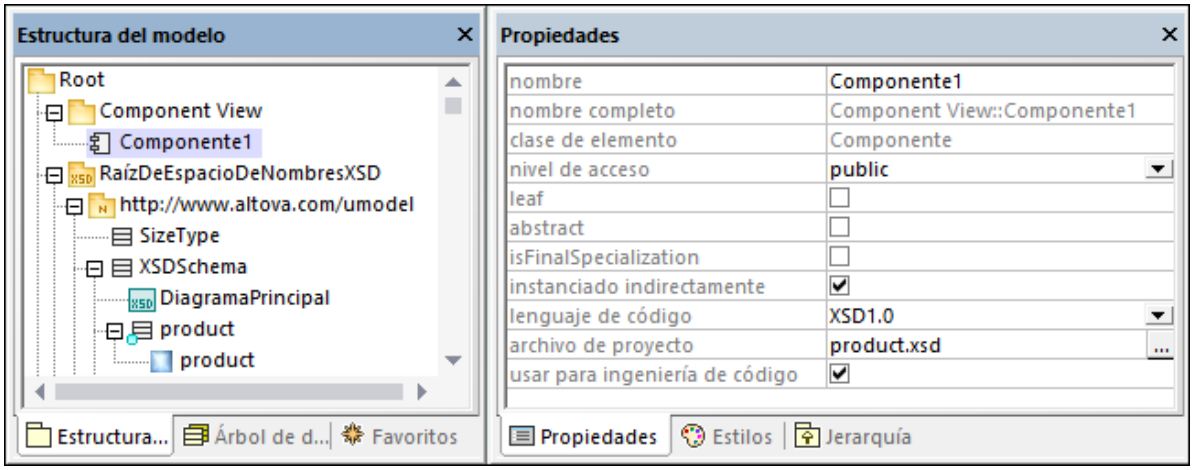
Habilitar la ingeniería de código

Con el fin de que se pueda generar un archivo de esquema a partir del modelo, vamos a añadir un componente de ingeniería de código que permite al esquema generar detalles. El componente de ingeniería de código es parecido a otros tipos de proyecto de UModel, véase también Agregar un componente de ingeniería de código.

Haga clic con el botón derecho en el paquete "Component View" en la Estructura del modelo y añada un elemento nuevo de tipo **Component**. Compruebe que cambia las propiedades del componente como explicamos a continuación:

1. Debe habilitar la propiedad `Utilizar para ingeniería de código`.

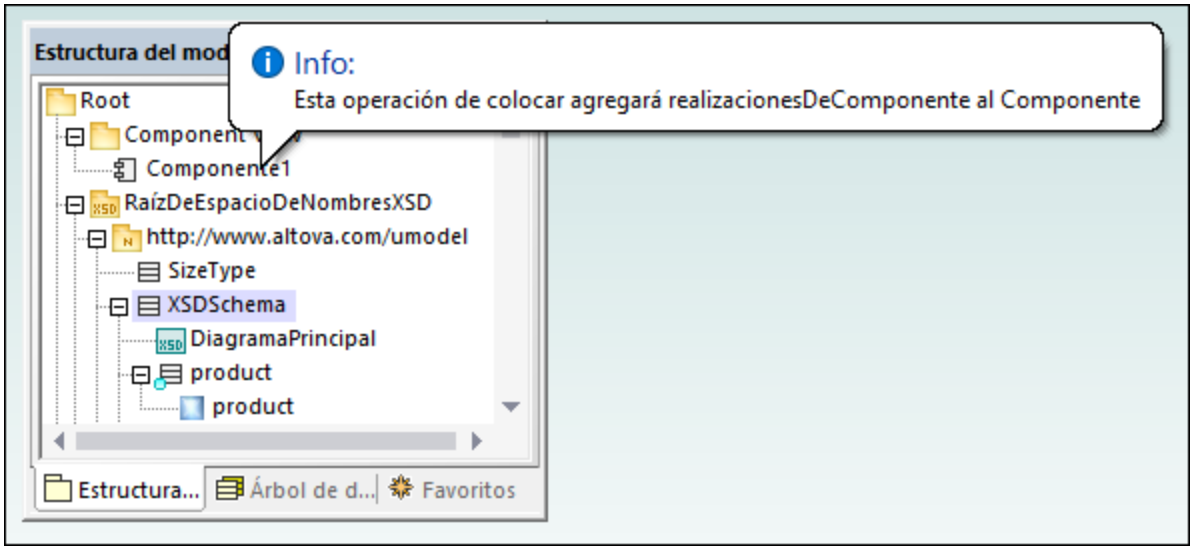
2. Defina la propiedad `lenguaje de código` del componente de ingeniería de código como "XSD 1.0".
3. La propiedad `archivo de proyecto` del componente de ingeniería debe apuntar al archivo de esquema que se quiere generar (en este ejemplo, **product.xsd**).



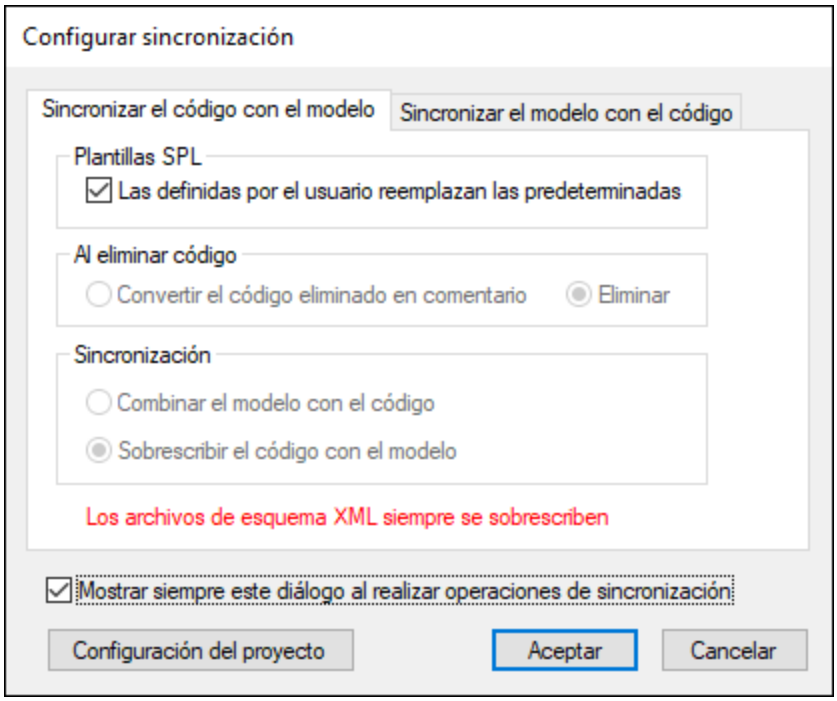
Nota: si falta una propiedad `archivo de proyecto`, introduzca **product.xsd** en la propiedad `directorio` y pulse **Entrar**. Ahora debería aparecer un cuadro de mensaje que le pide que haga referencia a un archivo de proyecto. Haga clic en **Sí** para confirmar.

Por último, el esquema XML debe realizarlo el componente de ingeniería de código, como se describe en Agregar un componente de ingeniería de código. En este ejemplo la forma más rápida de crear la relación **RealizaciónDeComponente** es:

- En la Estructura del modelo arrastre el esquema **XSDSchema** hasta el componente de ingeniería de código (**Component1**) y suéltelo cuando aparezca la información rápida:



Ahora puede generar el archivo del esquema. Para ello puede pulsar **F12** o seleccionar el comando de menú **Proyecto | Combinación/sobrescritura del código de programa con el proyecto de UModel**. Tenga en cuenta que la opción de combinar no es compatible con los esquemas XML; en estos casos en el cuadro de diálogo aparece un mensaje en rojo para informar de ello.



El esquema XML nuevo se genera en la misma carpeta que el proyecto de UModel.

9.3.2 Diagramas BPMN 1.0 / 2.0

Sitio web de Altova: Notación de modelado de procesos de negocio (BPMN)

BPMN es una notación estándar de diagramas de flujo que muestran procesos de negocio como flujos de trabajo de fácil comprensión para todos los participantes del proceso. UModel es compatible con la versión 1.0 y 2.0 de BPMN. UModel también es compatible con los diagramas BPMN 2.0 de coreografía, de colaboración y con los diagramas de procesos de negocio estándar BPMN 2.0. En un proyecto de UModel puede usar diagramas BPMN 1.0 y diagramas BPMN 2.0. Además puede convertir diagramas BPMN 1.0 en diagramas BPMN 2.0 en cualquier momento del proceso de modelado.

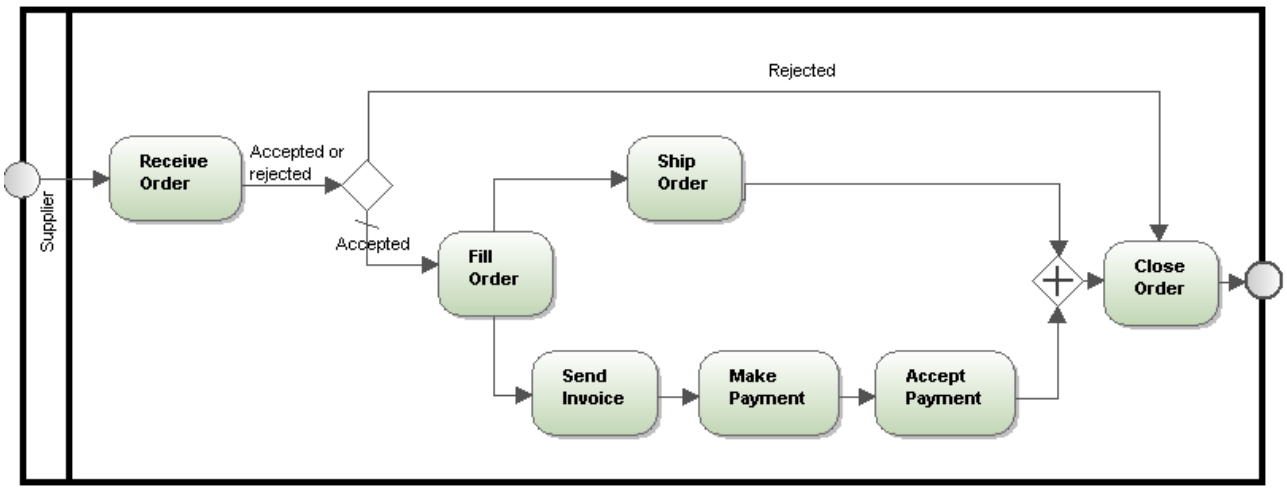
Hay cuatro categorías de elementos BPMN:

Objetos de flujo	Eventos, Actividades (Tareas o Subprocesos) y Compuertas
Objetos de conexión	Flujos de secuencia, Flujos de mensaje y Asociación
Compartimentos	Contenedores y Compartimentos
Artefactos	Objetos de datos, Grupos y Anotaciones textuales

En UModel los diagramas y objetos BPMN se insertan de la misma forma que los demás elementos de modelado.

Los objetos se pueden insertar con ayuda de los iconos de la barra de herramientas. Las asociaciones con otros objetos se pueden crear arrastrando los controladores de objeto hasta el objeto de destino. Al igual que en los demás diagramas, los elementos BPMN se pueden ver y configurar en la ventana *Propiedades*.

Recuerde que puede crear varias capas en cada diagrama BPMN. Para más información consulte el apartado Añadir capas a los diagramas.



Para convertir diagramas BPMN 1.0 en diagramas BPMN 2.0:

1. Haga clic con el botón derecho en un diagrama BPMN 1.0 y seleccione la opción **Convertir en diagrama BPMN 2**.
2. Si en el mismo paquete hay varios diagramas BPMN 1.0, aparece un aviso preguntando si desea convertir todos los diagramas del paquete.
3. Después aparece otro aviso preguntando si desea incluir el perfil BPMN 2 en el proyecto.
4. Haga clic en **Aceptar** para convertir los diagramas.

9.3.2.1 Objetos de flujo

Los objetos de flujo son elementos gráficos que definen el comportamiento de un proceso de negocios. Los hay de tres tipos: eventos, actividades y compuertas.

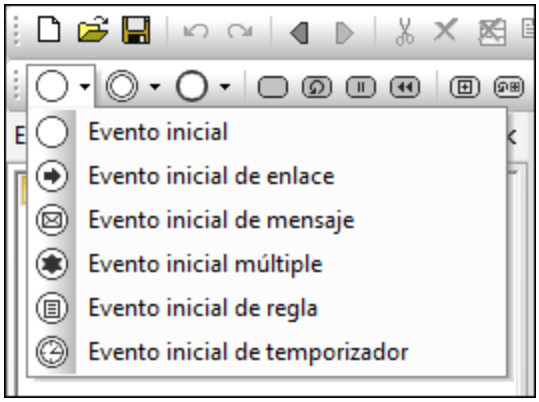
Eventos

Un evento es algo que ocurre durante un proceso de negocio y se representa con un círculo. Los eventos afectan al flujo del proceso y por lo general tienen una causa (desencadenador) y un resultado. Hay tres tipos de eventos: inicial, intermedio y final. Cada tipo de evento tiene un menú desplegable en la barra de herramientas.

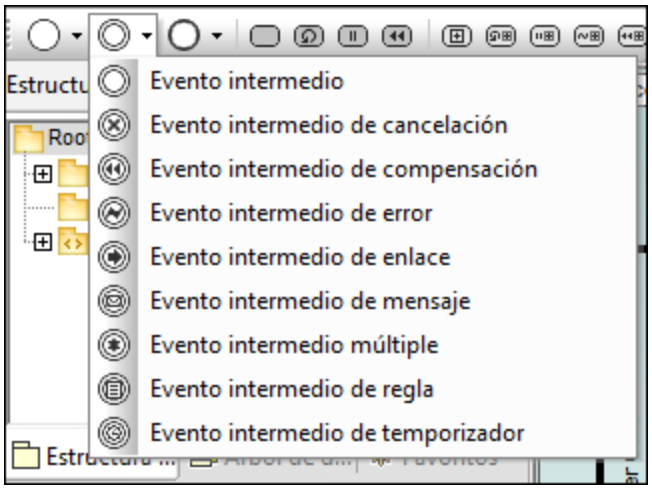
Para insertar un evento:

1. Haga clic en el menú desplegable para ver la lista de eventos del tipo que desea insertar.

2. Seleccione el evento que desea insertar y haga clic en el área de trabajo del diagrama para insertarlo.

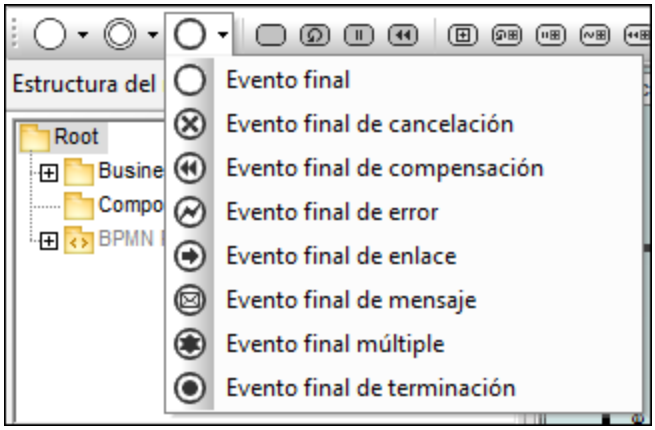


Eventos iniciales



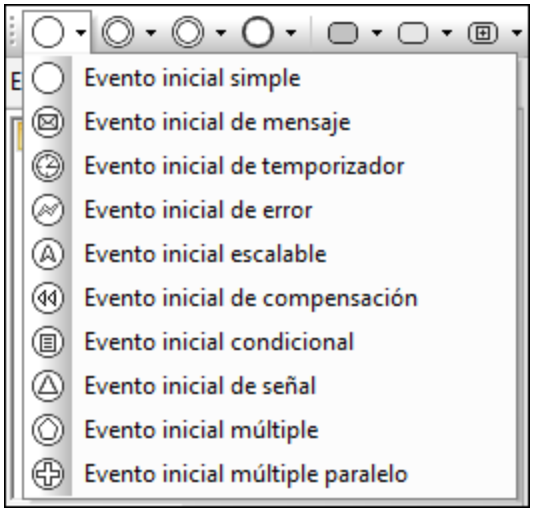
Eventos intermedios

Los eventos intermedios se pueden anexar al contorno de una tarea o subproceso e indican que la actividad debe interrumpirse cuando el evento se desencadena.

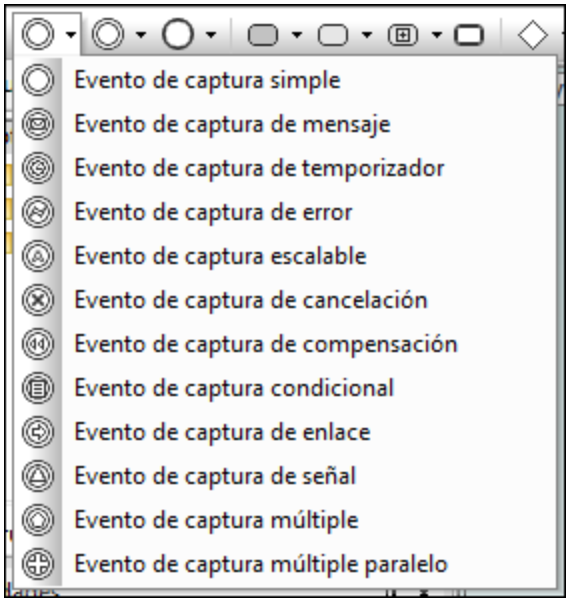


Eventos finales

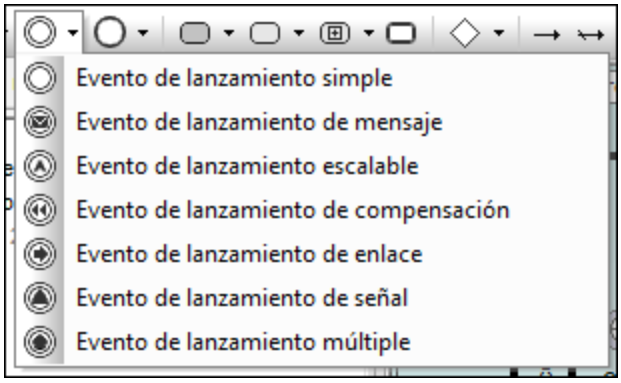
Eventos BPMN 2.0



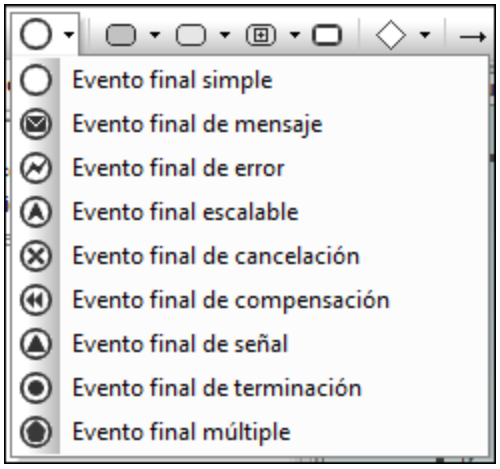
Eventos iniciales



Eventos de captura



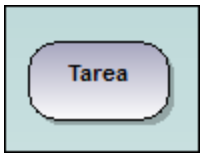
Eventos de lanzamiento



Eventos finales

Actividades

Las actividades son acciones que se desarrollan durante un proceso de negocio y se representan con rectángulos redondeados. Las hay de tres tipos: procesos, subprocesos y tareas. Las actividades pueden ocurrir una sola vez o varias veces en un ciclo.



Para insertar una actividad:

1. Haga clic en el icono de la tarea o subproceso en la barra de herramientas.
2. Haga clic en el área de trabajo del diagrama para insertar la actividad.

Tareas

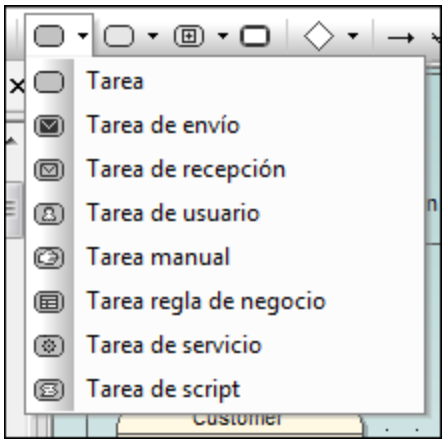
Las tareas son actividades que están incluidas en un proceso y no se pueden desglosar en subtareas porque son atómicas.

Tarea **(Ciclo)**

Tarea **(instancia múltiple)**

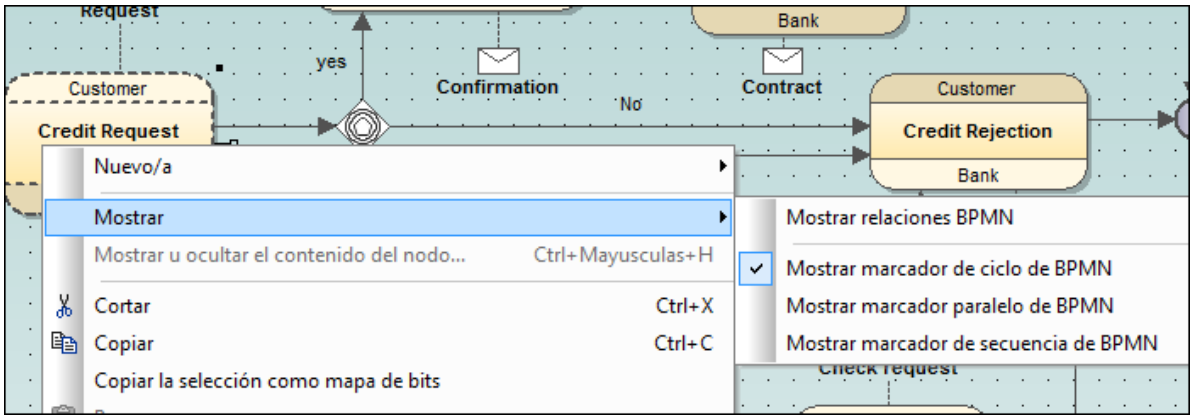
Tarea
(compensación)

Tareas BPMN 2.0



Para definir un marcador de ciclo, paralelo, de secuencia o de compensación:

- Haga clic con el botón derecho en la tarea insertada y seleccione un tipo de marcador (p. ej. **Mostrar | Mostrar marcador de ciclo de BPMN**).



Nota: también puede definir el marcador en la ventana *Propiedades* (en la entrada MultiInstanceLoopCharacteristics).

Subprocesos

Un subproceso es una actividad compuesta incluida en un proceso, que permite desarrollar el modelo del proceso de negocio de forma jerárquica. Un subproceso puede desglosarse en varias actividades subordinadas.

Un subproceso **contraído** se representa como elemento de nivel superior donde los datos del subproceso no son visibles. En el elemento aparece también el icono +, lo cual indica que existe una capa más de complejidad.

Un subproceso **expandido** muestra los datos del subproceso dentro de sus límites. Recuerde que un flujo de secuencia no puede cruzar el límite de un subproceso.

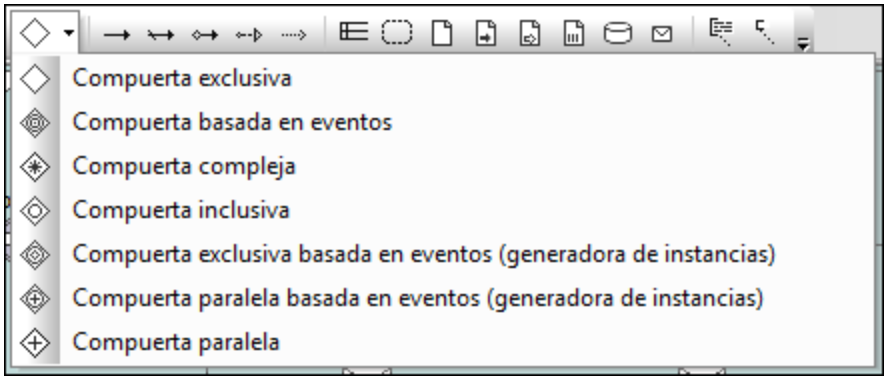
Compuertas

Las compuertas sirven para determinar cómo se ramifican y cómo convergen los flujos de secuencia dentro de un proceso. Las compuertas se representan con un icono en forma de rombo (*tabla siguiente*).

Compuerta **inclusiva** (OR)	
Compuerta **paralela** (AND)	
Compuerta **exclusiva basada en datos** (XOR)	
Compuerta **exclusiva basada en eventos** (XOR)	
Compuerta **compleja** (bifurcación/convergencia)	

Compuertas BPMN 2.0

En la imagen siguiente se ven las compuertas BPMN 2.0 compatibles. En UModel puede visualizar las compuertas exclusivas con o sin X. Para que el icono contenga la X debe cambiar el valor `showXIcon` de la compuerta exclusiva correspondiente a `true`.



9.3.2.1.1 Subprocesos expandidos

Los subprocesos expandidos muestran los detalles del proceso dentro de los límites del elemento.

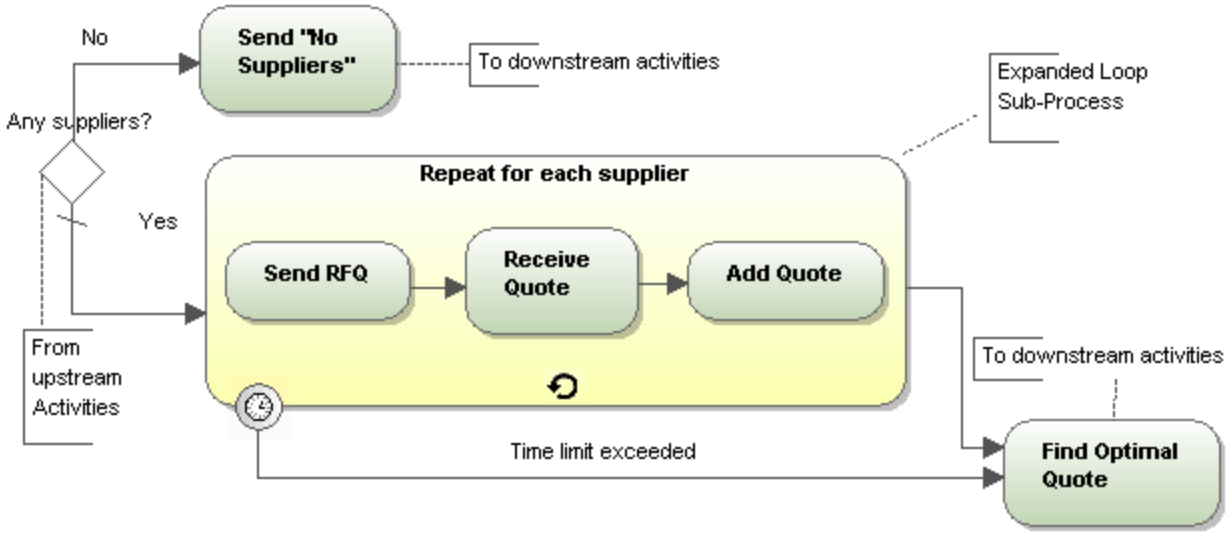
Subproceso expandido

Subproceso de **ciclo** expandido

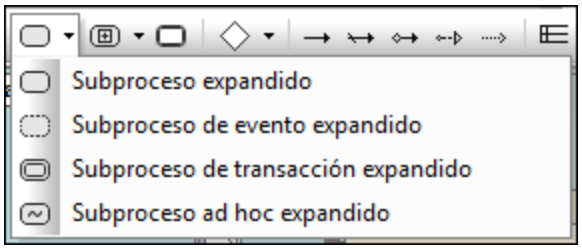
Subproceso de **instancia múltiple** expandido

Subproceso **ad hoc** expandido

Subproceso de **compensación** expandido

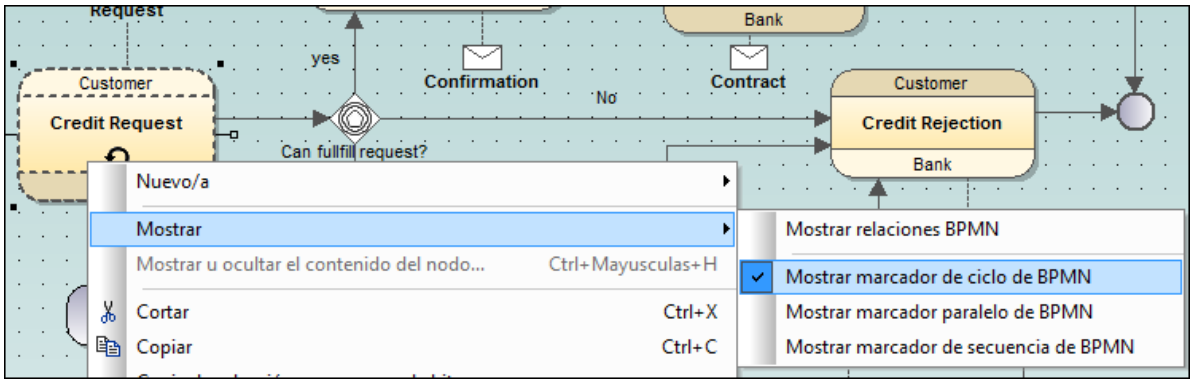


Subprocesos expandidos BPMN 2.0



Para definir un marcador de ciclo, paralelo, de secuencia o de compensación:

1. Haga clic con el botón derecho en la tarea insertada y seleccione un tipo de marcador (p. ej. **Mostrar | Mostrar marcador de ciclo de BPMN**).



9.3.2.1.2 Subprocesos contraídos

Los subprocesos contraídos ocultan los detalles del proceso. Cada tipo de subproceso contraído se representa con un icono diferente dentro del elemento `Subproceso`.

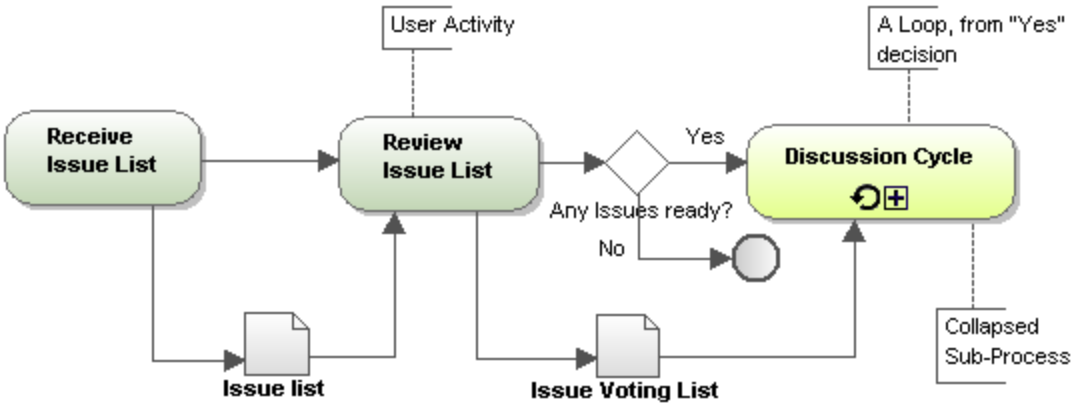
Subproceso contraído

Subproceso de **ciclo** contraído

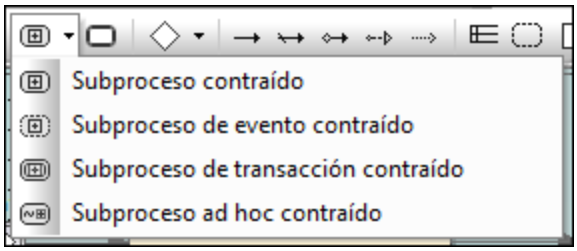
Subproceso de **instancia múltiple** contraído

Subproceso **ad hoc** contraído

Subproceso de **compensación** contraído

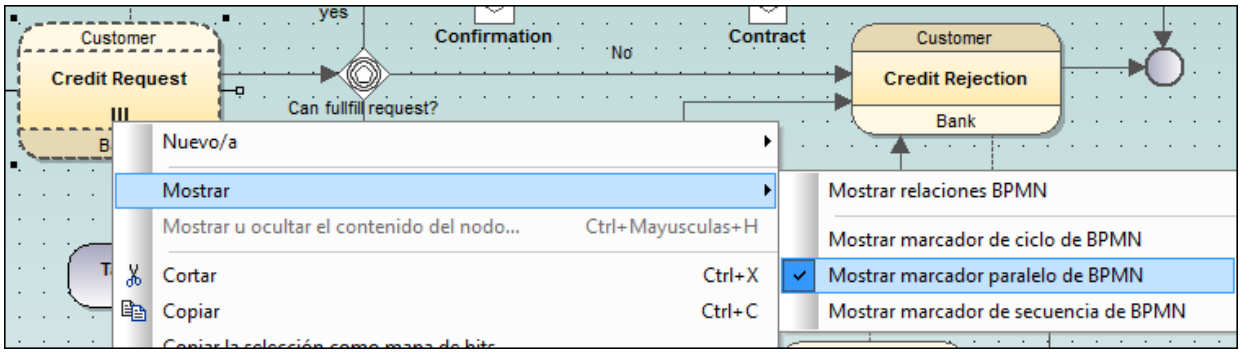


Subprocesos contraídos BPMN 2.0:



Para definir un marcador de ciclo, paralelo, de secuencia o de compensación:

- Haga clic con el botón derecho en la tarea insertada y seleccione un tipo de marcador (p. ej. **Mostrar | Mostrar marcador paralelo de BPMN**).

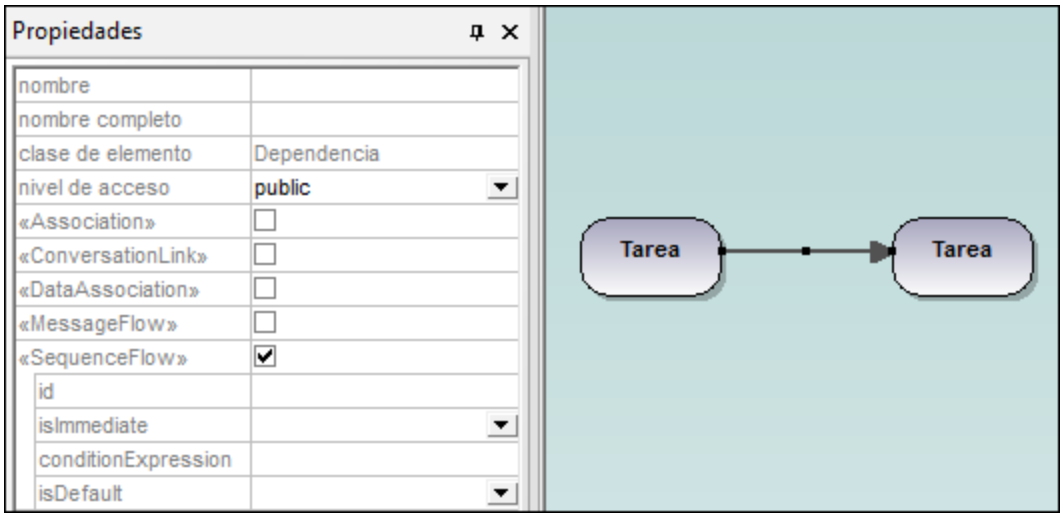


9.3.2.2 Objetos de conexión

Hay dos maneras de conectar objetos en BPMN: con flujos (usando una secuencia o un mensaje) y con asociaciones.

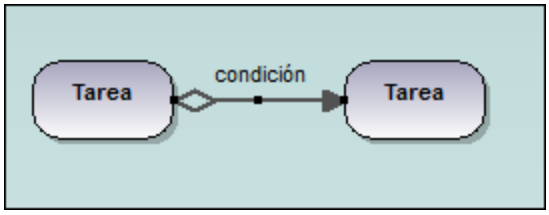
Flujo de secuencia

Un flujo de secuencia muestra en qué orden se llevan a cabo las actividades dentro de un proceso.



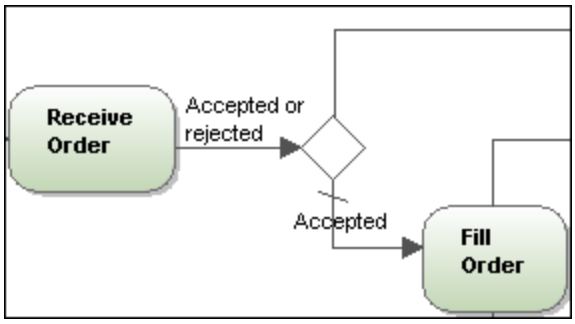
Flujo condicional

Este tipo de flujo puede tener una expresión condicional que se evalúa para determinar si el flujo se utiliza o no. Si el flujo condicional tiene su origen en una actividad, al principio de la flecha aparece un pequeño **rombo**.



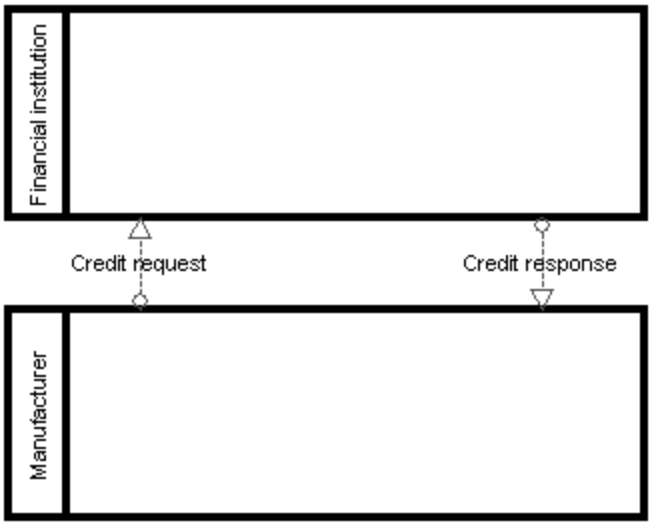
Flujo por defecto

Este tipo de flujo se utiliza si todos los demás flujos condicionales dan false como resultado en decisiones inclusivas o exclusivas. El flujo por defecto se señala con una **barra diagonal** al principio de la flecha del flujo.



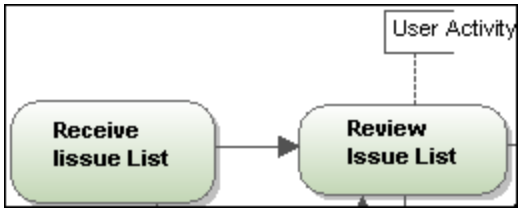
Flujo de mensaje

Un flujo de mensaje muestra el flujo de mensajes que tiene lugar entre dos participantes (entidades o roles), que pueden enviarlos y recibirlos. Cada participante se representa en un contenedor.



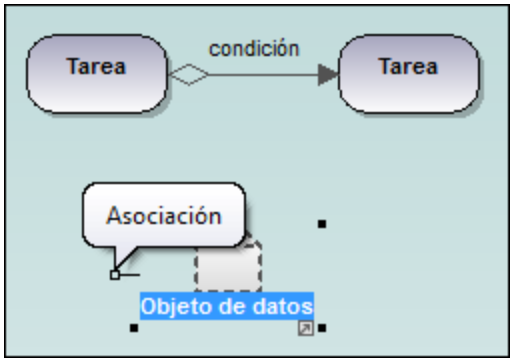
Asociaciones

Las asociaciones sirven para asociar texto o datos de objetos con objetos de flujo e indican cómo entran y salen datos de las actividades. El diagrama de la imagen siguiente, por ejemplo, muestra una anotación textual que ofrece información adicional (**User Activity**) para la tarea **Review Issue List**.



Para crear una asociación entre un objeto de datos y un control de flujo:

1. Haga clic en el controlador Asociación del objeto de datos (en el borde izquierdo del objeto).
2. Arrastre el conector hasta la flecha de control de flujo (que se resalta para indicar que puede soltar el conector).



Otra opción es hacer clic en el icono **Asociación** de la barra de herramientas, hacer clic en el objeto de datos y arrastrar el puntero del ratón hasta el control de flujo.

9.3.2.3 Contenedores y compartimentos

Contenedores

Los contenedores sirven para dividir y organizar las actividades. Un proceso de negocio puede mostrar la interacción que existe entre varios procesos o participantes. Cada participante se representa en un rectángulo que recibe el nombre de *contenedor.* Un participante puede ser un rol o una entidad.

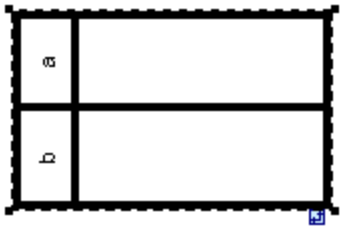
- Puede arrastrar objetos BPMN hasta un contenedor y colocarlos dentro cuando el contorno del contenedor se resalte.
- Puede seleccionar los objetos que están dentro del contenedor usando las teclas **Ctrl+clic** o arrastrando el puntero para seleccionar los objetos.
- Para mover el contenedor a otra posición, haga clic en su contorno o título y arrástrelo hasta la posición nueva.

Compartimento

Los contenedores pueden dividirse en compartimentos, que sirven para organizar las actividades del contenedor por categorías. Los compartimentos pueden ser horizontales o verticales.

Para añadir un compartimento en un contenedor:

1. Haga clic con el botón derecho en el título del contenedor y elija **Nuevo/a | Compartimento**. Esto añade un compartimento nuevo en el contenedor. Cada compartimento puede tener un nombre distinto (haga doble clic en el campo nombre y escriba el nombre del compartimento).



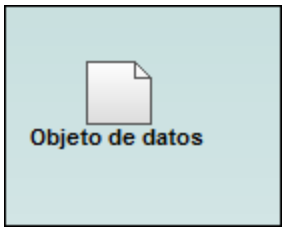
Nota: haga clic con el botón derecho en uno de los compartimentos y elija **Nuevo/a | <elemento>** para añadir elementos al contenedor.

9.3.2.4 Artefactos

Los artefactos sirven para mostrar información sobre un proceso, es decir, cómo se utilizan y actualizan los datos, los documentos y los demás objetos durante el proceso de negocio. Los artefactos no están relacionados directamente con el flujo de secuencia o de mensaje del proceso.

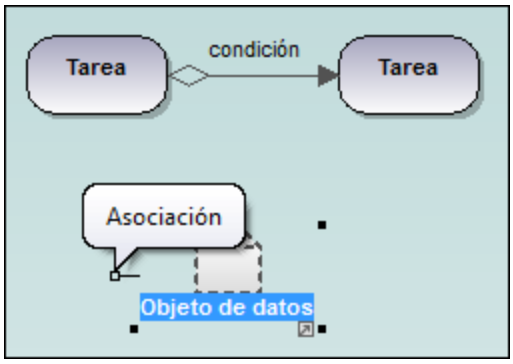
Objetos de datos

Los objetos de datos representan documentos o datos y su utilización durante el proceso de negocio. Puede usar los objetos de datos para definir cómo entran y salen datos de las actividades.



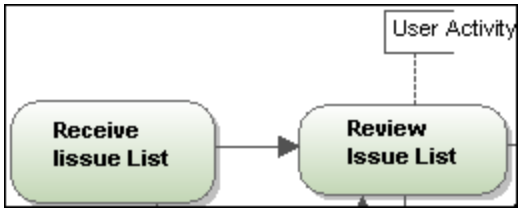
Para crear una asociación entre un objeto de datos y un control de flujo:

1. Haga clic en el controlador Asociación del objeto de datos (situado en el borde izquierdo del objeto).
2. Arrastre el conector hasta el control de flujo (que se resalta para indicar que puede soltar el conector).



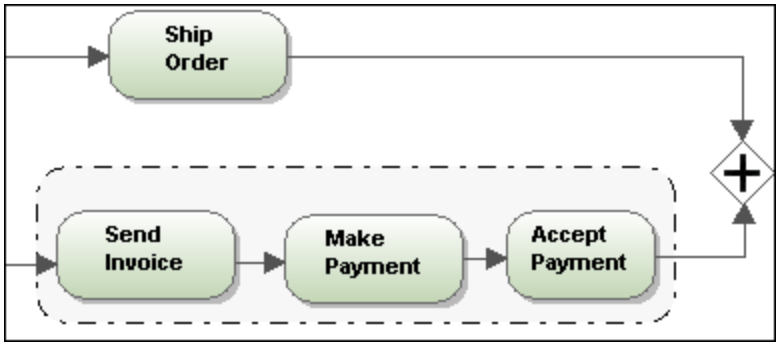
Anotaciones textuales

Las anotaciones textuales sirven para anotar secciones del proceso de negocio y se conectan al objeto por medio de asociaciones.



Grupos

Los grupos se utilizan para resaltar ciertas secciones de un diagrama, incluso de contenedores distintos. Los grupos no pueden conectarse a los flujos de secuencia ni de mensaje. Suelen colocarse detrás de tareas o procesos.

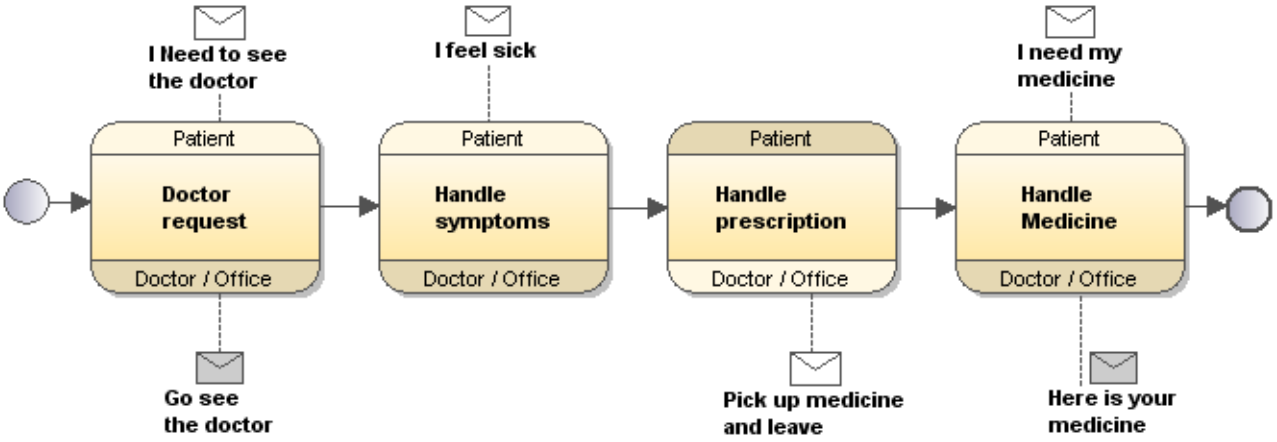


9.3.2.5 Diagrama de coreografía

Los diagramas de coreografía especifican cómo **coordinan** los participantes sus interacciones. También pueden entenderse como contratos entre los participantes, donde lo importante es el intercambio de la información (Mensajes).

Los contratos suelen tomar la forma de pedidos de compra que se envían a un proveedor, la confirmación del proveedor de que procesará el pedido y la ejecución del pedido. En las coreografías las actividades también se ordenan por medio de flujos de secuencia.

Las actividades están compuestas por interacciones que tienen lugar entre participantes. Las interacciones suelen denominarse *pauta de intercambio de mensajes* (o MEP por sus siglas en inglés). Una pauta de intercambio de mensajes es la *actividad* de la coreografía y también recibe el nombre de **tarea de coreografía**.



9.3.2.5.1 Tareas de coreografía

Hay cuatro tipos de tareas de coreografía:


Tarea de coreografía

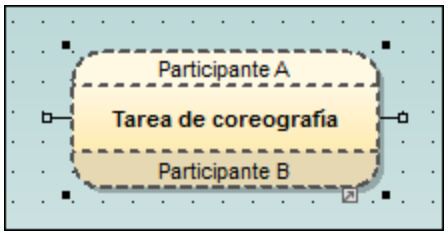

Sub-coreografía (contraída)


Sub-coreografía (expandida)


Coreografía de llamada

Para insertar una tarea de coreografía:

1. En la barra de herramientas haga clic en el icono del tipo de tarea que desea insertar (p. ej. **Tarea de coreografía**) y después haga clic en el área de trabajo del diagrama.

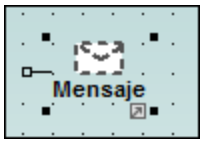


Por ejemplo, en la imagen anterior puede ver la vista predeterminada de una tarea de coreografía. En ella se resalta automáticamente el texto predeterminado **`Tarea de coreografía`**.

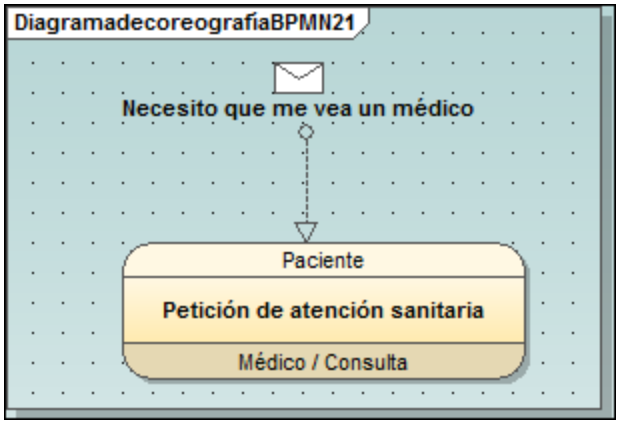
2. Escriba el nombre nuevo de la tarea de coreografía.
3. Haga clic en la banda superior para escribir el nombre del **`Participante A`** y en la banda inferior para escribir el nombre del **`Participante B`**.
   Las bandas de los participantes pueden estar sombreadas y no sombreadas. El iniciador de la actividad es el participante no sombreado (en este caso el Participante A).

Para añadir/asociar mensajes a una tarea de coreografía:

1. Haga clic en el icono **Mensaje** de la barra de herramientas y después en el área de trabajo del diagrama.

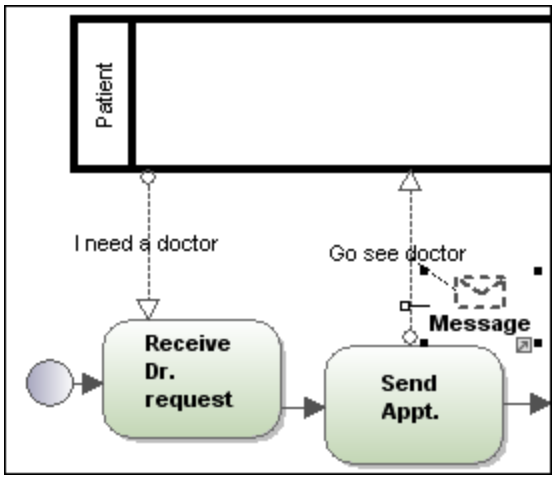


2. Escriba el nombre del mensaje (p. ej. Necesito que me vea un médico).
3. Haga clic en el controlador **`Asociación`** (del borde izquierdo) y arrástrelo hasta la tarea de coreografía.



Para añadir un mensaje a una línea (p. ej. de asociación):

1. Haga clic en la **línea** a la que dese añadir el mensaje.
2. Haga clic en el icono **Mensaje** de la barra de herramientas.
3. Haga clic otra vez en la línea.

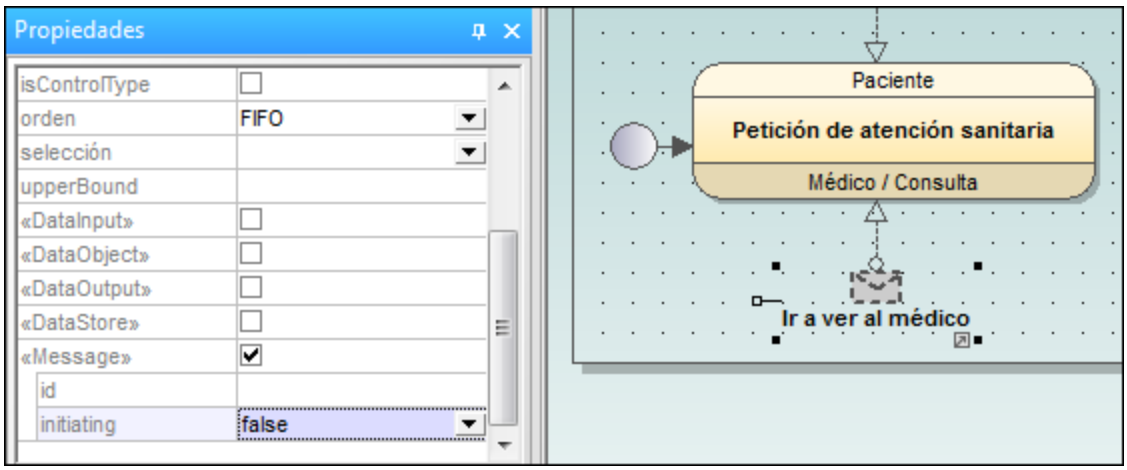


El mensaje se coloca encima de la línea y se anexa a la línea automáticamente.

Para cambiar el mensaje / participante de iniciación:

Cuando se inserta un `Mensaje`, el mensaje se define automáticamente como mensaje de iniciación (es decir, no está sombreado).

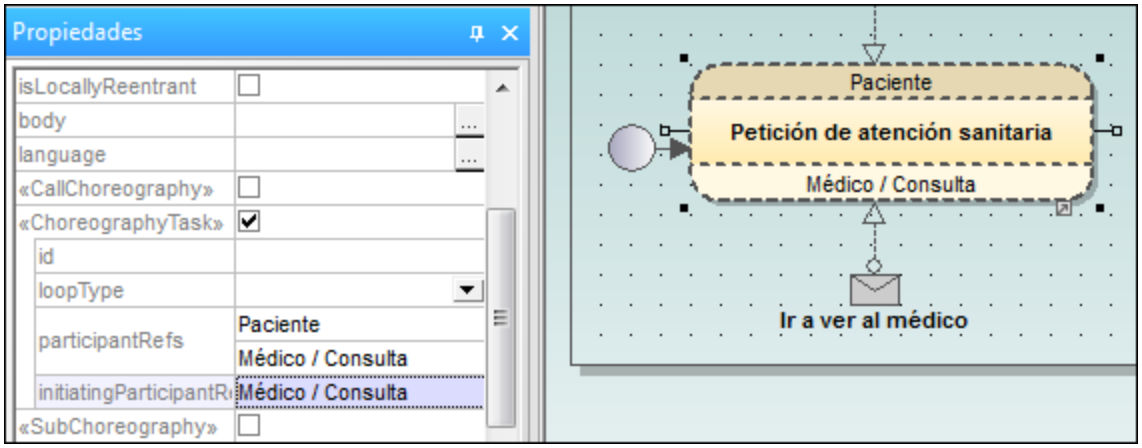
1. Haga clic en el mensaje y seleccione el valor false en el cuadro combinado initiating de la ventana *Propiedades*.



El mensaje aparece ahora sombreado.

Cuando se inserta una tarea de coreografía, el **`Participante A`** se define automáticamente como participante iniciador.

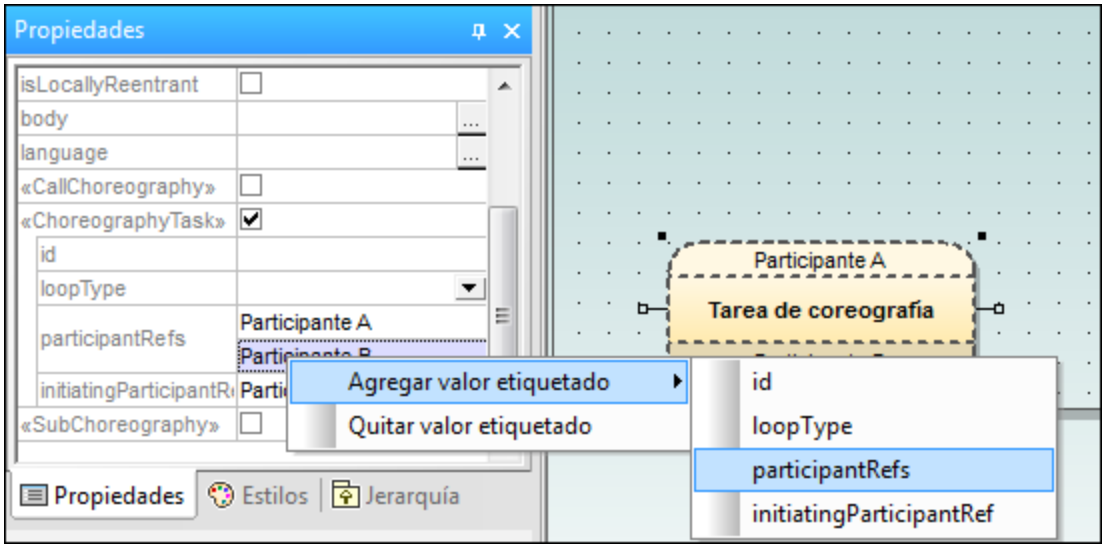
1. Haga clic en la tarea de coreografía que contiene el participante que debe funcionar como iniciador.
2. Escriba el nombre del participante que debe ser el iniciador en el cuadro combinado InitiatingParticipantRef de la ventana *Propiedades* (p. ej. Médico / Consulta).



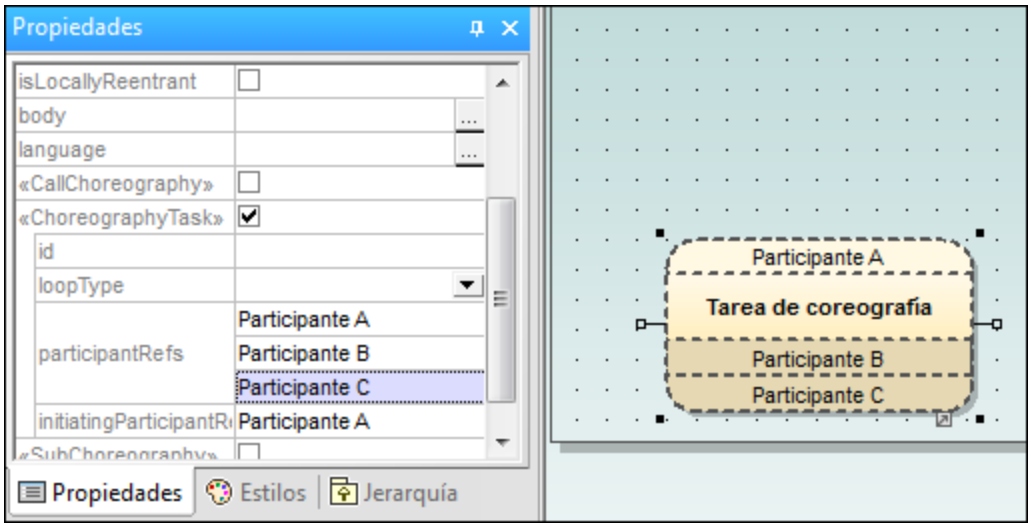
La banda del participante **`Médico / Consulta`** ya no está sombreada, lo cual indica que se trata del participante iniciador. La banda del participante **`Paciente`** está sombreada.

Para añadir participantes nuevos a una tarea de coreografía:

1. Haga clic en la tarea en la que desea añadir el participante.

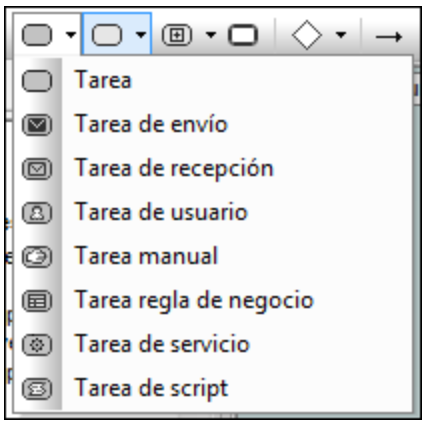


2. En la ventana *Propiedades* haga clic con el botón derecho en el campo participantsRefs y seleccione **Agregar valor etiquetado | participantRefs**.
3. Escriba el nombre del nuevo participante (p. ej. **`Participante C`**).

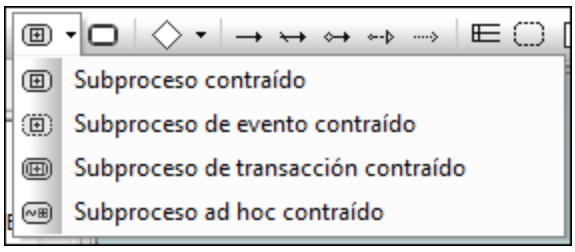


9.3.2.5.2 Tareas y subprocesos

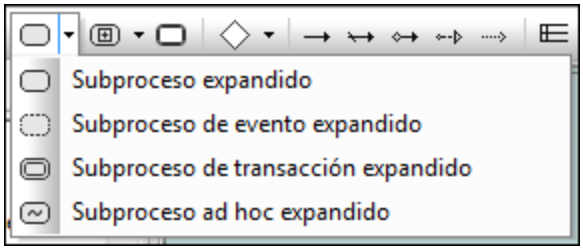
Haga clic en el icono desplegable **Tarea** para insertar una tarea determinada.



Haga clic en el icono desplegable **Subproceso contraído** para insertar un subproceso contraído determinado.



Haga clic en el icono desplegable **Subproceso expandido** para insertar un subproceso expandido determinado.



9.3.2.5.3 Objetos de datos

Los datos se representan por medio de cinco elementos de modelado y se insertan con ayuda de estos iconos:

Objeto de datos

Representa la información que fluye a través de los procesos (p. ej. correos electrónicos, documentos comerciales, etc.). Los objetos de datos ofrecen información sobre qué actividades deben llevarse a cabo y qué producen.

Datos de salida

Representa el resultado del proceso.

Datos de entrada

Es una entrada externa para todo el proceso. Pueden ser leídos por una actividad.

Colección de datos

Representa una colección de información (p. ej. una lista de pedidos).

Almacén de datos

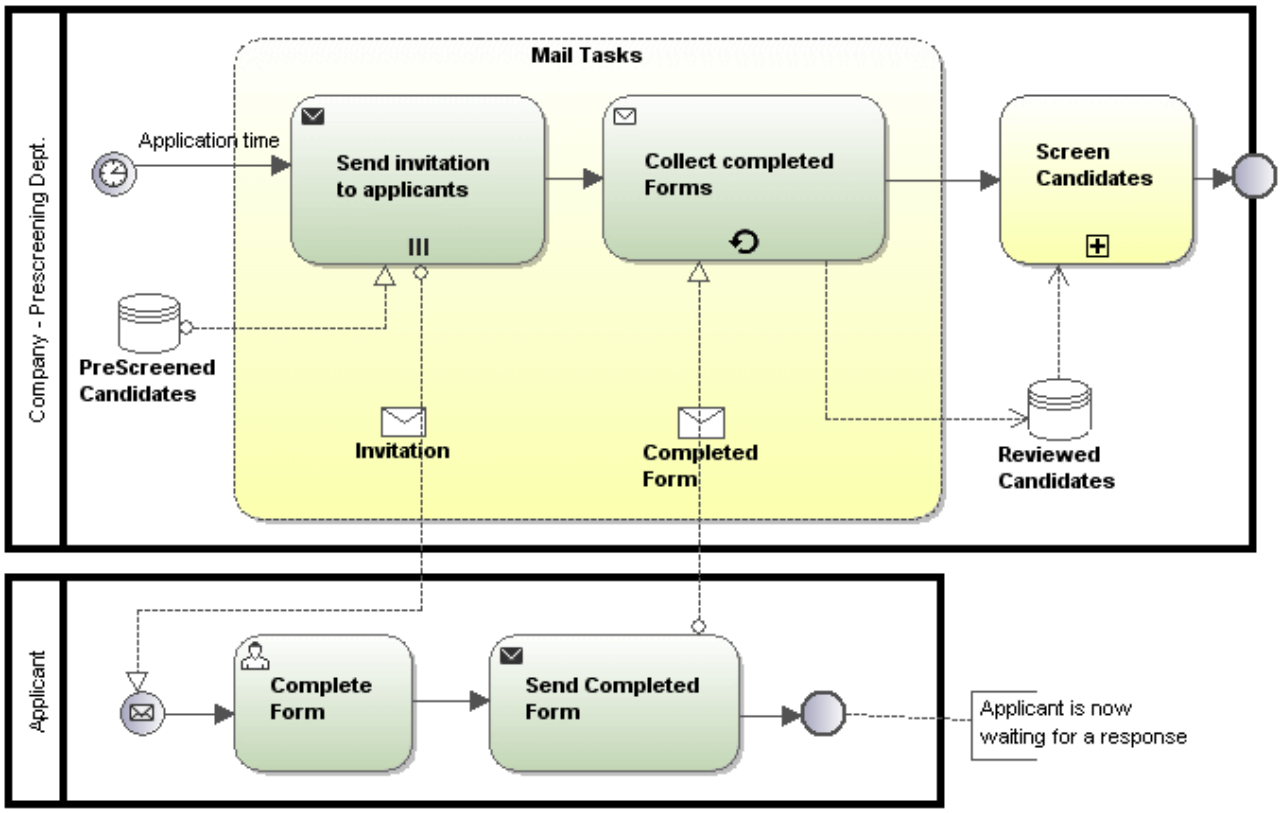
Una ubicación donde el proceso puede leer o escribir datos (p. ej. una base de datos).

9.3.2.6 Diagrama de colaboración

Los diagramas de colaboración muestran qué interacciones existen entre los procesos.

Una colaboración suele estar compuesta por dos contenedores o más, que representan los participantes que intervienen en la colaboración. Los intercambios de mensajes que se producen entre los participantes se representan por medio de flujos de mensaje, que conectan los dos contenedores o los objetos que están dentro de los contenedores. Los contenedores también pueden estar vacíos.

En un diagrama de colaboración puede combinar contenedores procesos y coreografías.



9.3.2.6.1 Conversación

Una conversación es una versión resumida de una colaboración y tiene acceso a los mismos elementos de modelado. Una conversación define un conjunto de intercambios de mensajes relacionados lógicamente: los intercambios de mensajes se relacionan entre sí para reflejar un caso único, como por ejemplo una solicitud seguida de una respuesta.

Las conversaciones tienen dos elementos gráficos que no están disponibles en los demás diagramas BPMN:

- nodos de conversación (conversación, sub-conversación y comunicación) y
- conectores de conversación.

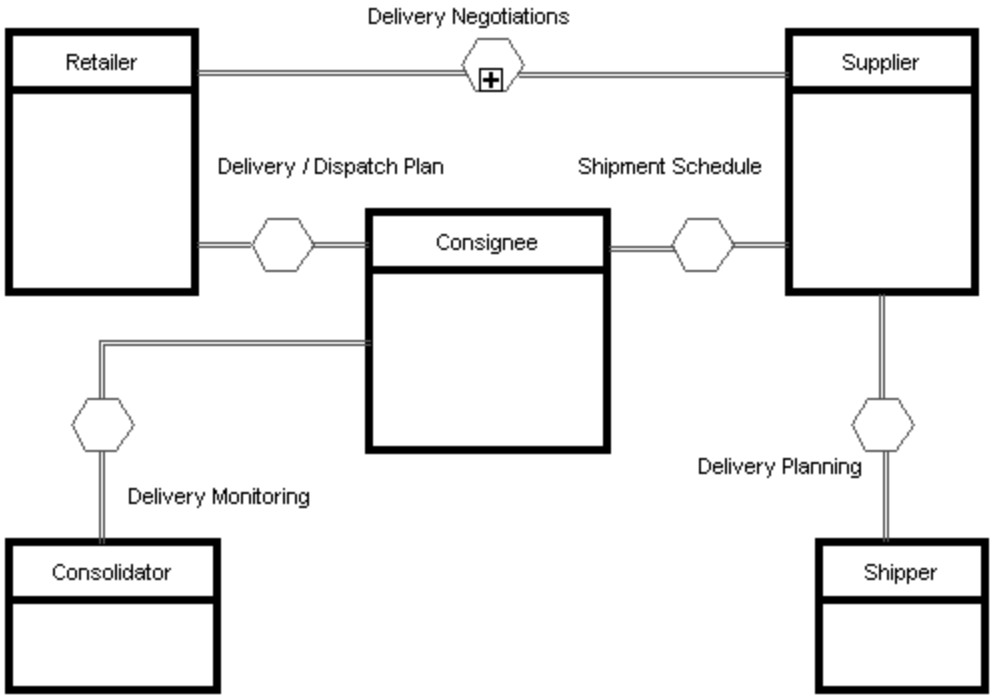

Conversación


Sub-conversación


Comunicación

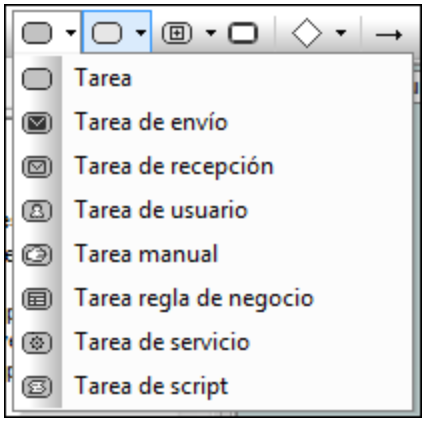

Participante / Contenedor


Conector de conversación

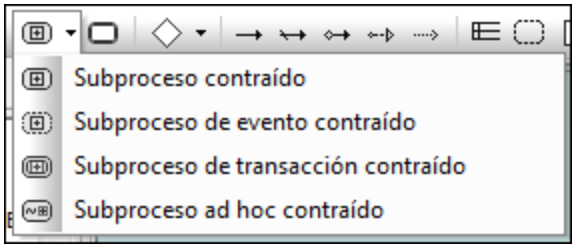


9.3.2.6.2 Tareas y subprocesos

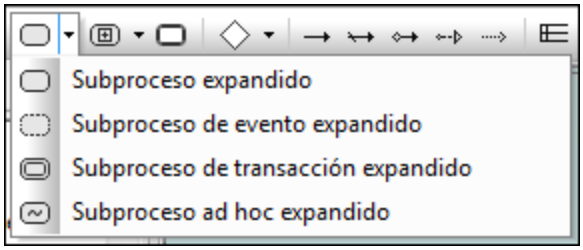
Haga clic en el icono desplegable **Tarea** para insertar una tarea determinada.



Haga clic en el icono desplegable **Subproceso contraído** para insertar un subproceso contraído determinado.



Haga clic en el icono desplegable **Subproceso expandido** para insertar un subproceso expandido determinado.



9.3.2.6.3 Objetos de datos

Los datos se representan por medio de cinco elementos de modelado y se insertan con ayuda de estos iconos:

Objeto de datos

Representa la información que fluye a través de los procesos (p. ej. correos electrónicos, documentos comerciales, etc.). Los objetos de datos ofrecen información sobre qué actividades deben llevarse a cabo y qué producen.

Datos de salida

Representa el resultado del proceso.

Datos de entrada

Es una entrada externa para todo el proceso. Pueden ser leídos por una actividad.

Colección de datos

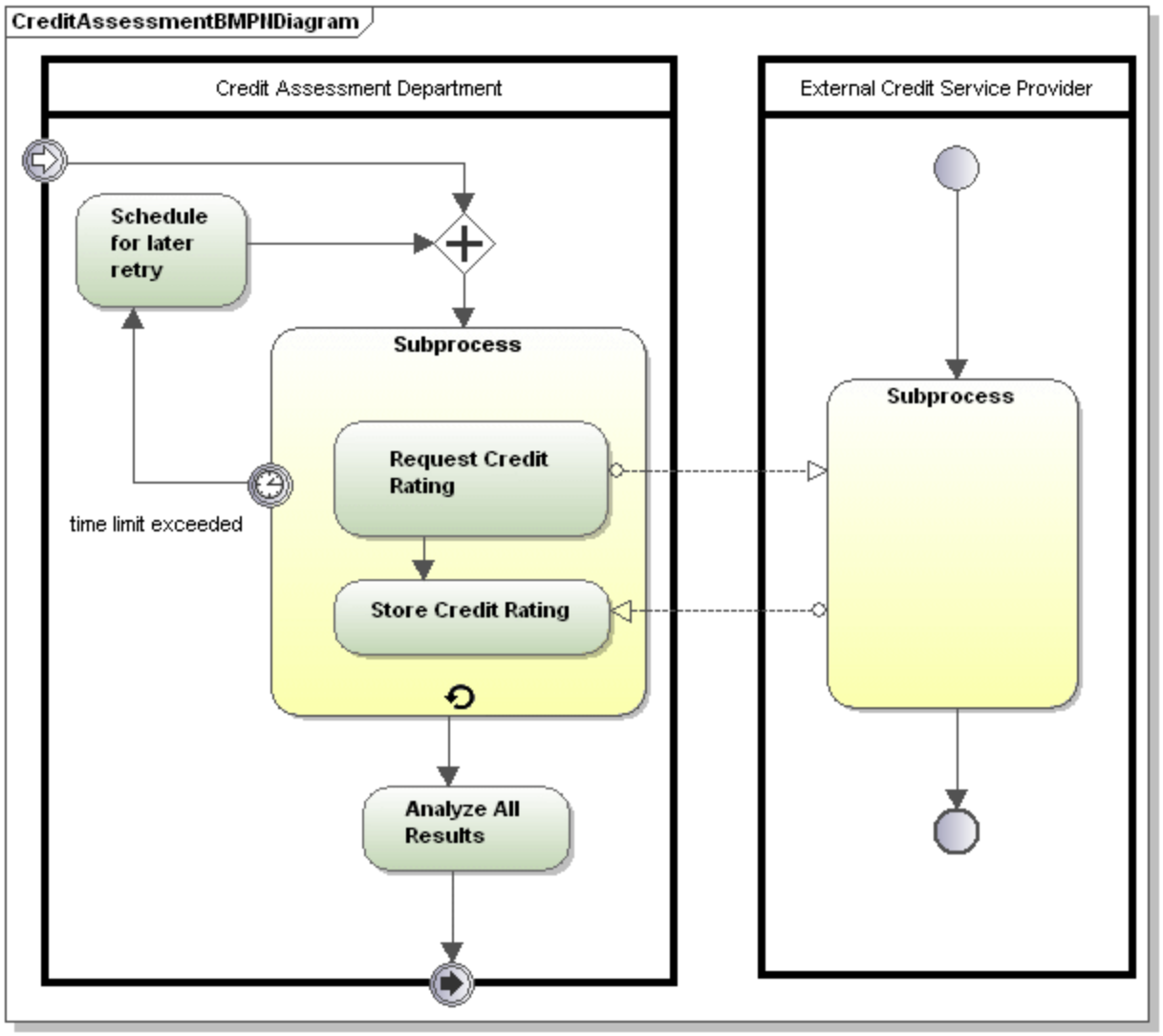
Representa una colección de información (p. ej. una lista de pedidos).

Almacén de datos

Una ubicación donde el proceso puede leer o escribir datos (p. ej. una base de datos).

9.3.2.7 Diagrama de procesos de negocio estándar BPMN 2.0

Los diagramas de procesos de negocio abarcan una amplia gama de datos y emplean varios tipos de modelado diferentes para crear procesos de negocio.



Hay tres tipos de procesos de negocio:

Procesos de negocio **privados** (internos) **no ejecutables**:

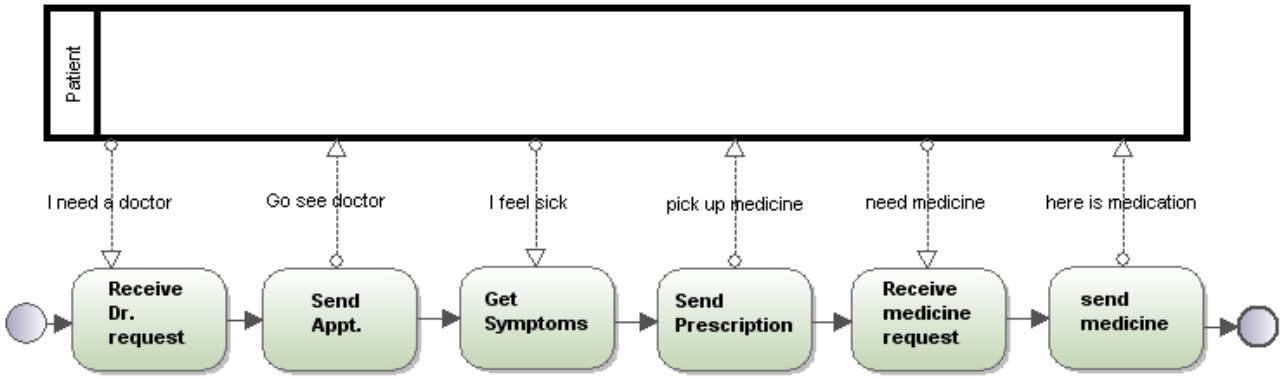
- Los procesos no ejecutables son aquellos en los que no hay datos suficientes para poder ejecutar el proceso (por lo general durante el ciclo de desarrollo).

Procesos de negocio **privados** (internos) **ejecutables** (internos):

- Los procesos ejecutables son procesos que se pueden ejecutar porque se han modelado completamente siguiendo las reglas semánticas de BPMN 2.0.

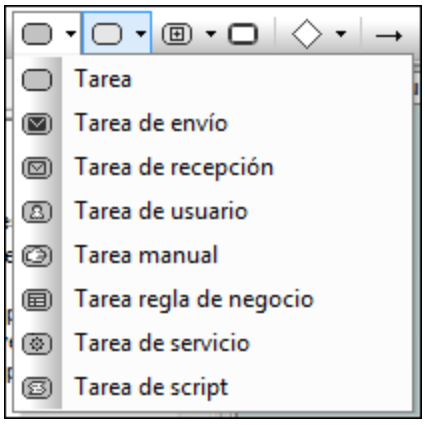
Procesos **públicos**:

- Definen la interacción que se produce entre un proceso **privado** y otro proceso o participante.

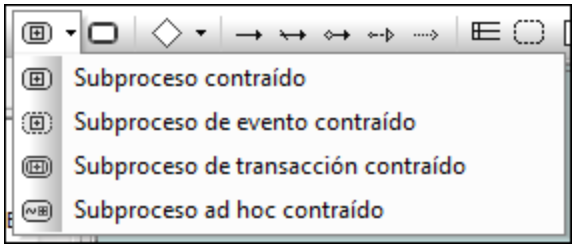


9.3.2.7.1 Tareas y subprocesos

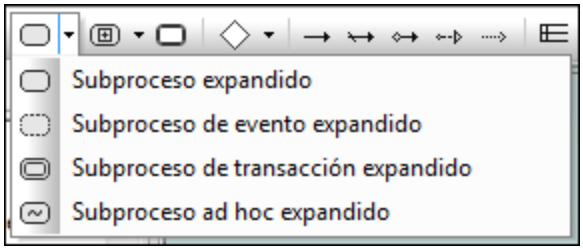
Haga clic en el icono desplegable **Tarea** para insertar una tarea determinada.



Haga clic en el icono desplegable **Subproceso contraído** para insertar un subproceso contraído determinado.



Haga clic en el icono desplegable **Subproceso expandido** para insertar un subproceso expandido determinado.



9.3.2.7.2 Objetos de datos

Los datos se representan por medio de cinco elementos de modelado y se insertan con ayuda de estos iconos:

Objeto de datos

Representa la información que fluye a través de los procesos (p. ej. correos electrónicos, documentos comerciales, etc.). Los objetos de datos ofrecen información sobre qué actividades deben llevarse a cabo y qué producen.

Datos de salida

Representa el resultado del proceso.

Datos de entrada

Es una entrada externa para todo el proceso. Pueden ser leídos por una actividad.

Colección de datos

Representa una colección de información (p. ej. una lista de pedidos).

Almacén de datos

Una ubicación donde el proceso puede leer o escribir datos (p. ej. una base de datos).

9.3.3 Diagramas SysML

Sitio web de Altova: Modelar diagramas SysML en UModel

SysML es un lenguaje de modelado gráfico que permite analizar, especificar, diseñar, verificar y validar sistemas como hardware, software, datos y procedimientos, entre otros. En UModel puede crear diagramas SysML desde cero o puede importar o exportar modelos SysML con XMI, consulte XMI: intercambio de metadatos XML.

Esta es una tabla con los diagramas que están disponibles en SysML.

Tipo	Diagrama	Notas	Abreviación
Estructura	Diagrama de definición de bloques	Modificado desde UML	bdd
	Diagrama de bloque interno	Modificado desde UML	ibd
	Diagrama de paquetes	Reutilizado desde UML	pkg
	Diagrama paramétrico	Específico de SysML	par
Requisito	Diagrama de requisitos	Específico de SysML	req
Comportamiento	Diagrama de actividades	Modificado desde UML	act
	Diagrama de secuencia	Reutilizado desde UML	sd
	Diagrama de máquina de estados	Reutilizado desde UML	stm
	Diagrama de casos de uso	Reutilizado desde UML	uc

Como se ve en la tabla, los diagramas SysML se pueden clasificar a grandes rasgos en diagramas de estructura, requisitos y comportamiento. Algunos de los diagramas SysML se reutilizan desde UML, algunos se modifican desde UML y algunos son específicos de SysML solamente. La abreviación de cada diagrama aparece por defecto en la esquina superior izquierda de la Ventana de diagramas, a no ser que decida ocultar el encabezado del diagrama.

Aparte de las características específicas de cada diagrama, diseñar proyectos SysML con UModel no es diferente que diseñar proyectos UModel estándar, véase Crear, abrir y guardar proyectos. Puede encontrar un proyecto de UModel de ejemplo que incluye varios diagramas SysML en: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Bank_SysML.ump**.

Crear diagramas SysML

Para que pueda crear diagramas SysML un proyecto de UModel debe incluir el *perfil SysML*, que es un perfil integrado de UModel. La aplicación le pedirá que incluya este perfil la primera vez que añada un diagrama SysML a un proyecto, como explicamos a continuación. También puede añadir el perfil SysML a un proyecto de forma explícita, véase Aplicar perfiles de UModel.

Para crear un diagramaSysML:

1. Elija una opción:
   a. Haga clic con el botón derecho en un paquete de la Ventana Estructura del modelo y seleccione **Diagrama nuevo | Diagramas SysML | <tipo de diagrama>** en el menú contextual, donde "tipo de diagrama" es uno de los tipos de diagrama SysML.
   b. Haga clic con el botón derecho en "Diagramas" or "Diagramas SysML" en la Ventana Árbol de diagramas y seleccione **Diagrama nuevo | <tipo de diagrama>** en el menú contextual, donde "tipo de diagrama" es uno de los tipos de diagrama SysML. Se abre un cuadro de diálogo en el que debe definir el propietario del diagrama. Seleccione el paquete en el que se debe guardar el diagrama y haga clic en **Aceptar**.
2. Si el proyecto actual de UModel no incluye el perfil SysML aparecerá un cuadro de diálogo que le pedirá que lo incluya. Haga clic en **Aceptar** para incluir el perfil SysML en el proyecto, véase también Aplicar perfiles de UModel.

Nota: si seleccionó el paquete "root" en el paso 1, los diagramas SysML se crean en su propio paquete "SysML".

9.3.3.1 Diagrama de definición de bloques

Los diagramas de definición de bloques están basados en los diagramas de clases UML pero las restricciones y extensiones vienen definidas por el estándar SysML. Este tipo de diagramas describe la relación que existe entre varios bloques, entre sus asociaciones, generalizaciones y dependencias.

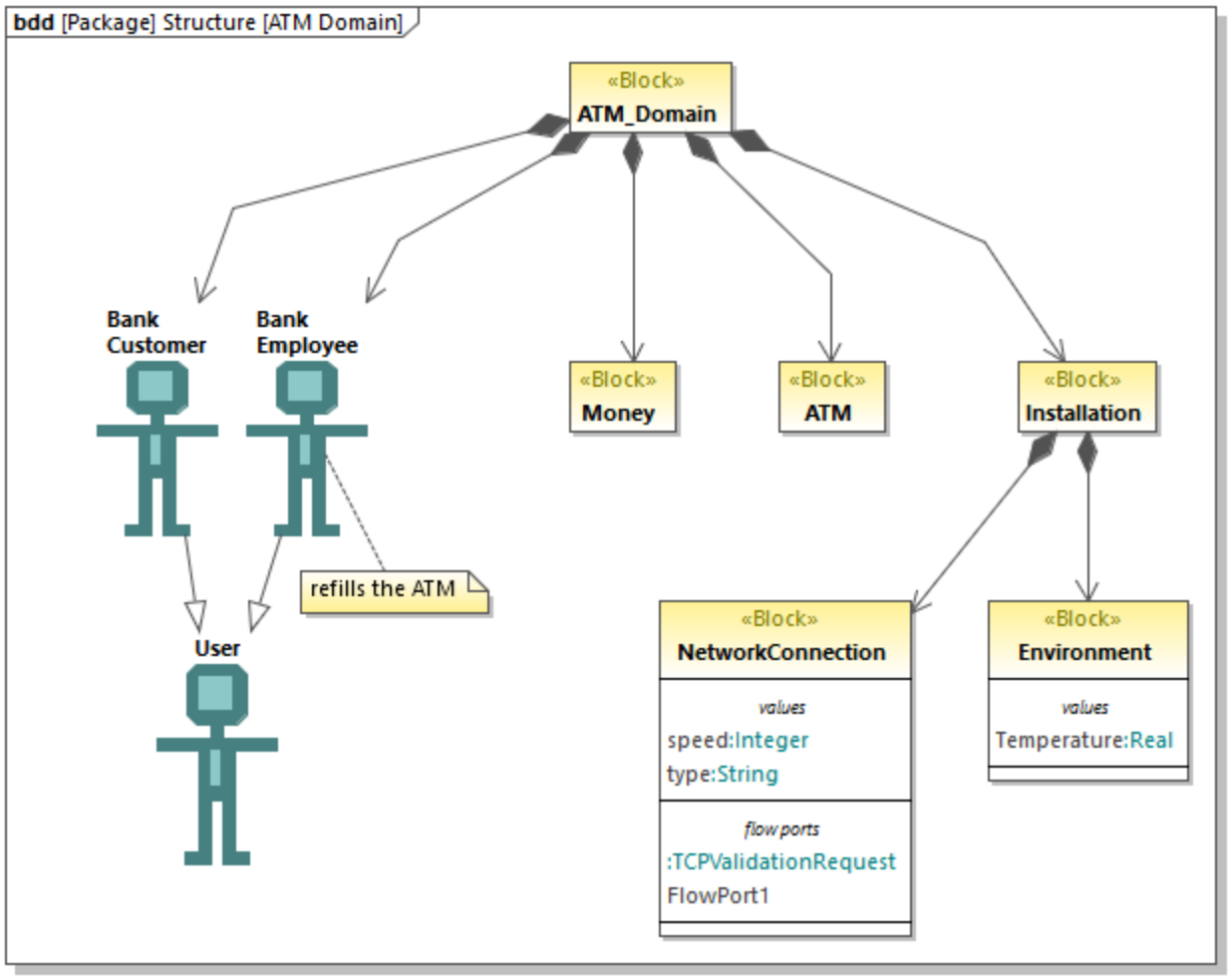


Diagrama de definición de bloques

Los bloques son unidades fundamentales para describir una estructura en SysMl y se parecen a las clases de los diagramas de clases UML. Los bloques pueden incluir componentes como partes, operaciones, propiedades y puertos. Una propiedad puede estar especializada; por ejemplo, puede ser una `PropiedadPartes`, una `PropiedadReferencias` o una `PropiedadValores`.

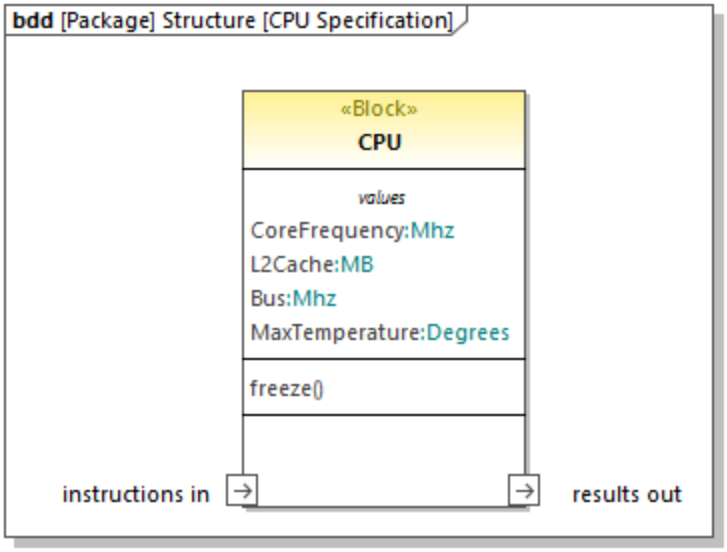
Para crear un bloque:

1. Cree un diagrama de definición de bloques nuevo, véase Crear diagramas SysML.
2. Elija una opción:
   - Haga clic con el botón derecho en un área del diagrama y seleccione **Nuevo/a | Bloque** en el menú contextual.
   - Haga clic en el botón **Bloque** de la barra de herramientas y después haga clic dentro del diagrama.

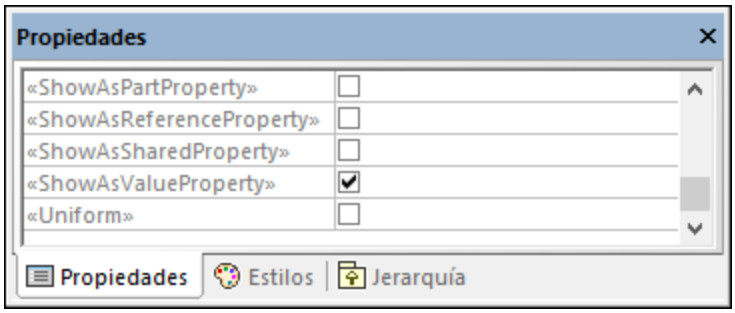
Para añadir una propiedad a un bloque:

- Haga clic con el botón derecho en un bloque y seleccione **Nuevo/a | Propiedad** (o **PropiedadPartes**, **PropiedadReferencias**, **PropiedadValores**, según el caso) en el menú contextual.

  Se añade un compartimento nuevo al bloque, por ejemplo, "partes" para **PropiedadPartes** o "valores" para **ValueProperty**.

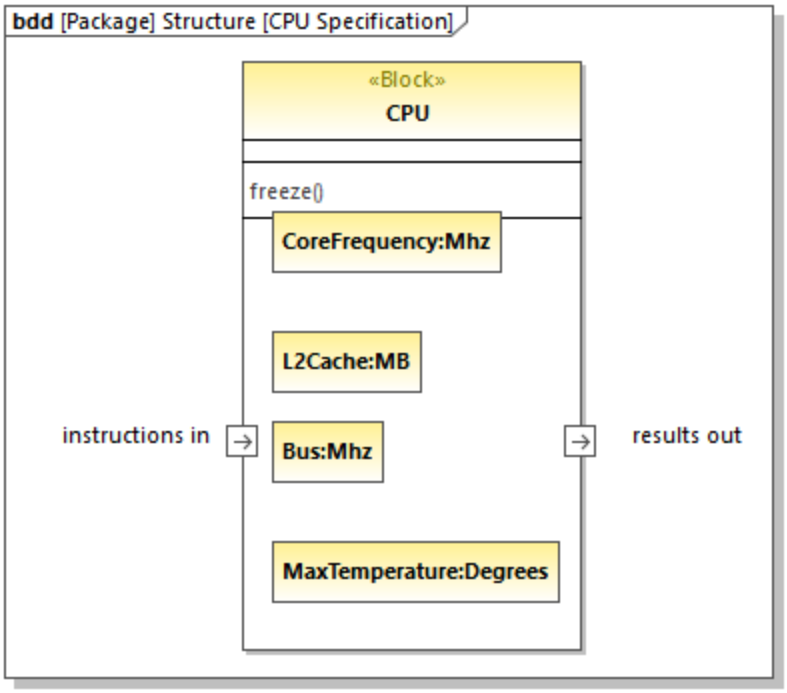


Puede cambiar la especialización de una propiedad en cualquier momento (por ejemplo, puede convertir **PartProperty** en **ValueProperty**). Para ello seleccione primero la propiedad en el diagrama o en la Estructura del modelo y después marque la casilla del estereotipo relevante en la ventana Propiedades, por ejemplo:



Para mostrar las propiedades de un bloque como nodos:

- Haga clic con el botón derecho y seleccione **Mostrar | Mostrar propiedades en forma de nodos en el nodo**.



Para deshacer la acción anterior basta con hacer clic con el botón derecho en una propiedad (por ejemplo, `Bus:Mhz` en la imagen anterior) y seleccionar **Eliminar sólo en el diagrama** en el menú contextual.

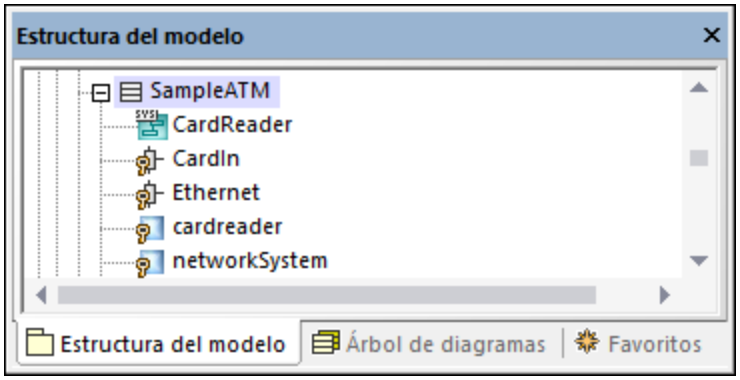
9.3.3.2 Diagrama de bloque interno

Los diagramas de bloques internos están basados en los diagramas de estructura compuesta pero las restricciones y extensiones vienen definidas por el estándar SysML.

Este tipo de diagramas describe la estructura interna de un bloque en cuanto a las propiedades y los conectores que existen entre ellas (es decir, la relación de sus partes constituyentes). Para ello se sirven de puertos, conectores y flujos. Esta es la forma habitual de crear un diagrama de bloqueo interno nuevo:

- Haga clic con el botón derecho en un bloque ya existente en la Estructura del modelo y seleccione **Diagrama nuevo | Diagramas de bloque interno SysML** en el menú contextual.

Si crea un diagrama de bloque interno nuevo sin hacer primero clic en un bloque que ya existe se creará también un bloque en la Estructura del modelo y el diagrama nuevo se anidará bajo ese bloque porque se asume que es el que lo describe. Por ejemplo, en la imagen siguiente el diagrama "CardReader" describe el bloque "SampleATM".



Puede encontrar el diagrama "CardReader" en este proyecto de ejemplo: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Bank_SysML.ump**.

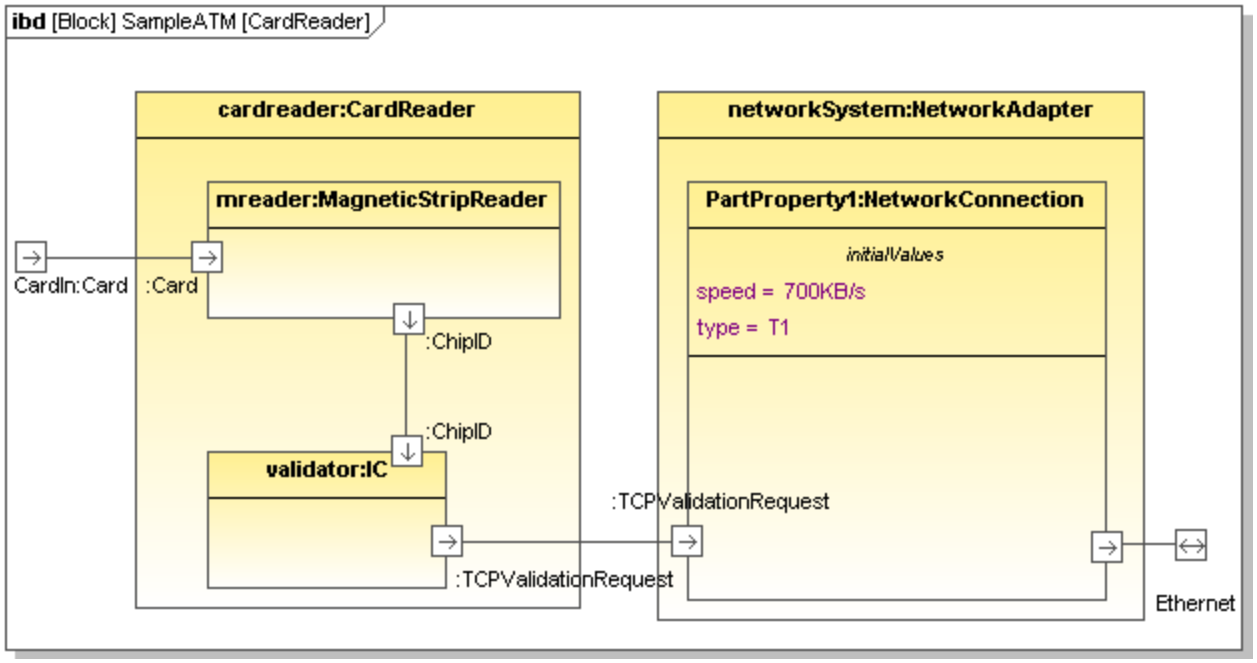


Diagrama CardReader

Las propiedades `cardreader` y `networksystem` de la imagen anterior tienen como tipo, respectivamente, `CardReader` y `NetworkAdapter`. Estos tipos existen en el mismo modelo y son bloques, lo que determina el aspecto de las propiedades en el diagrama. Tenga en cuenta que puede definir o cambiar el tipo de una propiedad desde la lista desplegable **tipo** en la ventana Propiedades.

Valores iniciales

Cuando una propiedad tiene un tipo que es un bloque, como en este ejemplo, se le pueden dar unos valores iniciales. Por ejemplo, la propiedad **`PartProperty1`** del diagrama **CardReader** tiene como valor inicial `speed = 700KB/s`.

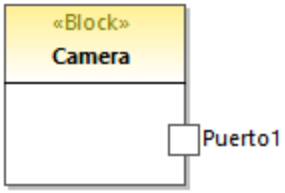
Para añadir valores iniciales a una propiedad:

1. Haga clic con el botón derecho en la propiedad y elija la opción **Nuevo/a | Valores iniciales** en el menú contextual.
2. Haga doble clic en el marcador de posición y escriba los valores (p. ej. velocidad = 700KB/s).

Puertos estándar

Para insertar un puerto estándar haga clic en el botón **Puerto** de la barra de herramientas y después en el diagrama. Esto añade el puerto al diagrama.

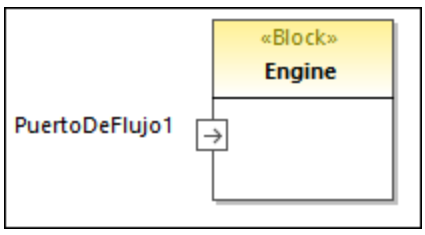
Para anexar el puerto a un bloque arrástrelo hasta el borde del bloque ("Camera" en este ejemplo) y suéltelo cuando se resalte el borde. Ahora el puerto está anexado al borde del bloque.



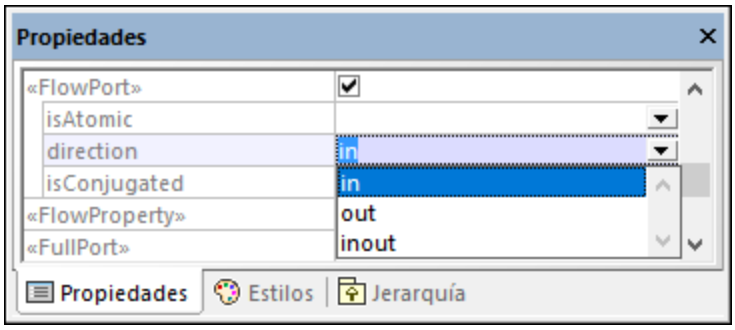
Para cambiar el nombre y el tipo del puerto primero debe seleccionar el puerto en el diagrama y después cambiar las propiedades `nombre` y `tipo` en la Ventana Propiedades.

Puertos de flujo

Para crear un puerto de flujo haga clic en el botón **PuertoDeFlujo** de la barra de herramientas y después haga clic en el borde de un bloque. Esto anexa el puerto de flujo al borde del bloque. También puede crear y anexar puertos de flujo en dos pasos, como se explica más arriba para los puertos estándar.

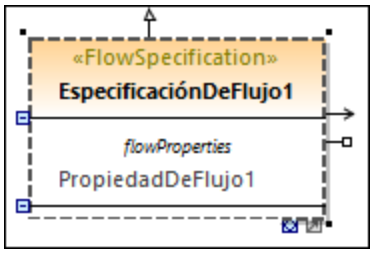


Para cambiar el nombre y el tipo del puerto edite las propiedades respectivas en la ventana Propiedades. Observe que los puertos de flujo tienen más propiedades en la ventana Propiedades con las que puede indicar la dirección (direction), por ejemplo in, out, inout.



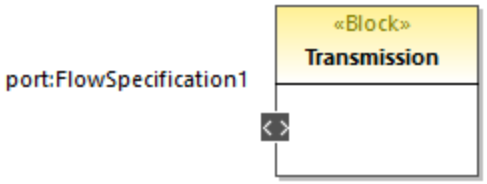
Para crear un puerto de flujo conjugado atómico:

1. Cree una `EspecificaciónDeFlujo` (interfaz) en un diagrama de definición de bloques (BDD).



2. En el diagrama de bloques internos (IBD) haga clic en el puerto de flujo que desea definir.
3. En la ventana Propiedades elija como tipo de puerto de flujo la `EspecificaciónDeFlujo` que creó en el primer paso.
4. En la ventana Propiedades elija el valor `true` para la propiedad `isConjugated`.

El puerto de flujo conjugado atómico tiene un icono con un fondo oscuro.



Unir puertos

Para unir dos puertos siga estos pasos:

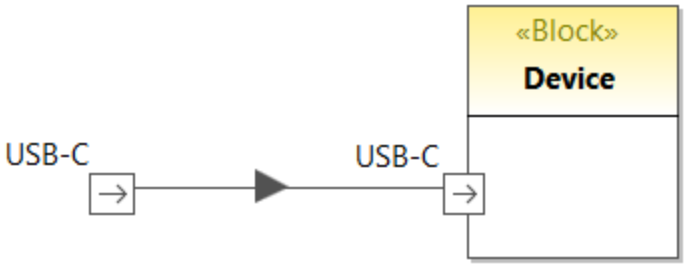
1. Haga clic en el icono **Conector**  de la barra de herramientas.
2. Haga clic en el primer puerto y arrastre el puntero hasta el segundo puerto.

3. Cuando el puerto de destino aparezca resaltado, suelte el conector.

Crear el flujo del elemento

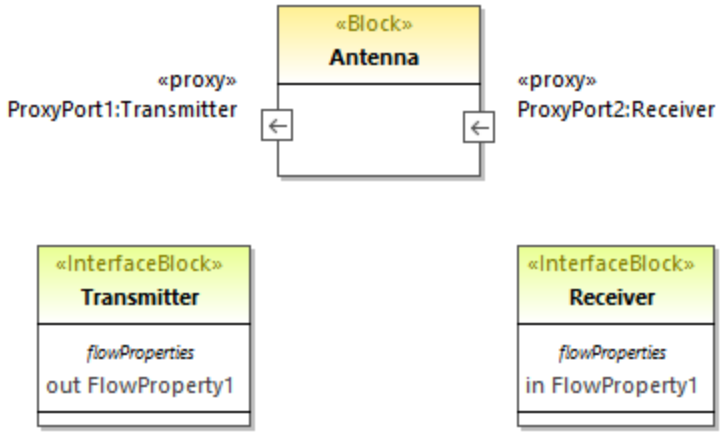
En los diagramas SysML puede crear flujos de elemento entre las asociaciones de bloques y en los demás conectores.

Para ello haga clic con el botón derecho en un conector y elija el comando **Nuevo/a | Flujo del elemento (de derecha a izquierda) / Flujo del elemento (de izquierda a derecha)**. Al conector se le añade una punta de flecha que indica la dirección del flujo del elemento.



Proxy ports and direction Puertos proxy y dirección

En las versiones más recientes de SysML los puertos proxy pueden indicar una dirección de forma parecida a los puertos de flujo de las versiones antiguas. Por ejemplo, el diagrama siguiente consiste en un bloque ("Antenna") con dos puertos proxy que indican una dirección.



Un ejemplo de cómo puede añadir una dirección a los puertos proxy:

1. Añada un bloque y dos bloques de interfaz al diagrama. En el ejemplo anterior el bloque sería "Antenna" y los dos bloques de interfaz serían "Transmitter" y "Receiver".
2. Seleccione "Transmitter" y pulse **F7** para añadir una propiedad de flujo nueva.
3. Añada un puerto proxy al bloque y cambie su tipo a "Transmitter" en la ventana Propiedades.
4. Seleccione la propiedad de flujo en el diagrama y, desde la ventana Propiedades, cambie la propiedad de dirección a `out`. Observe que la dirección del puerto proxy cambia y refleja esta acción.

5. Seleccione "Receiver" y pulse **F7** para añadir una propiedad de flujo nueva.
6. Añada un segundo puerto proxy al bloque y cambie su tipo a "Received" en la ventana Propiedades.
7. Seleccione la propiedad del flujo en el diagrama y, en la ventana Propiedades, cambie la propiedad de dirección a `in`. La dirección del puerto proxy cambia y refleja esta acción.

9.3.3.3 Diagrama paramétrico

El diagrama paramétrico es un tipo de diagrama propio de SysML que combina análisis técnico con modelado de diseños. Los diagramas paramétricos son similares a los diagramas de bloques internos, con la diferencia de que los paramétricos solamente muestran los conectores que están conectados a parámetros de restricción en uno de sus extremos.

Los diagramas paramétricos se sirven de bloques de restricción, definidos en un diagrama de definición de bloques, para limitar las propiedades de otros bloques en el diagrama paramétrico. Los bloques de restricción se dibujan con esquinas redondeadas y no con recuadros.

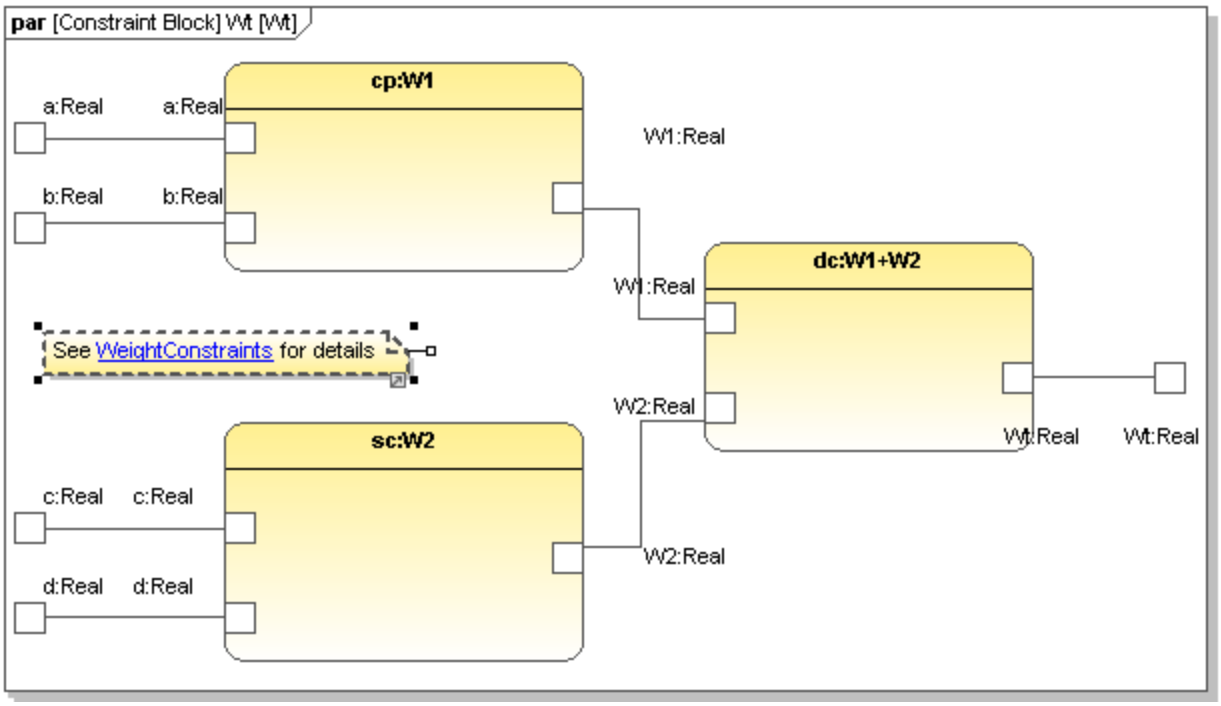
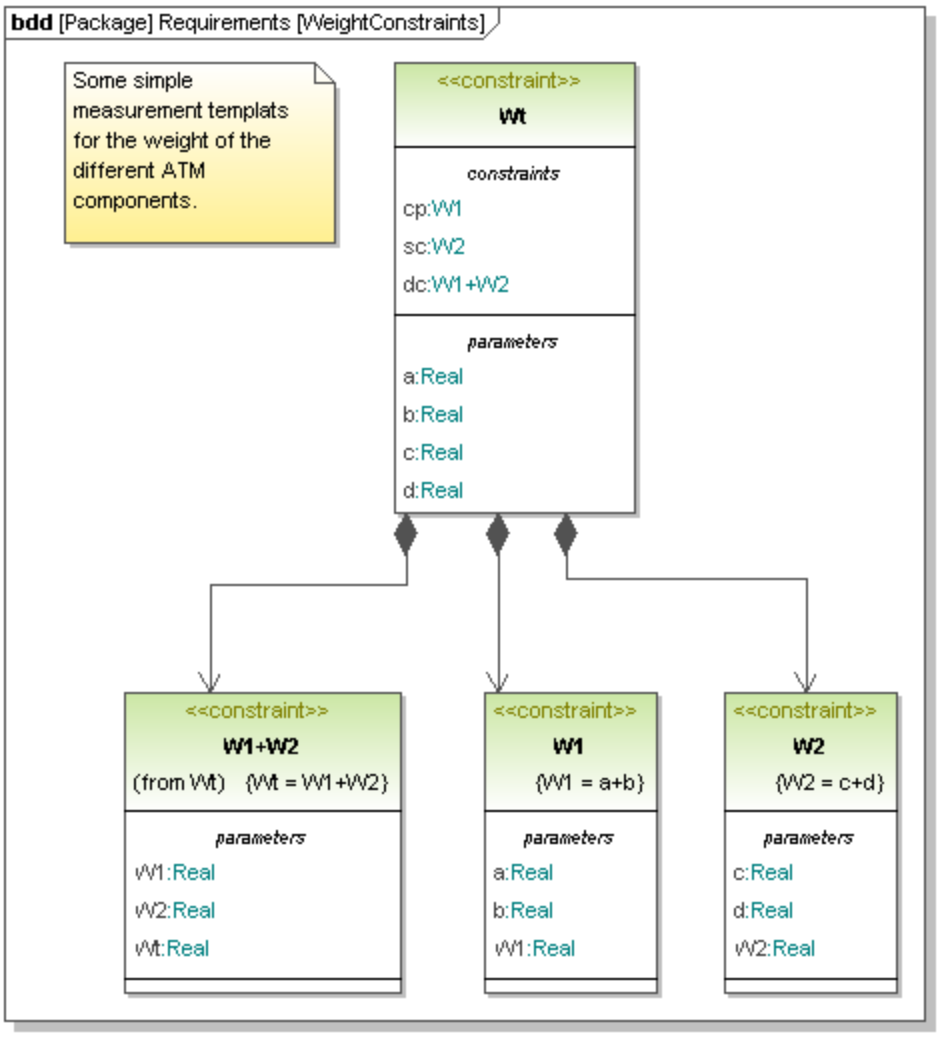


Diagrama paramétrico Wt

La palabra clave «`constraint`» de una definición de bloque indica que el bloque es un bloque de restricción. Los parámetros de la restricción aparecen en el compartimento *parámetros*.



9.3.3.4 Diagrama de paquetes

Los diagramas de paquetes sirven para organizar elementos de un modelo en paquetes. En este tipo de diagramas también puede definir las dependencias entre los paquetes y los elementos de modelado de dentro de un paquete. Por ejemplo, el diagrama siguiente se puede ver la organización del modelo definido en el proyecto de ejemplo **Bank_SysML.ump**, que está en el directorio **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**. Los enlaces de los elementos **Requirements**, **Structure** y **Use Cases** apuntan a los respectivos paquetes dentro de ese mismo modelo; consulte también Agregar hipervínculos a elementos.

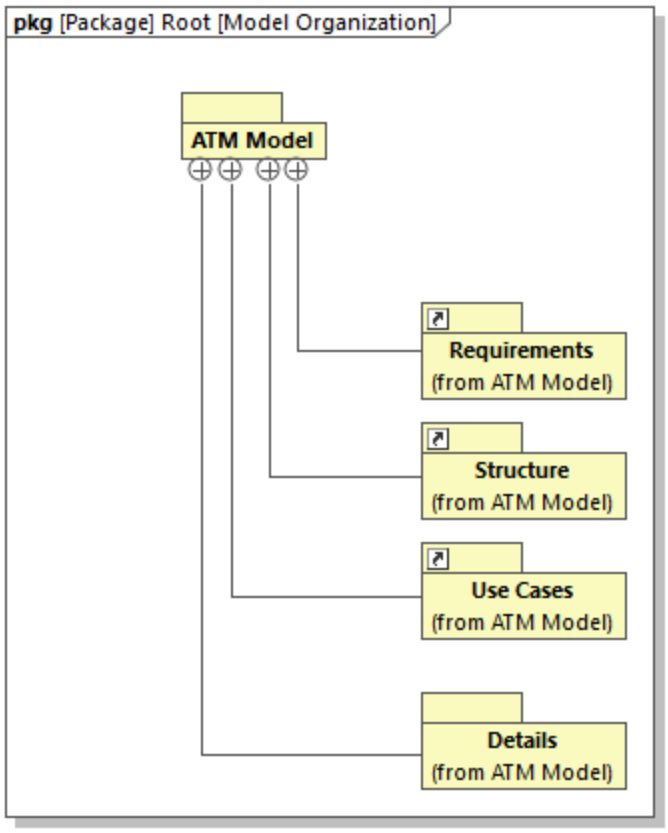
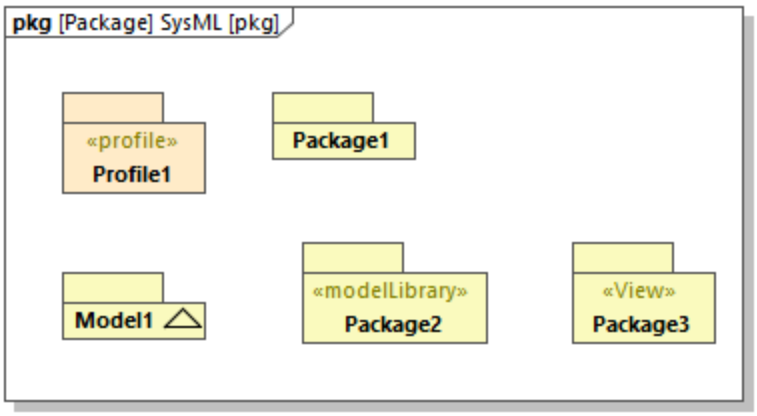


Diagrama de paquetes

El paquete de diagramas de la imagen anterior es solo una de las formas en que se puede organizar un modelo; también puede organizar modelos por jerarquía o tipo de diagrama, por ejemplo.

Puede añadir varios elementos a un diagrama de paquetes haciendo clic en los botones correspondientes de la barra de herramientas (como **Paquete**, **Perfil** o **Vista**) y después haciendo clic dentro del diagrama. Sin embargo, no todos los tipos de diagramas tienen este tipo de botones; en ese caso puede añadir elementos siguiendo estos pasos:

1. Haga clic en el botón **Paquete** de la barra de herramientas y después haga clic dentro de diagrama para añadir un paquete nuevo.
2. En la ventana Propiedades marque la casilla de selección con el estereotipo deseado (por ejemplo, «`ModelLibrary`»).



En el diagrama de paquetes anterior, Package 2 tiene el estereotipo «ModelLibrary» y Package 3 el estereotipo «View». Consulte también Aplicar estereotipos.

9.3.3.5 Diagrama de requisitos

El diagrama de requisitos es un tipo de diagrama diseñado para SysML que combina los modelos de comportamiento y de estructura de SysML con modelos de análisis técnico (como modelos de rendimiento, de fiabilidad, etc.).

Se trata de una construcción de modelado para requisitos textuales y para la relación entre los requisitos y los demás elementos de modelado que los cumplen.

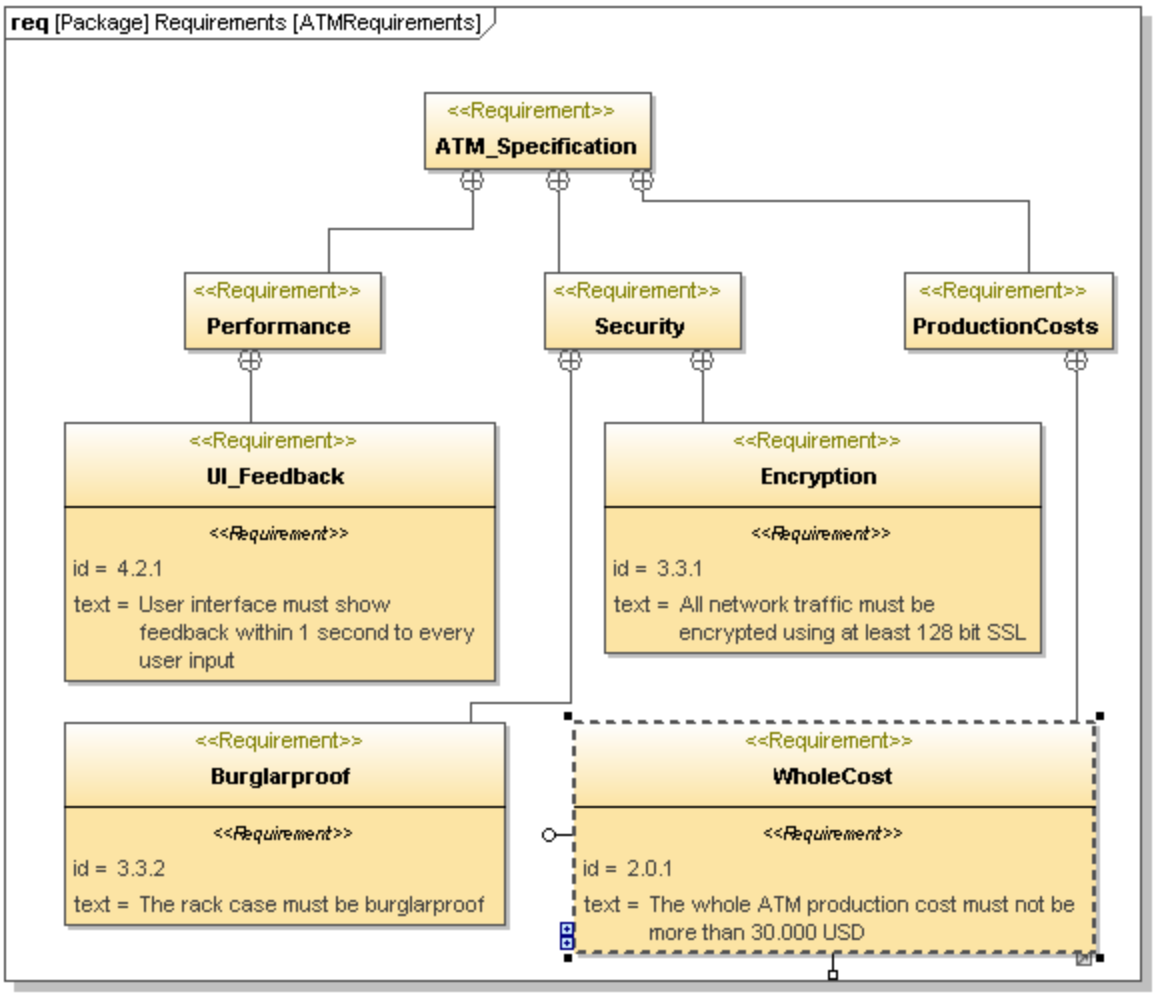


Diagrama de requisitos

Con los diagramas de requisitos a menudo es necesario crear textos de varias líneas para mantener el tamaño de los bloques de requisitos no exceda lo razonable.

Para crear un texto con varias líneas:

1. Haga doble clic en el texto.
2. Mantenga pulsada la tecla **Ctrl** y pulse **Entrar**.

9.3.3.6 Diagrama de actividades

Los diagramas de actividades SysML expresan información sobre el comportamiento dinámico de un sistema, como el flujo de objetos durante la operación del sistema. Este tipo de diagramas muestran el orden en que se realizan las acciones y qué estructura lleva a cabo qué acción. Los flujos también pueden ser flujos de control

o de objetos. Puede añadir cualquiera de estos tipos de flujo con los botones correspondientes de la barra de herramientas:

→ **Flujo de control**

⊢→| **Flujo de objetos**

El diagrama de actividades de ejemplo de la imagen siguiente usa estos dos flujos. El flujo de control lo representan las líneas discontinuas y el flujo de objetos lo representan las líneas ininterrumpidas.

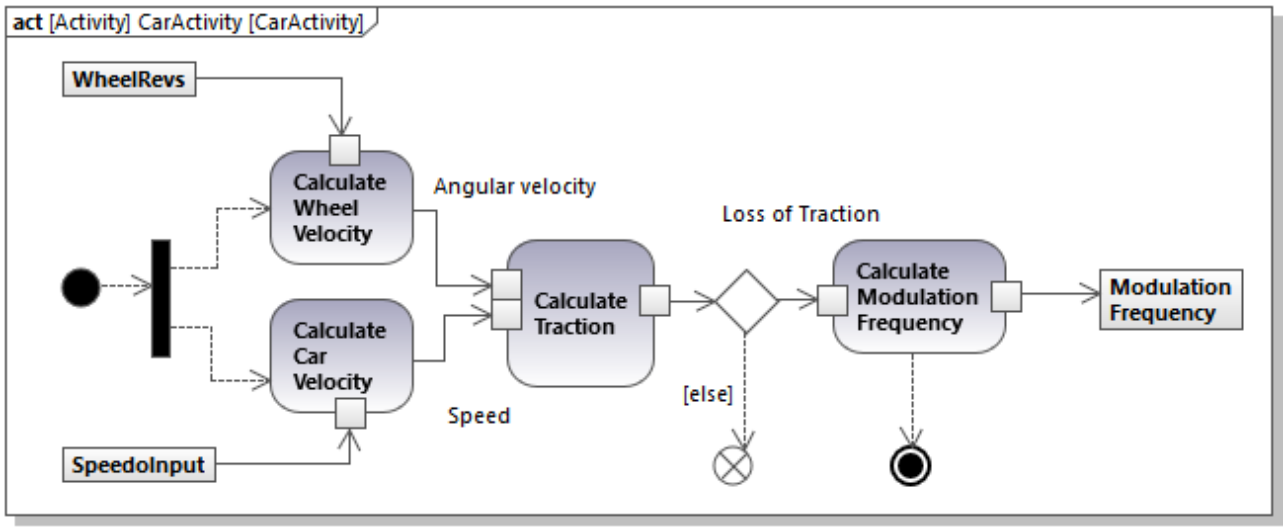


Diagrama de actividades SysML

El diagrama de actividades SysML se modifica con UML y sus extensiones SysML. Para información general sobre cómo diseñar diagramas de actividades UML con UModel consulte Diagrama de actividades.

9.3.3.7 Diagrama de secuencia

Los diagramas de secuencia SysML describen el comportamiento dinámico de un sistema, al igual que los diagramas de actividad, pero de forma más precisa que estos. No solo informa sobre el *orden* de las acciones y qué estructuras llevan a cabo las acciones, sino que también ofrece información sobre las estructuras que invocan una acción en concreto. Por eso los diagramas de secuencia acaban siendo complejos a no ser que se centren en un caso muy específico. La imagen siguiente muestra un fragmento de un diagrama de secuencia SysML del proyecto de ejemplo **Bank_SysML.ump**.

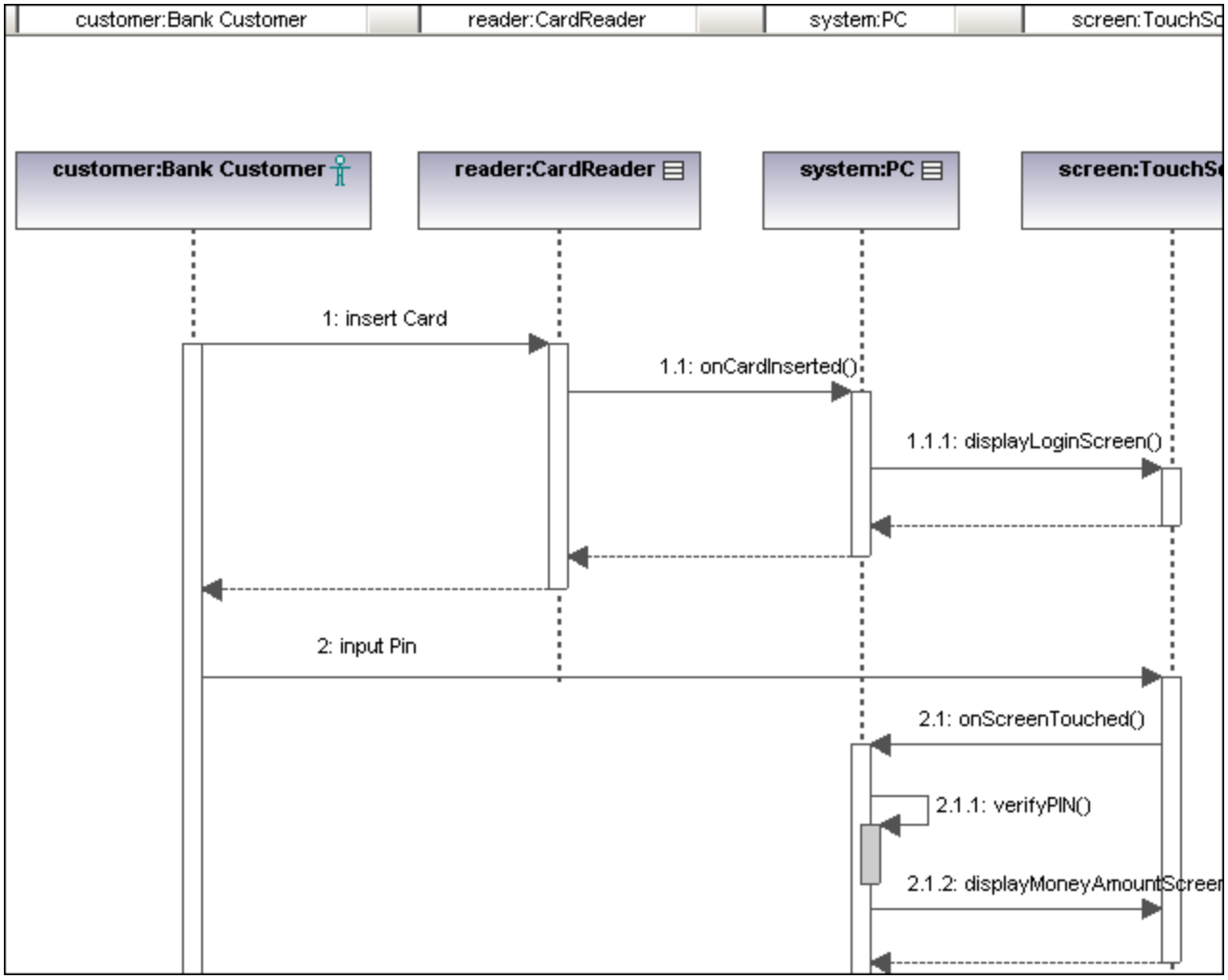


Diagrama de secuencia SysML

Los diagramas de casos de uso SysML siguen la especificación UML. Diseñar este diagrama en UModel no requiere ningún conocimiento específico si sabe crear diagramas UML de secuencia estándar. Para más información sobre estos últimos consulte Diagrama de secuencia.

9.3.3.8 Diagrama de máquina de estados

Los diagramas de máquina de estados SysML ilustran las transiciones entre estados en un sistema en ejecución. Al igual que los diagramas de secuencia y actividad SysML, los diagramas de máquina de estados SysML también reflejan el comportamiento de un sistema.

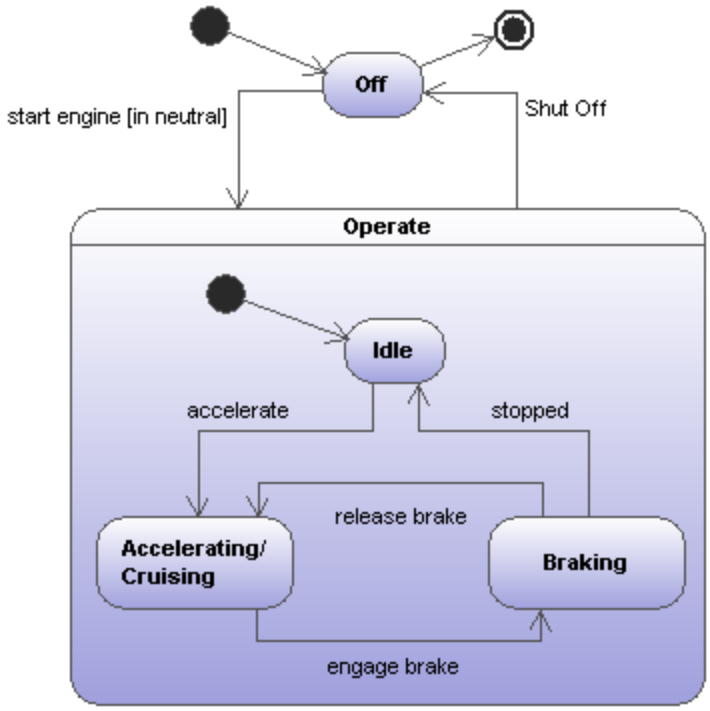


Diagrama de máquina de estados SysML

Los diagramas de casos de uso SysML siguen la especificación UML. Diseñar este diagrama en UModel no requiere ningún conocimiento específico si sabe crear diagramas UML de máquina de estado estándar. Para más información sobre estos últimos consulte Diagrama de máquina de estados.

9.3.3.9 Diagrama de casos de uso

Este diagrama de casos de uso SysML muestra los elementos y las relaciones que describen los servicios proporcionados en un sistema. También muestra varios actores (usuarios u operadores del sistema) que consumen esos servicios. En el proyecto de ejemplo **Bank_SysML.ump**, que se encuentra en el directorio **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples**, vemos estos ejemplos de servicios::

- Un cliente de un banco usa un cajero automático para sacar dinero
- Un empleado de un banco realiza tareas de mantenimiento en un cajero automático
- Un empleado de un banco rellena de dinero un cajero automático

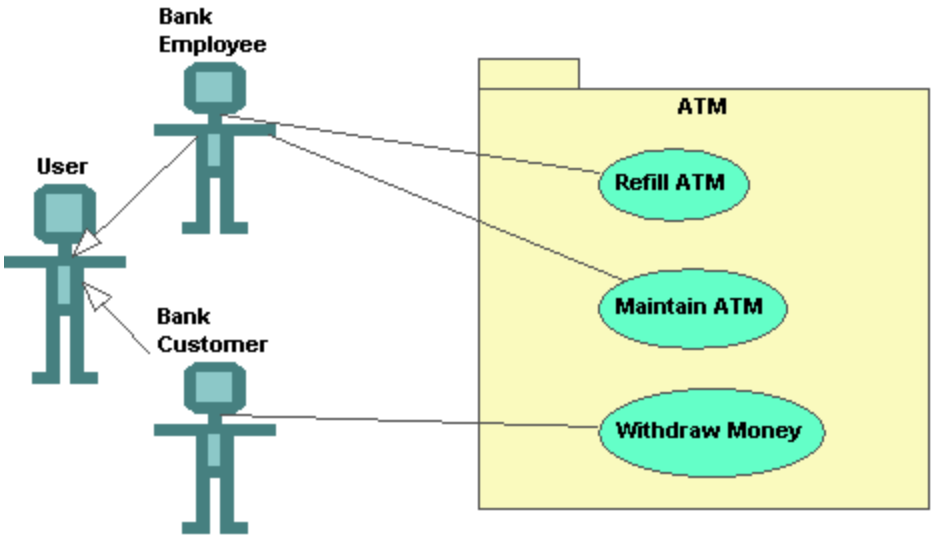
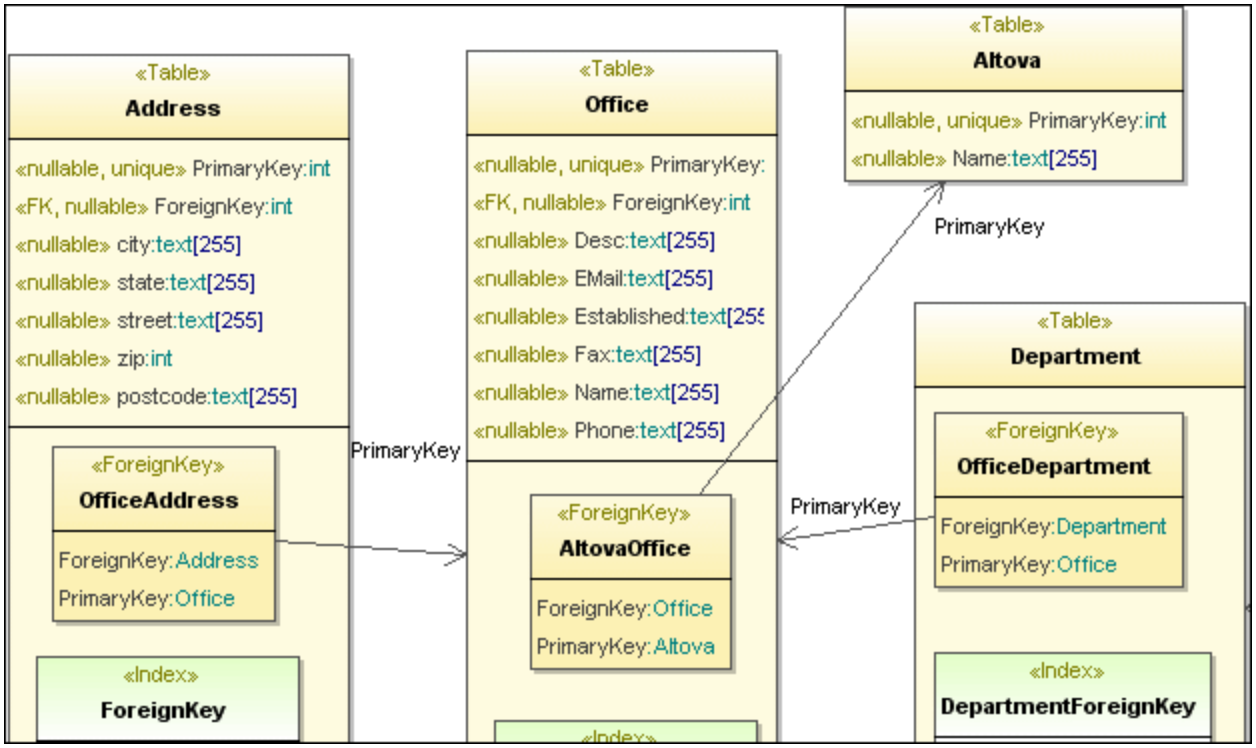


Diagrama de casos de uso SysML

Los diagramas de casos de uso SysML siguen la especificación UML. Para más información sobre este tipo de diagramas, consulte el apartado Casos de uso del *Tutorial*.

10 Trabajar con bases de datos en UModel

En UModel puede importar bases de datos SQL para ver su estructura o modificarla usando UML (*puede consultar la lista de bases de datos compatibles en el apartado* *Compatibilidad con bases de datos*). La estructura de la BD se puede consultar con facilidad en diagramas de base de datos UML como el que aparece a continuación:



Estos elementos de base de datos se pueden importar al modelo de base de datos:

- Tablas
- Restricciones de comprobación
- Claves principales, foráneas y únicas
- Índices
- Vistas
- Disparadores
- Procedimientos almacenados
- Funciones

Nota: las vistas, los disparadores, los procedimientos almacenados y las funciones no se pueden añadir en UModel, solamente se pueden importar.

Tras importar la estructura de la base de datos en UModel, podrá modificarla y aplicar los cambios realizados a la base de datos con el comando **Combinar el código de programa con el proyecto de UModel**. Esto crea un archivo script de cambios de base de datos que se puede ejecutar o guardar para ejecutarlo más tarde. Además, si se realizaron cambios en la base de datos desde la última vez que se sincronizó, podrá combinar estos cambios con el modelo (o sobrescribirlo con los cambios).

Para más información sobre las correspondencias entre elementos de base de datos y elementos de UModel consulte el apartado Correspondencias con elementos de BD.

10.1 Modelar bases de datos con UModel

Puede modelar bases de datos en UModel de una de las siguientes maneras:

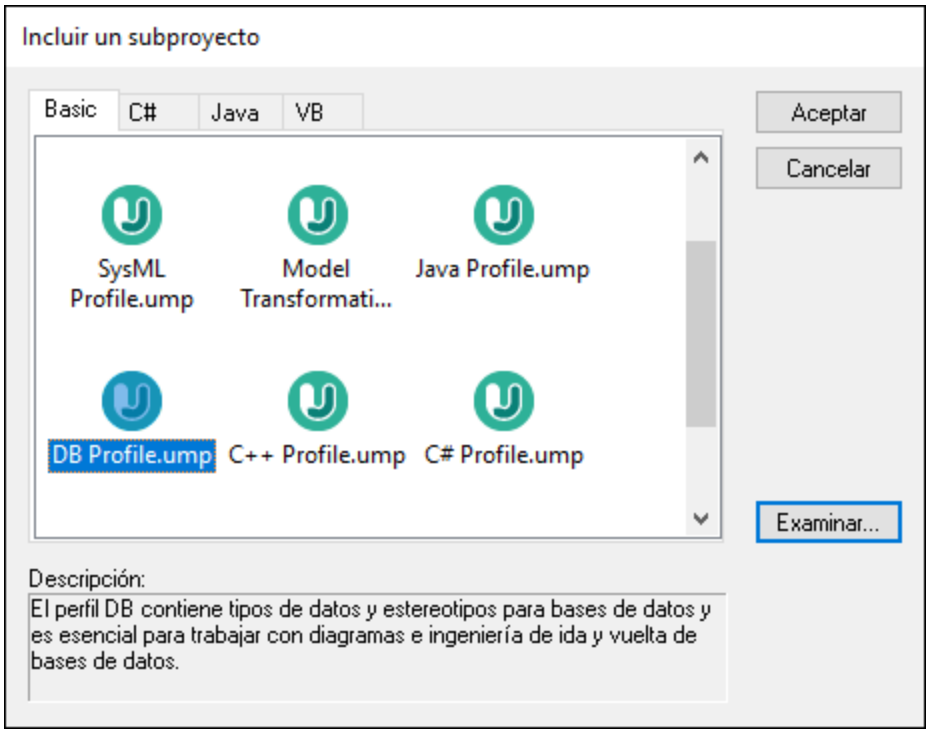
1. Sin ingeniería de código. En este caso se modelan los objectos de la base de datos sin conectarse a una BD real (por ejemplo, si solo quiere crear un diagrama que ilustre la estructura potencial de una BD).
2. Con ingeniería de código. En este caso se conecta a una BD, importa su estructura en el modelo y puede ver las definiciones de los objetos de BD directamente en UModel. Cuando lee la estructura de la BD, UModel puede generar automáticamente diagramas de BD. También puede modificar los objetos de la BD en el modelo (por ejemplo, añadir una tabla nueva o eliminar una que ya existe) y luego actualizar la BD real con scripts que genera UModel. La base de datos y el proyecto de UModel se pueden sincronizar en ambas direcciones, de forma parecida a como funciona la sincronización de lenguajes de programación. También tiene la opción de sincronizar únicamente los cambios (combinar) o sobrescribir todos los datos (sea en la BD con los datos del modelo o viceversa).

En los dos casos anteriormente descritos, el proyecto debe contener el perfil de BD que viene con UModel. Este perfil contiene todos los metadatos necesarios (como estereotipos UML) que le permiten ver o diseñar objetos de BD en UModel.

Si eligió trabajar con ingeniería de código, el perfil de BD y toda la configuración de ingeniería de código necesaria se añadirá automáticamente a su proyecto la primera vez que importe una BD en el modelo. En caso contrario deberá incluir el perfil de BD manualmente.

Para añadir manualmente el perfil de BD a un proyecto de UModel:

1. Cree un proyecto nuevo en UModel o abra uno que ya existe (*véase* *Crear, abrir y guardar proyectos*).
2. En el menú **Proyecto**, haga clic en **Incluir un subproyecto...**.



3. En la pestaña *Basic*, seleccione **DB Profile.ump** y haga clic en el botón **Aceptar**.

También puede elegir esta opción:

1. En la ventana Árbol de diagramas, haga clic en **Diagramas** y seleccione **Diagrama nuevo | Diagrama de base de datos**.
2. Cuando la aplicación lo solicite, seleccione el paquete al que debe pertenecer el nuevo diagrama.
3. Cuando la aplicación le informe de que se va a añadir el perfil de BD a su proyecto, haga clic en el botón **Aceptar**.

Ahora que se ha añadido el perfil de BD de UModel, ya puede empezar a modelar objetos de BD. Por ejemplo, al hacer clic con el botón derecho dentro de un diagrama de base de datos, el menú contextual ofrece opciones para crear una tabla nueva. Asimismo, al hacer clic con el botón derecho en una tabla, el menú contextual ofrece opciones para crear una columna, claves, índices, etc. Para más información, consulte Diseñar objetos de base de datos.

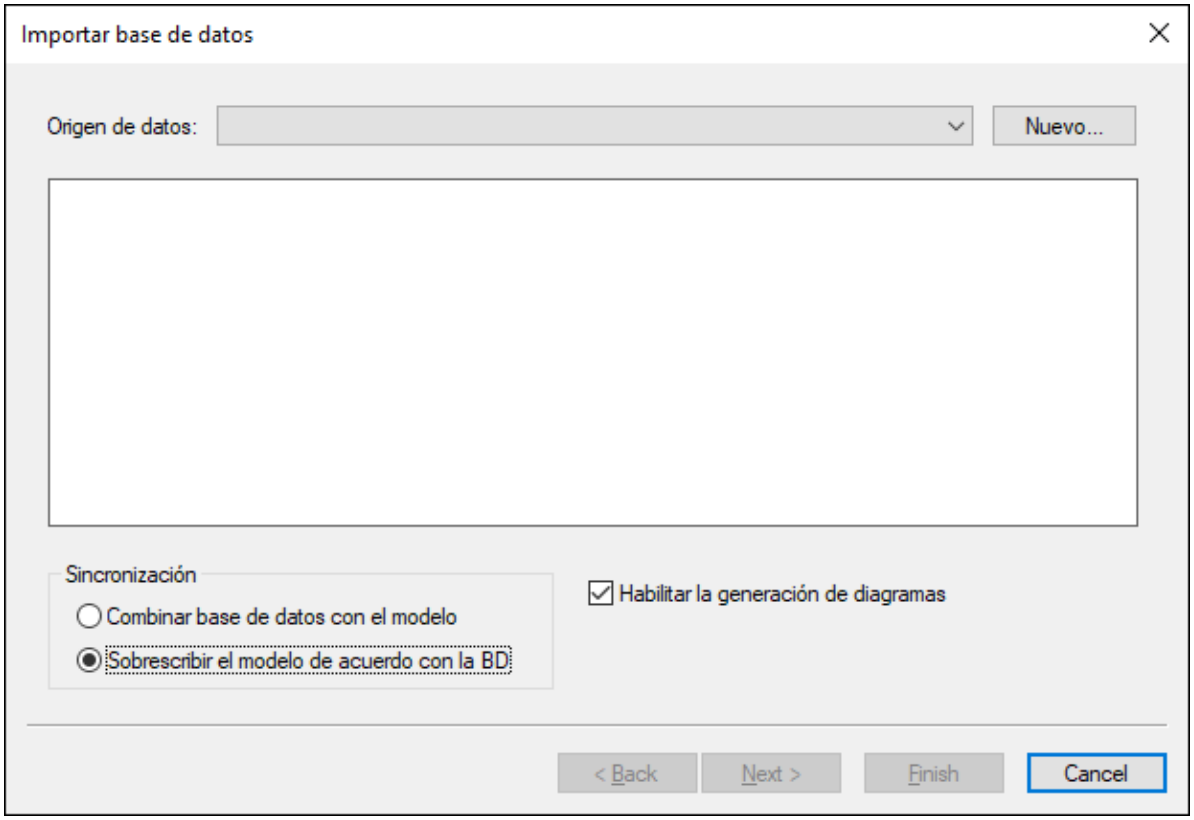
Para establecer una conexión a una base de datos y trabajar con ingeniería de código, consulte Importar bases de datos SQL en UModel.

10.1.1 Importar bases de datos SQL en UModel

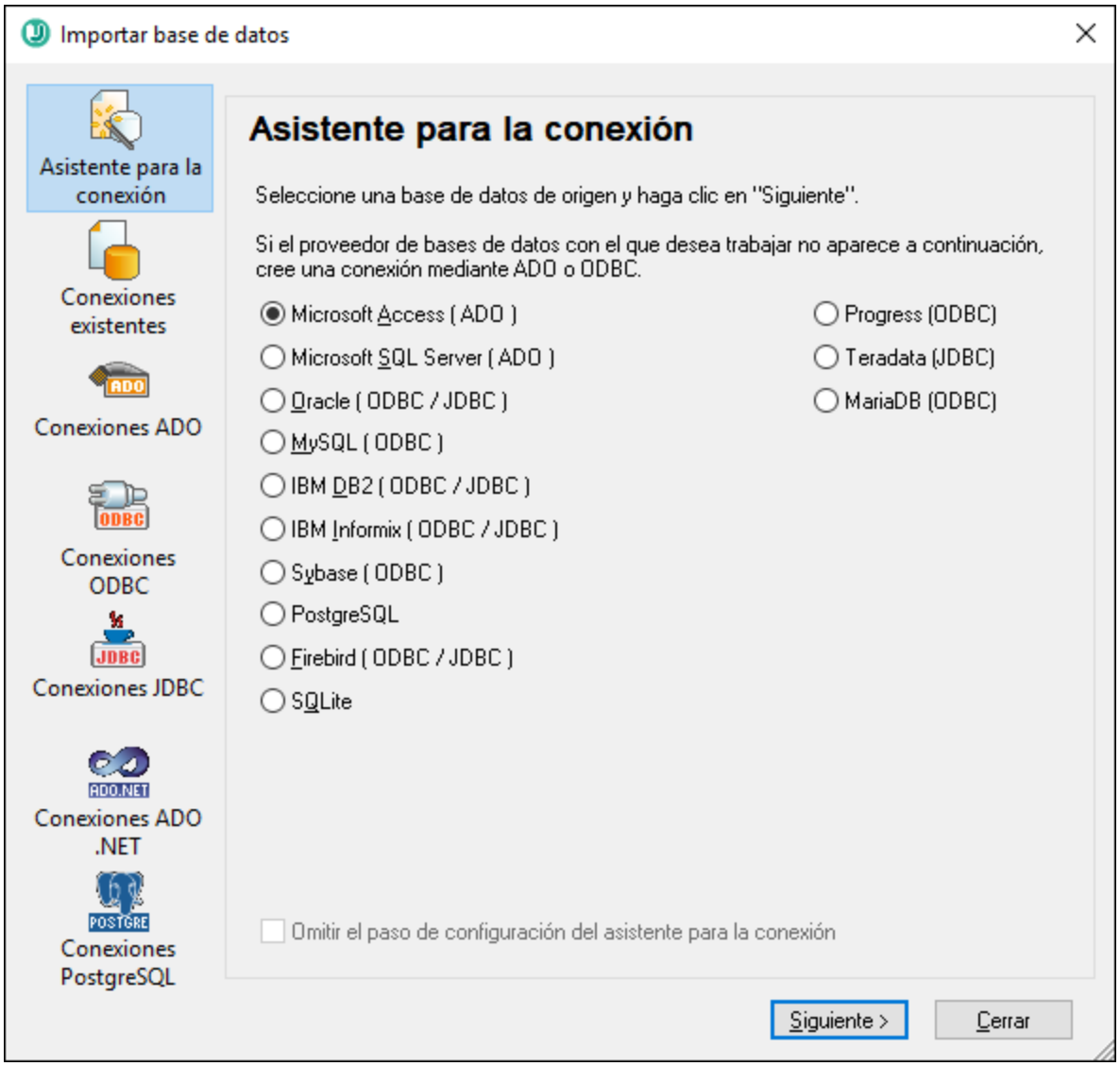
Estas instrucciones muestran cómo importar la estructura de una BD en UModel. También aprenderá a generar diagramas UML que ilustren la estructura de la BD. En este ejemplo usaremos una base de datos de ejemplo de Microsoft Access; sin embargo, los pasos sin muy similares para el resto de bases de datos con las que es compatible UModel.

Para importar una base de datos SQL en UModel:

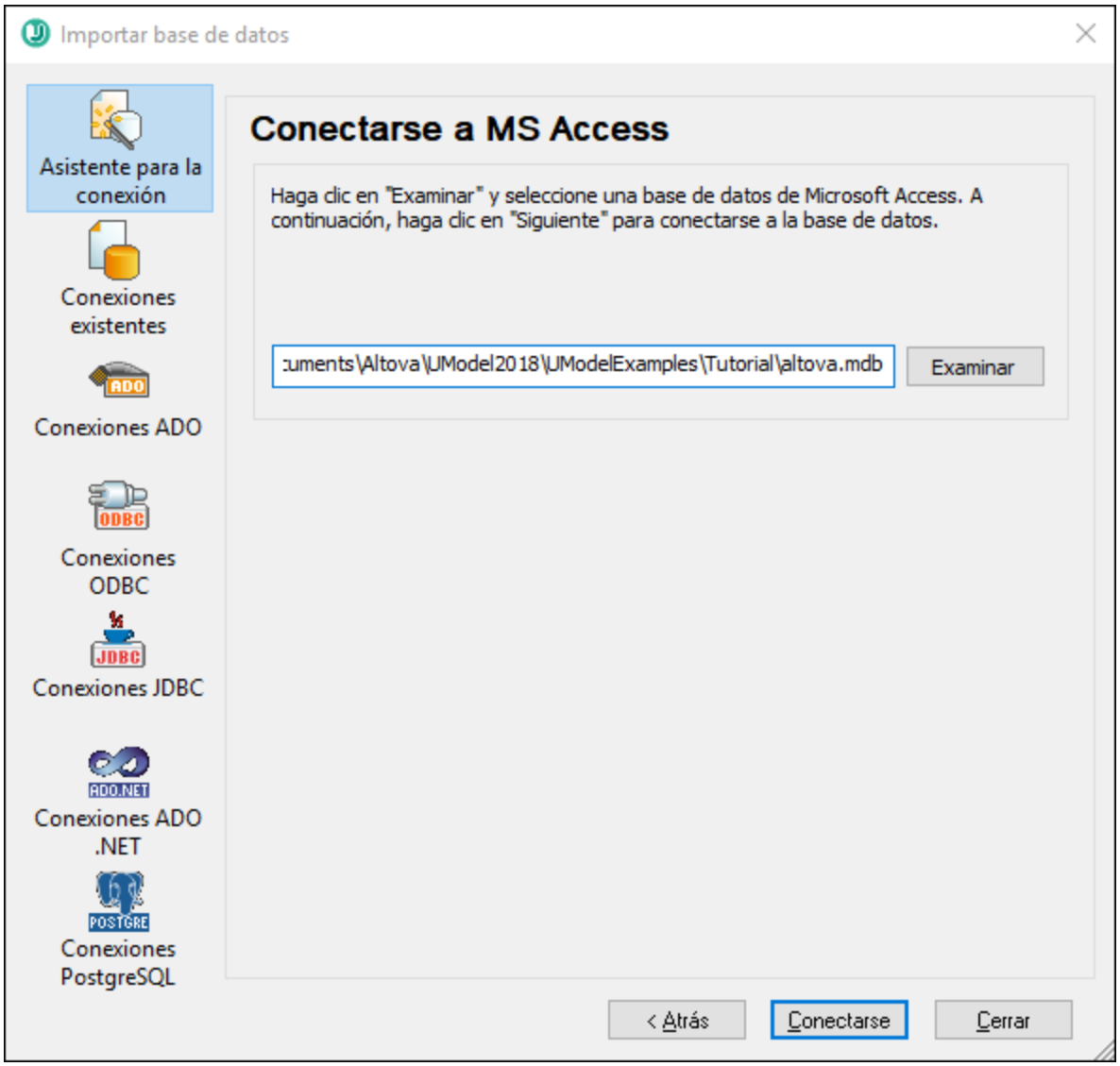
1. Seleccione el comando de menú **Proyecto | Importar bases de datos SQL...**.
2. Si es la primera vez que importa una BD en UModel, haga clic en el botón **Nuevo...** del cuadro de diálogo. Si ya importó una BD anteriormente, haga clic en el cuadro combinado *Origen de datos* y seleccione la BD.



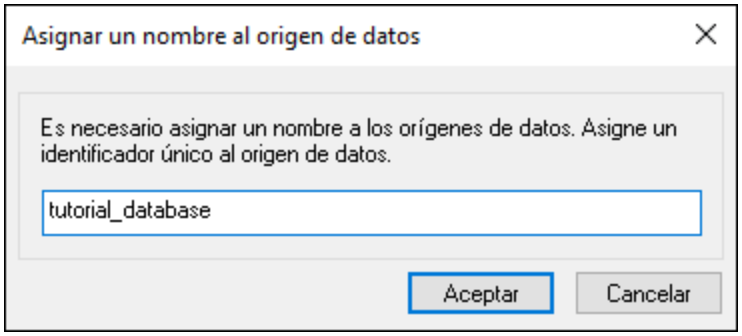
3. En este ejemplo nos conectaremos a una BD local de Microsoft Access. Por lo tanto, debe seleccionar **Microsoft Access (ADO)** como tipo de BD. Haga clic en el botón **Siguiente**. Para usar otro tipo de BD, seleccione la base de datos que necesite y siga las instrucciones del asistente. Según el tipo de BD, puede que necesite un controlador de BD para poder conectarse. Para ver un ejemplo concreto, consulte Ejemplos de conexión a bases de datos.



4. Navegue hasta el archivo de BD **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Tutorial\altova.mdb** y haga clic en **Conectarse**.

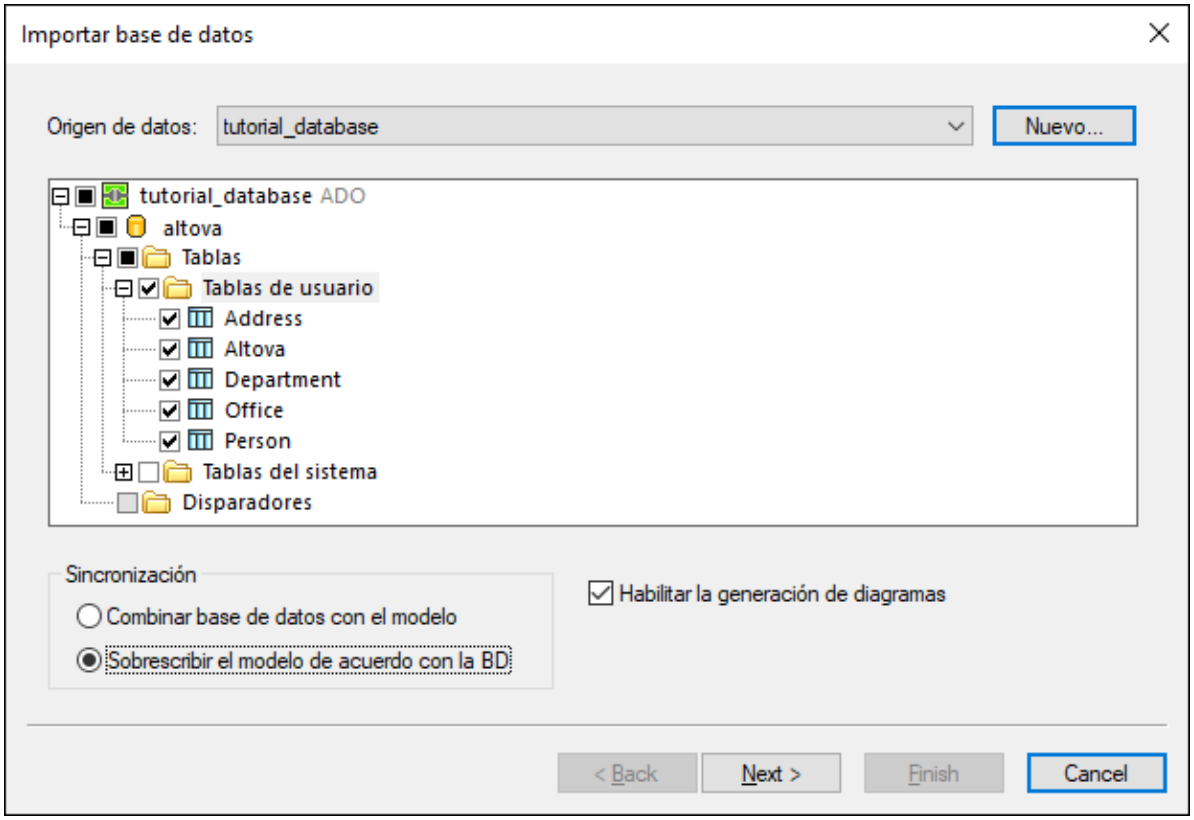


5. Introduzca un nombre descriptivo para su origen de datos. Es el nombre que deberá buscar después cuando quiera conectarse de nuevo a esta base de datos.

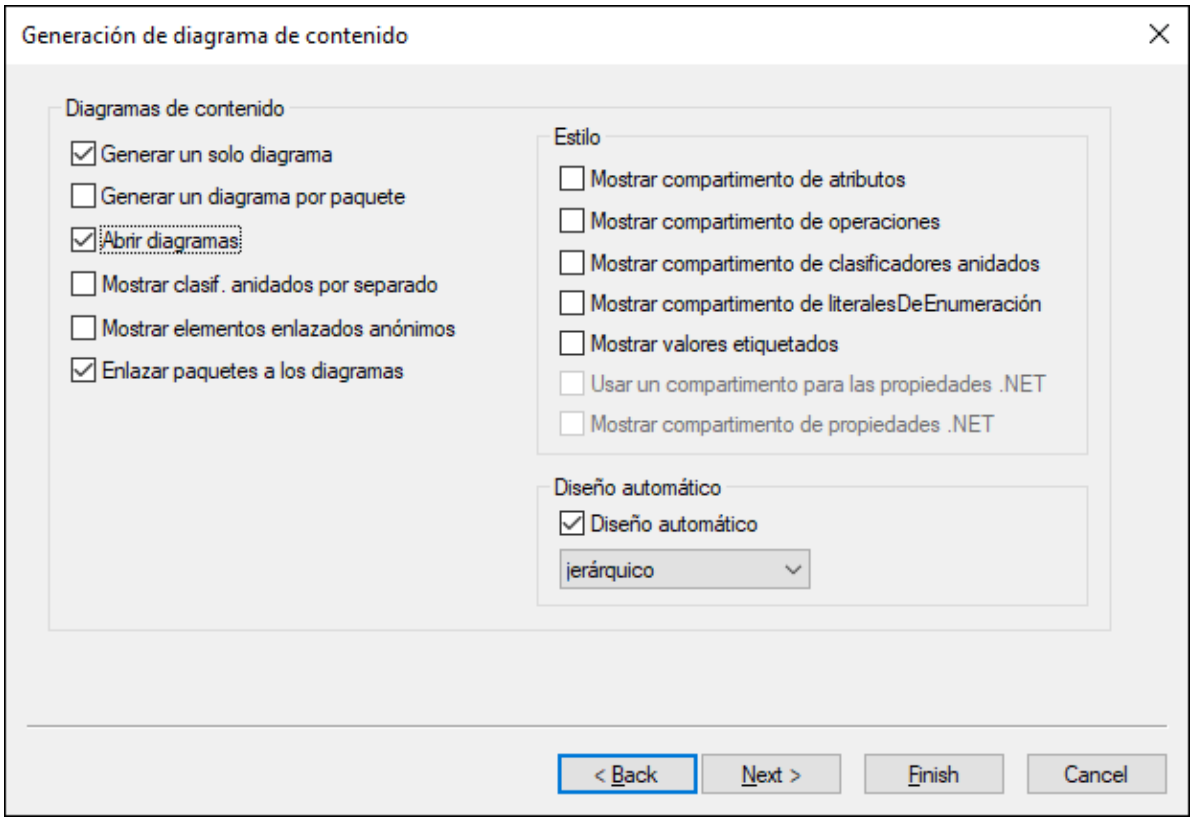


6. Seleccione los objetos de BD que quiera importar al modelo. En este ejemplo vamos a importar todas las tablas de usuario. Observe que la opción *Sobrescribir el modelo de acuerdo con la BD* viene

preseleccionado (es decir, que se reemplazarán todos los elementos del proyecto con los que se importen de la BD). Para proyectos que ya existen, cambie esta opción a *Combinar base de datos con el modelo*.

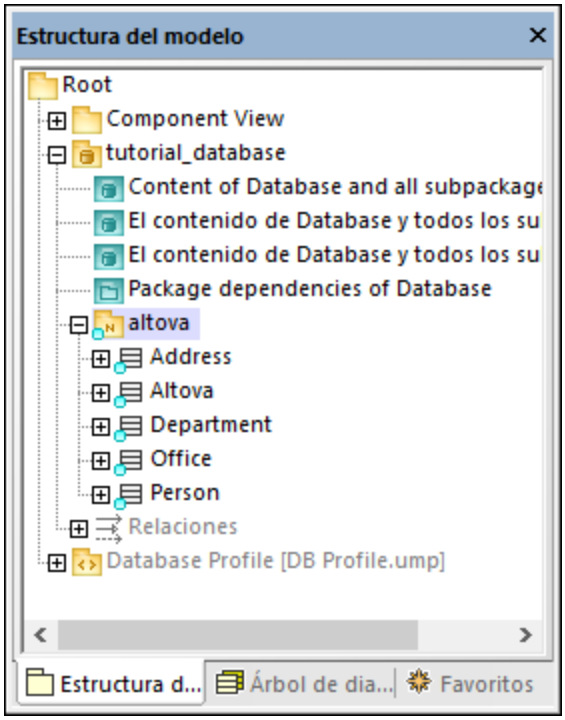


7. Haga clic en el botón **Siguiente**. Seleccione las opciones correspondientes de generación de diagrama:

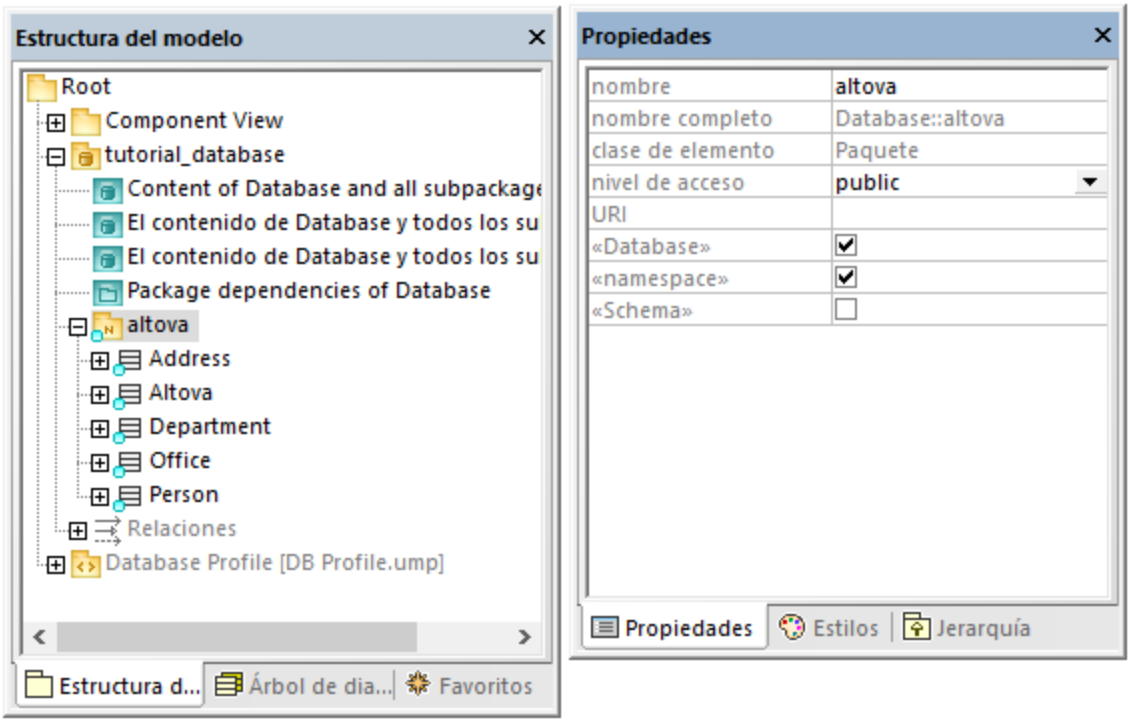


8. Haga clic en **Finalizar**.

Ahora el proyecto contiene todos los objetos importados desde la base de datos (las tablas y su estructura). También se crean dos diagramas, uno de BD que muestra los objetos de la BD y un diagrama de dependencias entre paquetes.



Como muestra la imagen anterior, el origen de datos (`tutorial_database` en este ejemplo) se ha convertido en un paquete del modelo. La BD (`altova`) también se ha convertido en un paquete con los estereotipos «`Database`» y «`namespace`». Para ver las propiedades de un paquete, haga clic en el paquete y consulte la ventana *Propiedades*.



Nota: cuando se importa una BD, UModel crea paquetes y aplica estereotipos en función del tipo de BD. El modelo anterior corresponde a las bases de datos de Access.

En el modelo, todas las tablas de BD se convierten en clases y reciben el estereotipo «Table». Observe también que, tras la importación, el perfil de la BD (**DB Profile.ump**) se ha añadido automáticamente al proyecto.

En este punto, el proyecto está configurado para la ingeniería de código de BD a modelo. Es decir, siempre que quiera actualizar un proyecto de UModel con los últimos cambios en la BD, ejecute el siguiente comando:

- En el menú **Proyecto**, haga clic en *Combinar base de datos con el modelo* o en *Sobrescribir el modelo de acuerdo con la BD*.

Si lo que quiere es sincronizar la BD con los datos del modelo, consulte Configurar la ingeniería de ida y vuelta para bases de datos.

10.1.2 Diseñar objetos de base de datos

En UModel puede crear, editar o eliminar objetos de base de datos (como tablas, columnas, claves foráneas, etc) desde un diagrama de BD o desde la ventana *Árbol de diagramas*.

Al visualizar o designar objetos de base de datos en UModel, tenga en cuenta estas reglas básicas:

- las tablas con clases con el estereotipo «Table»
- las columnas son propiedades de clases

- las claves principales, foráneas y únicas son clases con los estereotipos «PrimaryKey», «ForeignKey» y «UniqueKey» respectivamente.
- las restricciones de consultas son clases con el estereotipo «CheckConstraint».
- los índices son clases con el estereotipo «Index».

Para una tabla detallada de las correspondencias entre objetos de BD y elementos de UModel, consulte Correspondencias con elementos de BD.

Añadir tablas

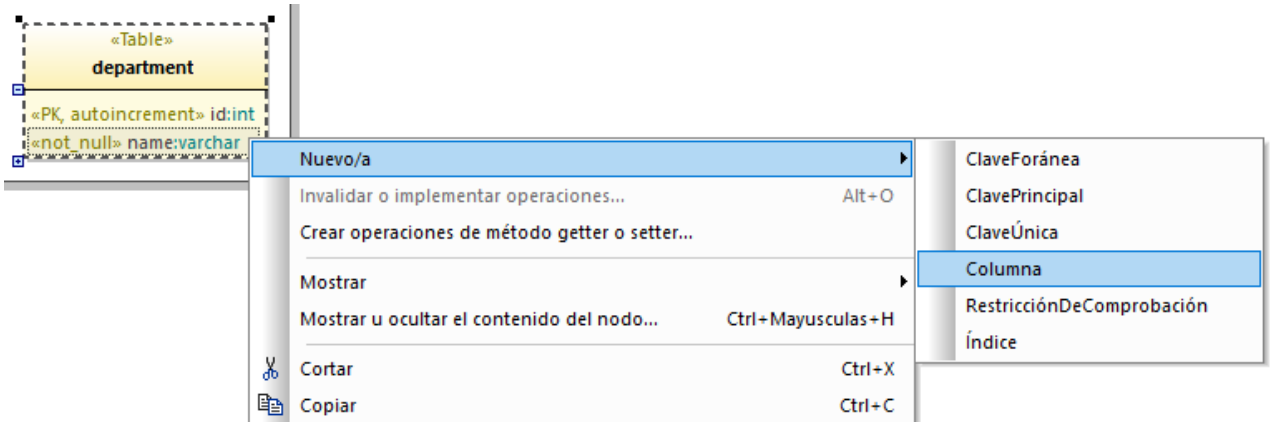
Para añadir una tabla al modelo tiene varias opciones:

1. Cree un diagrama de base de datos o abra uno que ya existe. Para crear un diagrama de BD nuevo, haga clic con el botón derecho en la ventana *Estructura del modelo* y seleccione **Diagrama nuevo | Diagrama de base de datos** en el menú contextual.
2. Ahora elija una de estas opciones:
   a. Haga clic con el botón derecho dentro del diagrama y seleccione **Nuevo/a | Tabla** en el menú contextual.
   b. Haga clic en el botón **Tabla** de la barra de herramientas y haga clic dentro del diagrama para añadir una tabla nueva.

Nota: puede añadir una clase de tabla en cualquier lugar del modelo. Sin embargo, es una buena práctica, especialmente si quiere usar ingeniería de código, que todas las clases de tablas pertenezcan a un mismo paquete que tenga el estereotipo «Database». Se crea automáticamente un paquete así cada vez que importa una BD que ya existe en el modelo (*véase Importing SQL Databases into UModel*).

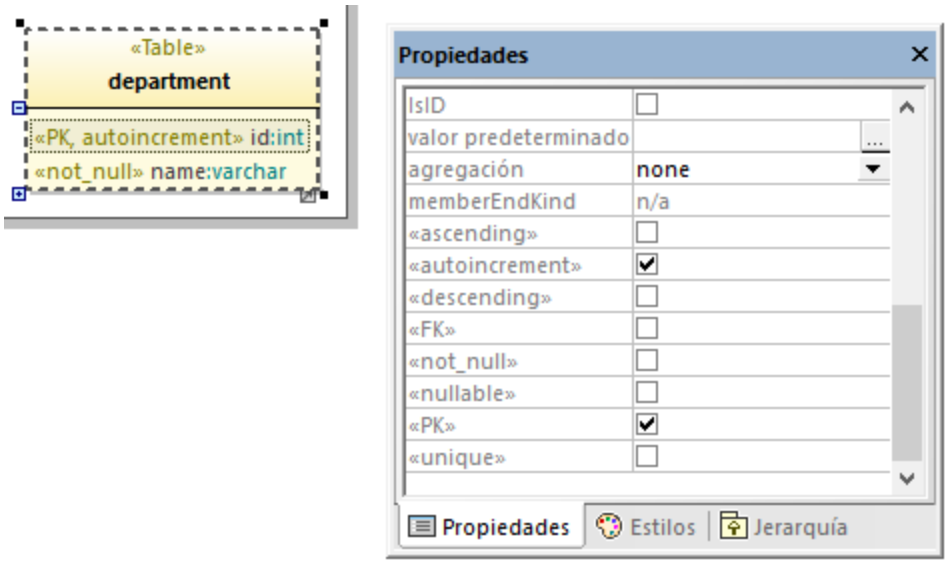
Añadir otros objetos de BD

Para añadir columnas, índices, claves foráneas, etc. a una tabla, haga clic con el botón derecho en la tabla de un diagrama y seleccione el comando correspondiente en el menú contextual, por ejemplo:



También puede hacer clic en un botón de la barra de herramientas del diagrama y después hacer clic dentro de la tabla de destino.

Para definir atributos de columna como "autoincrement", "nullable" o "clave principal", primero haga clic en la columna y luego marque la casilla correspondiente (estereotipo) en la ventana Propiedades:

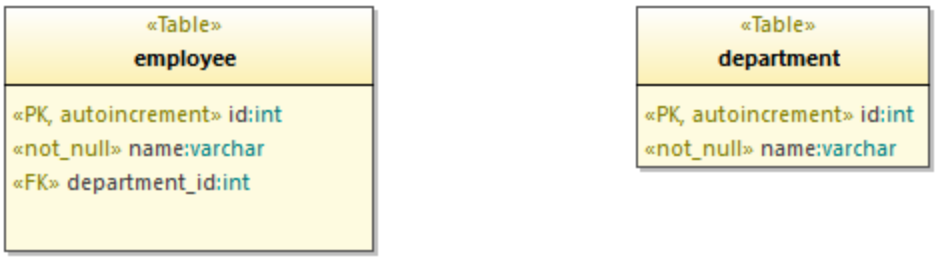


También puede crear la columna y definir todos los atributos necesarios conforme va tecleando. Por ejemplo, para crear una columna primaria con incremento automático, el nombre "id" y el tipo "int", haga lo siguiente:

1. Seleccione una tabla del diagrama y pulse la tecla F7.
2. Comience a teclear `<<PK, autoincrement>> id:int`. Conforme va tecleando, UModel le ayuda automáticamente a elegir los valores necesarios de una lista.

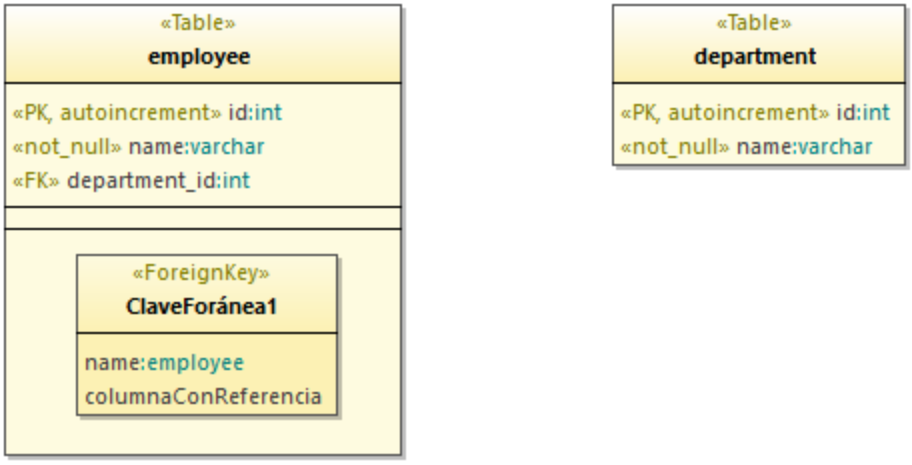
Añadir relaciones de BD

Se suelen añadir relaciones para mostrar las relaciones de dependencia de las claves foráneas entre columnas de distintas tablas. Por ejemplo, imaginemos que tenemos las siguientes clases:

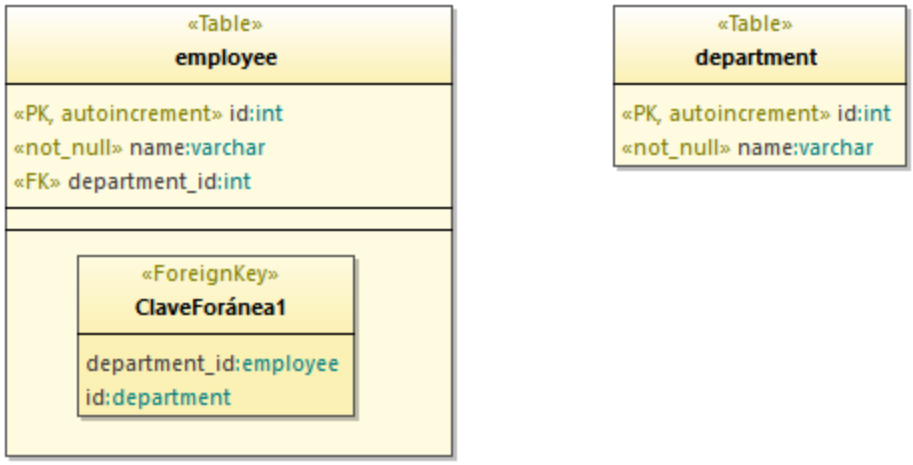


Para añadir una relación de clave foránea entre la columna **department_id** de la tabla "person" y la columna **id** de la tabla "department", siga estos pasos:

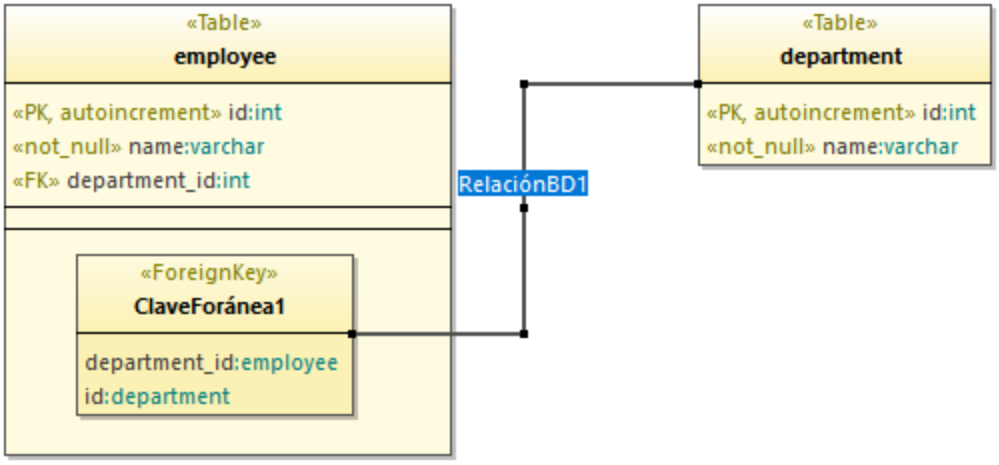
1. Haga clic con el botón derecho en la tabla "employee" y seleccione **Nuevo/a | ClaveForánea** en el menú contextual. Se añadirá una nueva clase llamada "ClaveForánea1" dentro de la clase "employee".



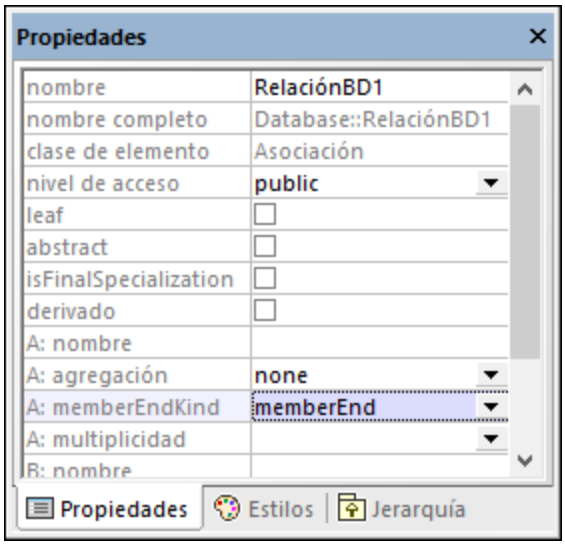
2. En la clase "ClaveForánea1", cambie la primera entrada de columna para que se corresponda con la columna y la tabla a las que pertenece (en este ejemplo, department_id:employee). Ahora cambie la segunda entrada de columna para que se corresponda con la columna y la tabla a las que hace referencia (en este ejemplo, id:department).



3. Haga clic en el botón de la barra de herramientas **Asociación de relación entre bases de datos** y arrastre con el ratón desde "ClaveForánea1" hasta la clase "department".



4. Seleccione la línea de relación y, en la ventana *Propiedades*, cambie la propiedad **A :memberEndKind** a **memberEnd**.



5. Pulse la tecla **F11** para comprobar si la sintaxis del proyecto contiene algún error (*véase el punto siguiente para más información*).

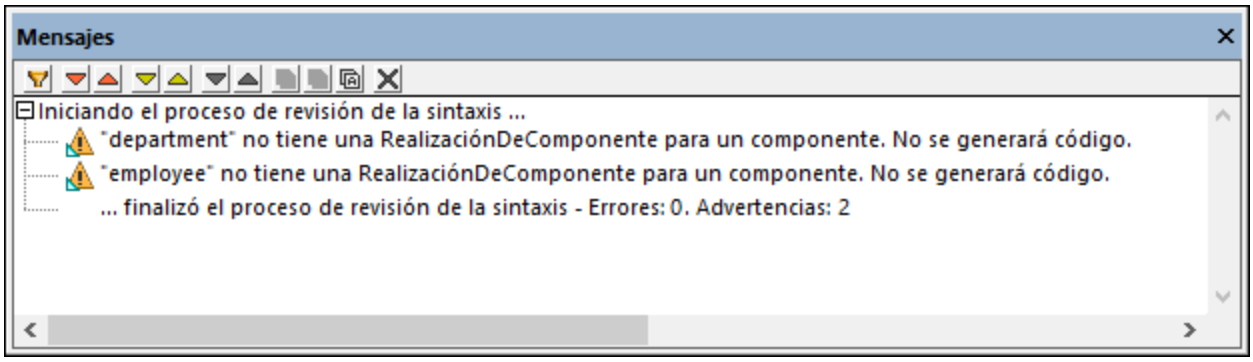
Nota: puede añadir varias entradas de columna por clase "ClaveForánea" si lo necesita. También puede añadir varios índices para la misma tabla.

Revisar la sintaxis del proyecto

Al crear o cambiar objetos de BD en UModel, es recomendable comprobar con regularidad si la sintaxis de nuestro proyecto contiene algún problema potencial de diseño (por ejemplo, tablas que no contengan al menos una columna, referencias claves foráneas que falten, etc.). Existen dos formas de revisar la sintaxis del proyecto:

- en el menú **Proyecto**, haga clic en **Revisar la sintaxis del proyecto**.
- pulse la tecla **F11**.

UModel valida el proyecto y muestra cualquier posible problema en la ventana *Mensajes*, como en la imagen siguiente:



Las dos advertencias de la imagen anterior indican que no se generará ningún código para las tablas "departament" y "employee". Puede ignorar estas advertencias si no necesita ingeniería de código en su proyecto de UModel. En caso contrario, consulte Configurar la ingeniería de ida y vuelta para bases de datos.

10.1.3 Configurar la ingeniería de ida y vuelta para bases de datos

Siempre que importe una base de datos en UModel, como se muestra en el apartado Importar bases de datos SQL en UModel, su proyecto queda vinculado a esa BD y usted puede sincronizar elementos desde la BD al modelo y viceversa.

Si quiere sincronizar solo desde la BD al modelo, no necesita configurar nada, ya que UModel se encarga de realizar en segundo plano las asignaciones necesarias. Por ejemplo, con cada sincronización las nuevas tablas se convierten en nuevas clases en el modelo, las definiciones de columnas de BD modificadas se actualizan en el modelo, etc. Todos los diagramas de su base de datos también se actualizarán para reflejar esos cambios.

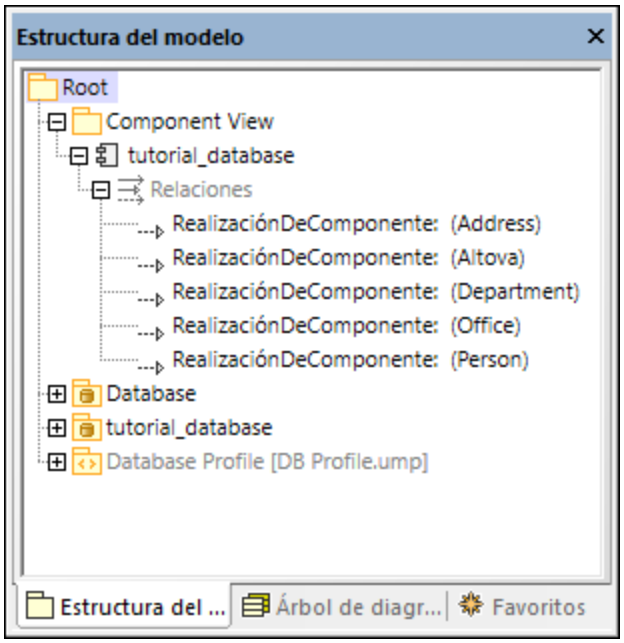
Sin embargo, si realiza cambios en el modelo y quiere sincronizarlos con la BD, entonces sí deberá aplicar la configuración correspondiente al proyecto de UModel. Es posible que también necesite aplicar esta configuración si quiere evitar que el proyecto (o unas tablas en concreto) se sincronicen con la base de datos.

Cada vez que se lleva a cabo una sincronización, los datos pueden combinarse o sobrescribir a los que ya existen; puede configurar una opción u otra con el comando de menú **Proyecto | Configurar sincronización**.

Nota: algunos tipos de BD no permiten cambiar la estructura de la base de datos debido a su diseño. Por ejemplo, las bases de datos de Microsoft Access no permiten que se modifiquen los nombres de las tablas o las columnas. Asimismo, SQLite no permite renombrar columnas. Por lo tanto, si realiza dichos cambios en el modelo, eso no conllevará una actualización de la BD y UModel probablemente mostrará advertencias en la ventana *Mensajes*.

La ingeniería de ida y vuelta para bases de datos es similar a la que se aplica al código de programación y gira en torno a un componente del paquete "Component View" que vincula el proyecto con la BD real. En concreto, siempre que importe una BD por primera vez se genera automáticamente un componente de ingeniería de

código bajo el paquete "Component View". Por ejemplo, si siguió todos los pasos del apartado Importar bases de datos SQL en UModel, entonces se generó un componente llamado **tutorial_database**:



Como hemos explicado antes, cada clase del modelo se corresponde con una tabla de BD. Para que sea posible la ingeniería de código, el componente de ingeniería de código debe comprender todas las clases (tablas) del modelo (observe todas las **realizacionesDeComponente** de la imagen anterior). Las clases que no estén comprendidas en este componente no formarán parte de la ingeniería de código. Si no va a querer actualizar nunca la BD desde el modelo no necesita hacer nada. UModel creará de forma automática todas las asociaciones siempre que sincronice desde la BD al modelo.

Sin embargo, si sí va a querer sincronizar desde el modelo a la BD, cada nueva clase (tabla) que añada debe tener una relación **realizaciónDeComponente** con el componente de la ingeniería de código. De lo contrario, cuando intente actualizar la BD desde el modelo, UModel muestra una advertencia parecida a esta: `Tabla1 no tiene una RealizaciónDeComponente para un componente. No se generará código.`.

La forma más sencilla de crear una **realizaciónDeComponente** con un componente a partir de una clase es arrastrar la clase hasta el componente de ingeniería de código. Así, por ejemplo, si ha creado una nueva clase (tabla), arrastre la clase (en la ventana *Estructura del modelo*) hasta el componente **tutorial_database** para crear la relación. También puede añadir o eliminar esas relaciones desde un diagrama de componentes (*véase el apartado Diagramas de componentes*).

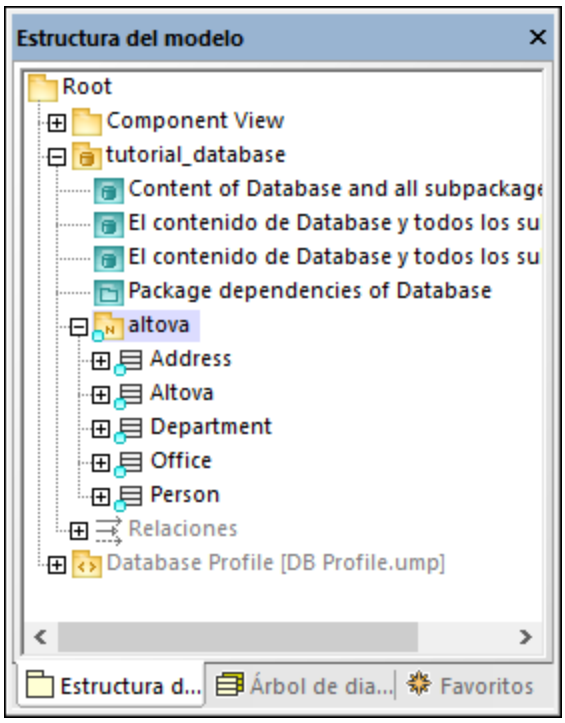
Para ver un ejemplo completo, consulte el apartado Ejemplo: actualizar una BD desde el modelo.

10.1.4 Ejemplo: actualizar una BD desde el modelo

Este ejemplo muestra cómo actualizar la estructura de una BD mediante scripts generados por UModel. La BD usada en este ejemplo es una BD local de Access que encontrará en **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Tutorial\altova.mdb**. En este ejemplo vamos a añadir una nueva tabla a la BD en UModel y después generar un script SQL que actualice la estructura de la BD Access subyacente.

Para este ejemplo, primero debe importar la BD en el modelo, como se muestra en el apartado Importar bases de datos SQL en UModel. Como se ilustra a continuación, una vez haya importado la BD, el proyecto incluirá:

- un componente de ingeniería de código que se encarga de generar código en ambas direcciones (desde el modelo a la BD y viceversa). Para ver este componente de ingeniería de código, expanda el "Component View".
- un paquete que representa la estructura de la base de datos que se ha importado (por ejemplo, cada tabla de BD es una clase).
- el perfil de BD necesario para trabajar con proyectos de modelado de bases de datos.



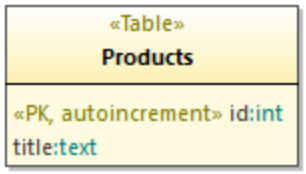
Añadir una tabla

Ahora vamos a añadir una nueva tabla a la BD del modelo.

1. Haga doble clic en el diagrama "Content of tutorial_database...".
2. Haga clic con el botón derecho dentro del diagrama y seleccione **Nuevo/a | Tabla** en el menú contextual.
3. Introduzca un nombre, como "Products".

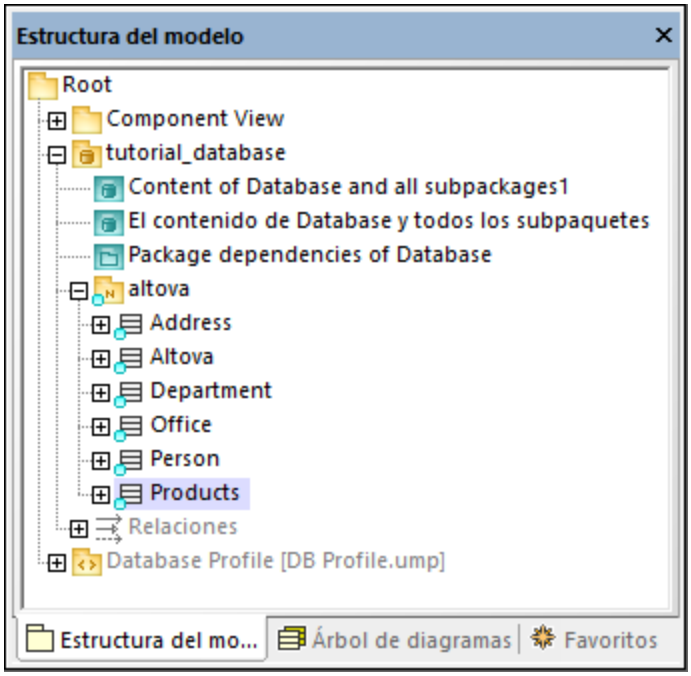
4. Haga clic en la tabla y pulse la tecla **F7** para añadir una propiedad (que se convertirá en una columna de tabla en la BD).
5. Telcee `<<PK, autoincrement>> id:int` dentro de la propiedad.

6. Usando los mismos pasos que acabamos de describir, añada una nueva columna "title" de tipo "text".



Preparar el modelo para la ingeniería directa

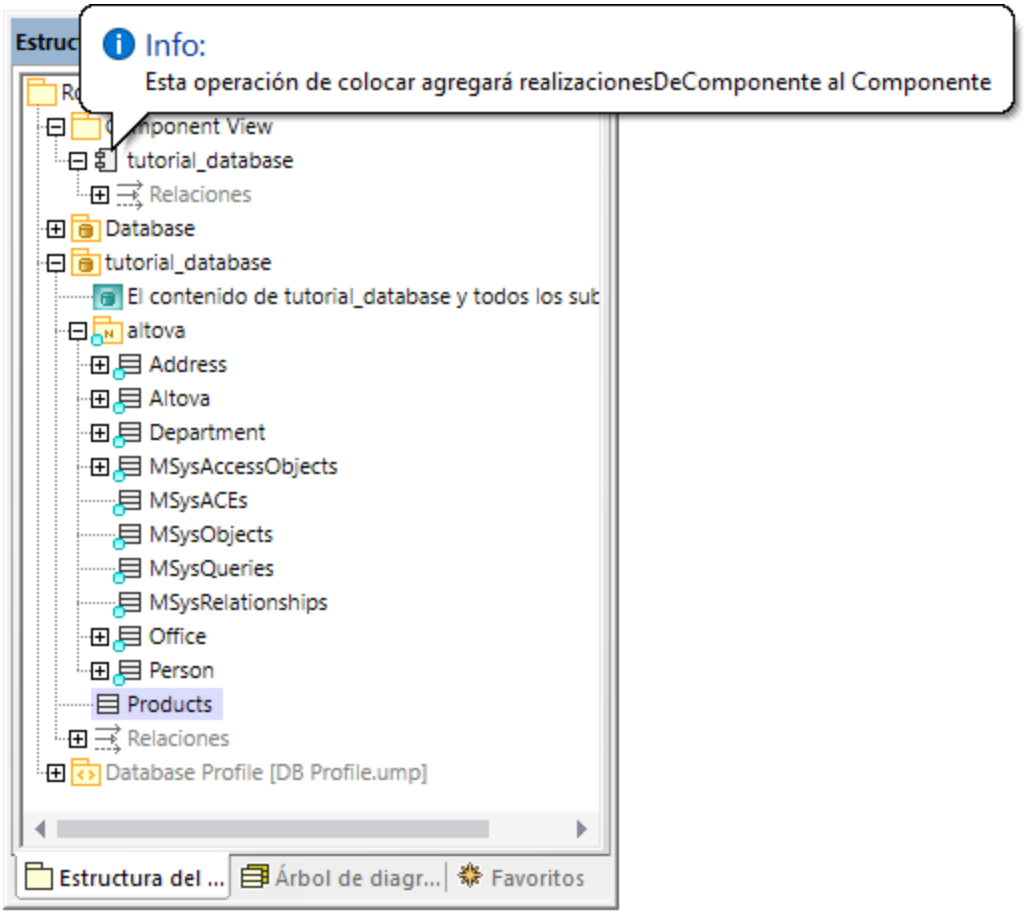
Antes de que se pueda aplicar la ingeniería directa a una BD, esta debe pertenecer al espacio de nombres correcto. Para ello, en la ventana *Estructura del modelo*, asegúrese de que la clase "Products" se encuentra bajo el espacio de nombres "tutorial_database". Si no lo está, arrástrela hasta allí. Su modelo ahora debería tener este aspecto:



Como hemos explicado en el apartado Configurar la ingeniería de ida y vuelta para bases de datos, es recomendable validar la sintaxis del proyecto antes de actualizar la BD. Si pulsa la tecla **F11** para revisar la sintaxis aparecerá una advertencia en la ventana *Mensajes* informando de que la tabla "Products" no tiene ninguna realización con un componente.

Para crear rápidamente esa realización que falta, haga lo siguiente:

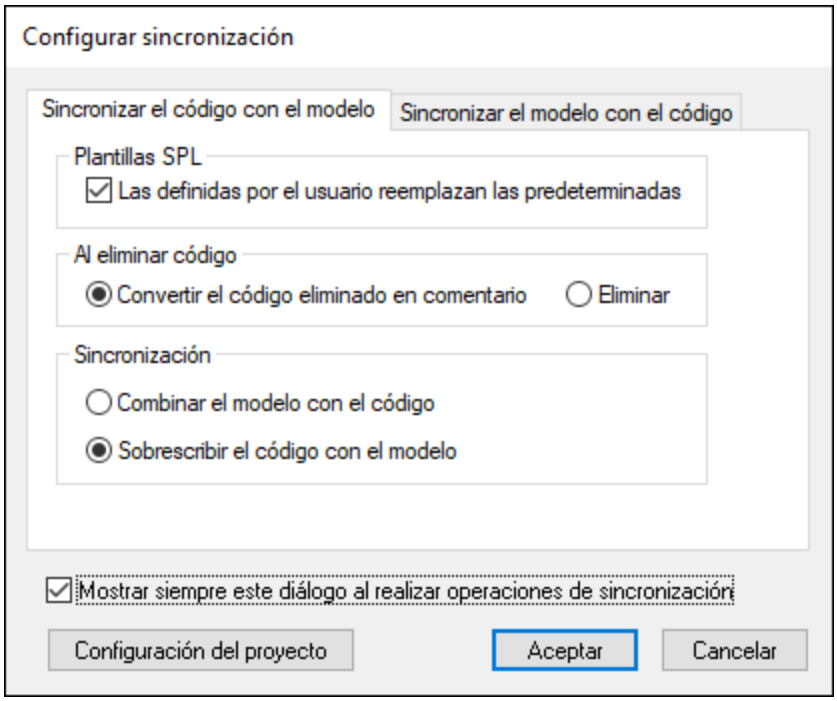
- En la ventana *Estructura del modelo*, arrastre la clase "Products" hasta el componente "tutorial_database".



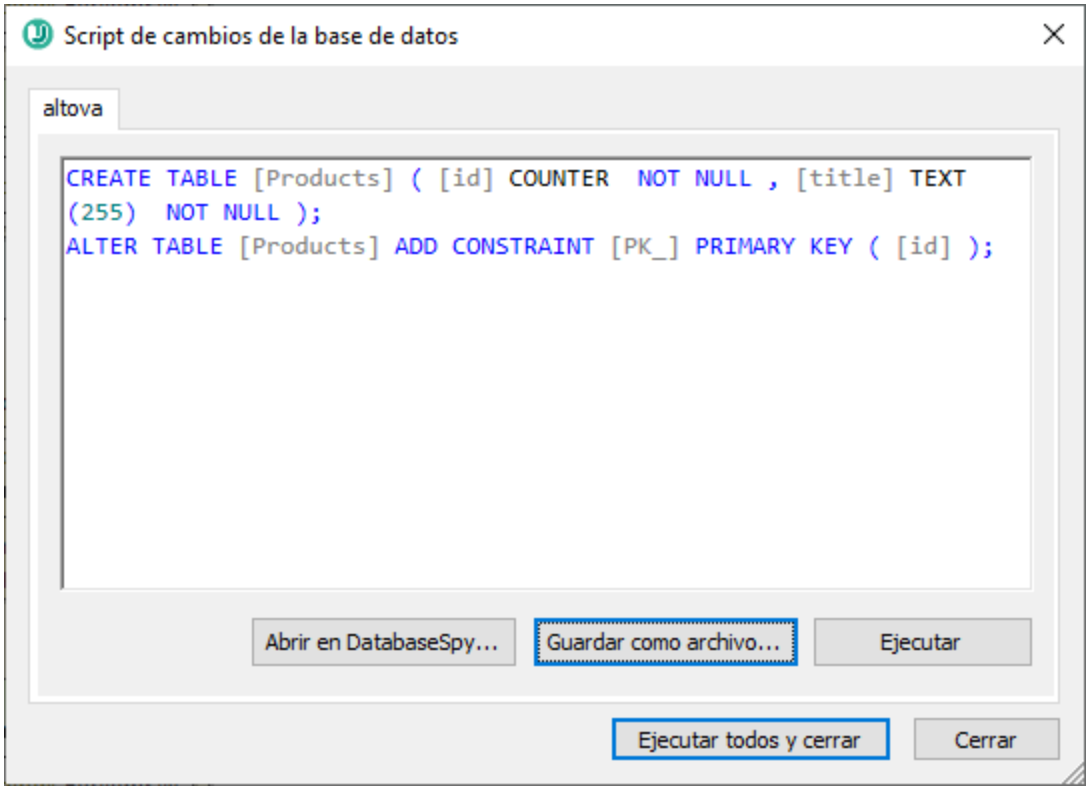
Generar el script SQL

Si al pulsar la tecla **F11** el proyecto no muestra más errores o advertencias, puede generar el script de BD:

1. En el menú **Proyecto**, haga clic en **Combinar el código de programa con el proyecto de UModel** y luego en **Sobrescribir el código con el modelo**. ("Código de programa" en el contexto de las bases de datos significa la propia BD)
2. En el siguiente cuadro de diálogo puede escoger entre combinar los cambios con la BD o sobrescribir esos cambios en la BD. Para este ejemplo vamos a marcar la opción **Sobrescribir el código con el modelo**. Por supuesto, en función del caso puede que quiera escoger la opción **Combinar el modelo con el código**. Para más información, consulte el apartado Configurar la sincronización del código.



3. Haga clic en **Aceptar**. Se genera un script de BD con los cambios que ha realizado en el modelo.



En este punto tiene varias opciones:

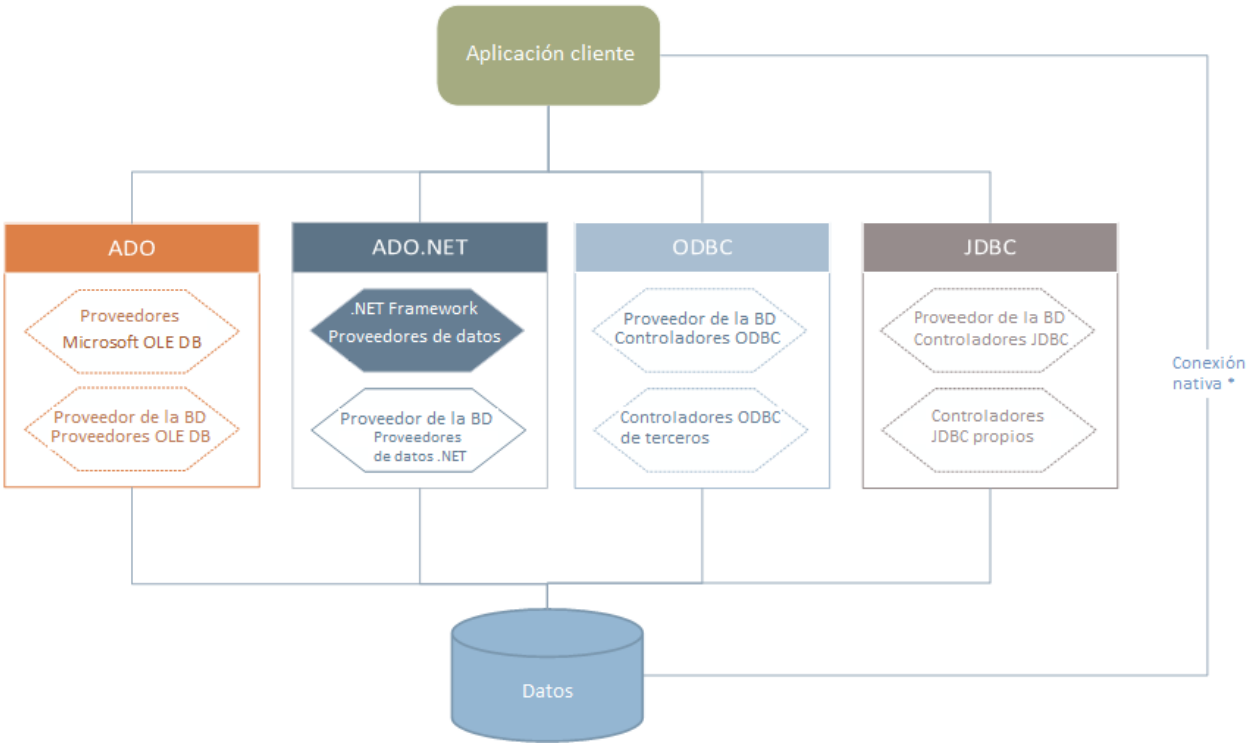
- abra el script en Altova DatabaseSpy para revisarlo o para ejecutarlo. Para más información sobre DatabaseSpy, consulte https://www.altova.com/es/databasespy.
- guarde el script en un archivo para conservarlo o ejecutarlo más tarde.
- haga clic en **Ejecutar** y ejecute el script en la BD. Solo debe ejecutar un script si entiende totalmente las consecuencias de esa acción (principalmente, el hecho de que la BD se actualizará con efecto inmediato).

10.2 Conectarse a un origen de datos

En su definición más sencilla, una base de datos es un archivo local como un archivo de base de datos Microsoft Access o SQLite. En casos más complejos, una base de datos puede residir en un servidor de bases de datos remoto o de la red que no tienen por qué usar el mismo sistema operativo que la aplicación que se conecta a la BD y que consume los datos. Por ejemplo, mientras que UModel se puede ejecutar en sistemas operativos Windows, puede que la base de datos a la que desea acceder (p. ej. una base de datos MySQL) esté en un equipo Linux.

Para interactuar con los diferentes tipos de bases de datos, UModel se sirve de las interfaces de conexión de datos y los controladores de BD disponibles en su sistema operativo o publicados por los principales proveedores de BD. La tecnología de base de datos evoluciona constantemente y por tanto consideramos que este mecanismo garantiza la mayor compatibilidad y flexibilidad en las principales plataformas.

En el siguiente diagrama puede ver un resumen de las opciones de conectividad de datos que pueden existir entre UModel (*aplicación cliente*) y un almacén de datos (que puede ser un archivo o un servidor de base de datos).



** Las bases de datos SQLite, PostgreSQL, CouchDB y MongoDB admiten conexiones nativas directas. Para conectarse a este tipo de bases de datos no es necesario tener instalado controladores específicos.*

Por tanto, tal y como muestra el diagrama, desde UModel puede acceder a los principales tipos de BD con estas tecnologías de acceso de datos:

- ADO (Microsoft® ActiveX® Data Objects), que a su vez utiliza OLE DB
- ADO.NET (un conjunto de bibliotecas disponibles en Microsoft .NET Framework que permiten la interacción con datos)

- JDBC (Java Database Connectivity)
- ODBC (Open Database Connectivity)

Nota: algunos proveedores de ADO.NET no son compatibles o tienen una compatibilidad limitada. Consulte el apartado *Notas sobre compatibilidad con ADO.NET* para obtener más información.

Tecnologías de acceso a datos

La interfaz de conexión de datos que se debe utilizar dependerá de la infraestructura de software con la que cuente. Lo normal es elegir una tecnología de acceso a datos y un controlador de base de datos que se integre bien con el sistema de base de datos al que desea conectarse. Por ejemplo, para conectarse con una base de datos Microsoft Access 2013, puede generar una cadena de conexión ADO que utilice un proveedor nativo como **Microsoft Office Access Database Engine OLE DB Provider**. Para conectarse a Oracle, por su parte, lo mejor será descargar e instalar las interfaces JDBC, ODBC o ADO .NET más recientes del sitio web de Oracle.

Lo más probable es que los controladores de los productos Windows (como Microsoft Access o SQL Server) ya estén disponibles en su sistema, pero puede que necesite descargar controladores para otros tipos de bases de datos. Los principales proveedores de BD publican software cliente y controladores con frecuencia. Además, puede encontrar otros controladores de otras organizaciones para las tecnologías de acceso de datos mencionadas. En la mayoría de los casos hay varias maneras de conectarse a una base de datos. Las características y el rendimiento de la base de datos dependerán de la tecnología de acceso o de los controladores utilizados

10.2.1 Iniciar el asistente para la conexión de base de datos

UModel cuenta con un asistente que le guiará durante la conexión a un origen de datos. Antes de empezar a seguir las instrucciones del asistente, debe tener en cuenta que algunos tipos de BD requieren instalar y configurar algunos elementos, como controladores o software cliente. Normalmente este tipo de productos se obtienen del proveedor de BD correspondiente, que los acompañan de la documentación pertinente específica para su versión de Windows. Para ver una lista de controladores de base de datos agrupados por tipo de BD consulte el apartado Resumen de controladores de base de datos.

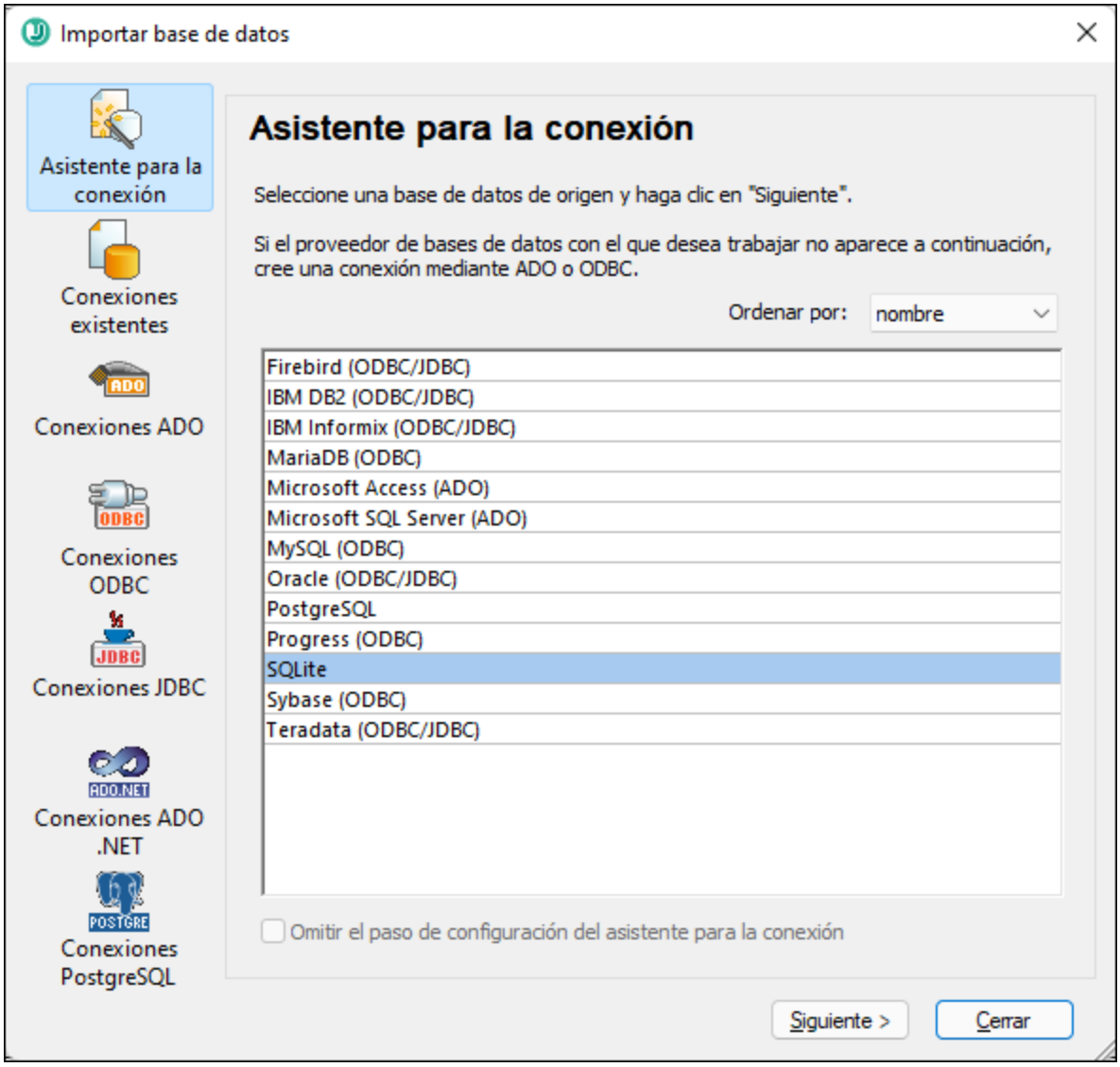
Para iniciar el asistente para la conexión de BD (*imagen siguiente*) siga estos pasos:

1. En el menú **Proyecto** haga clic en **Importar base de datos SQL**.
2. Después haga clic en el botón **Nuevo**.

Con estos pasos ha iniciado correctamente el asistente para la conexión a bases de datos (*imagen siguiente*). En la parte izquierda de la ventana puede seleccionar la forma de conectarse a la BD que prefiera:

- Con el asistente para la conexión, que le pide que elija un tipo de base de datos y después le indica qué pasos seguir para conectarse a ese tipo de BD
- Con una conexión que ya existe
- Con uno de estos tipos de tecnología de acceso de datos: ADO, ADO.NET, ODBC o JDBC
- Con una conexión PostgreSQL nativa

En el panel del asistente para la conexión (*imagen siguiente*) las bases de datos se pueden ordenar alfabéticamente por tipo de BD o por uso reciente. Seleccione la opción que prefiera en el cuadro combinado *Ordenar*. Una vez haya seleccionado el tipo de BD haga clic en **Siguiente**.



Siga las instrucciones que aparecen en pantalla, que dependerán del tipo de BD, de la tecnología de acceso (ADO, ADO.NET, ODBC, JDBC) y del controlador utilizados. Para ver ejemplos de conexión de cada tipo de BD consulte el apartado Ejemplos de conexión a bases de datos.

También puede consultar estos apartados:

- Conexiones ADO
- Conexiones ADO.NET
- Conexiones ODBC
- Conexiones JDBC

10.2.2 Resumen de controladores de base de datos

En la tabla que aparece más abajo puede ver una lista de controladores de BD que puede utilizar para conectarse a una base de datos a través de las diferentes tecnologías de acceso de datos. Esta lista no contiene todos los controladores disponibles en la actualidad y, por tanto, puede usar otros controladores que no aparecen en la lista.

Aunque puede que Windows venga con algunos controladores de BD, es posible que necesite descargar más controladores. Por lo general, se recomienda utilizar el controlador más reciente publicado por el proveedor de la BD.

Los proveedores de bases de datos pueden ofrecer controladores para descargar en paquetes o junto con el software cliente de base de datos. En este último caso, el cliente de base de datos suele incluir todos los controladores necesarios u ofrecer una opción durante la instalación para seleccionar los controladores y componentes que el usuario desee instalar. El software cliente de base de datos suele incluir funciones de administración y configuración que permiten simplificar la administración de la BD y documentación que explica cómo instalar y configurar el cliente y sus componentes.

Es muy importante configurar bien el cliente de BD para establecer correctamente la conexión con la BD. Antes de instalar y usar el software cliente de BD, recomendamos que lea detenidamente las instrucciones de instalación y configuración porque pueden variar según la versión de la BD y según la versión de Windows.

Si desea conocer las características y limitaciones de cada tecnología de acceso de datos con respecto a cada tipo de BD, consulte la documentación de la BD correspondiente y pruebe la conexión. Debe tener en cuenta estos puntos para evitar problemas de conexión:

- Algunos proveedores de ADO.NET no son compatibles o tienen una compatibilidad limitada (*véase* *Notas sobre compatibilidad con ADO.NET*).
- Cuando instale un controlador de BD, recomendamos que tenga la misma plataforma que la aplicación de Altova (32 o 64 bits). Por ejemplo, si usa una aplicación de Altova de 32 bits en un sistema operativo de 64 bits, instale el controlador de 32 bits y configure la conexión de base de datos con ayuda del controlador de 32 bits (*véase* *Ver los controladores ODBC disponibles*).
- Cuando configure un origen de datos ODBC, recomendamos crear el nombre de origen de datos (DSN) como DSN del sistema en lugar de como DSN de usuario (*véase* *Conexiones ODBC*).
- Cuando configure un origen de datos JDBC, compruebe que tiene instalado JRE (Java Runtime Environment) o Java Development Kit (JDK) y que la variable de entorno CLASSPATH del sistema operativo está configurado (*véase* *Conexiones JDBC*).
- Si necesita consultar las instrucciones de instalación e información sobre compatibilidad de los controladores o clientes de BD, consulte la documentación que viene con el paquete de instalación.

Base de datos	**Interfaz**	Controladores
Firebird	ADO.NET	Proveedor de datos ADO.NET (https://www.firebirdsql.org/en/additional-downloads/)
	JDBC	Controlador Firebird JDBC (https://www.firebirdsql.org/en/jdbc-driver/)
	ODBC	Controlador Firebird ODBC (https://www.firebirdsql.org/en/odbc-driver/)
IBM DB2	ADO	Proveedor IBM OLE DB para DB2

Base de datos	Interfaz	Controladores
	ADO.NET	Proveedor IBM Data Server para .NET
	JDBC	Controlador IBM Data Server para JDBC y SQLJ
	ODBC	Controlador IBM DB2 ODBC
IBM DB2 for i	ADO	• Proveedor IBM DB2 para i5/OS IBMDA400 OLE DB • Proveedor IBM DB2 para i5/OS IBMDARLA OLE DB • Proveedor IBM DB2 para i5/OS IBMDASQL OLE DB
	ADO.NET	Proveedor de datos .NET Framework Data Provider para IBM i
	JDBC	Controlador IBM Toolbox para Java JDBC
	ODBC	Controlador iSeries Access ODBC
IBM Informix	ADO	Controlador IBM Informix OLE DB
	JDBC	Controlador IBM Informix JDBC
	ODBC	Controlador IBM Informix ODBC
Microsoft Access	ADO	• Proveedor de Microsoft Jet OLE DB • Proveedor de Microsoft Access Database Engine OLE DB
	ADO.NET	Proveedor de datos de .NET Framework para OLE DB
	ODBC	• Controlador de Microsoft Access
MariaDB	ADO.NET	Si falta el conector .NET especial para MariaDB, use **Connector/NET** pararMySQL (https://dev.mysql.com/downloads/connector/net/).
	JDBC	MariaDB Connector/JDBC (https://downloads.mariadb.org/)
	ODBC	MariaDB Connector/ODBC (https://downloads.mariadb.org/)
Microsoft SQL Server	ADO	• Controlador Microsoft OLE DB para SQL Server (MSOLEDBSQL) • Proveedor de Microsoft OLE DB para SQL Server (SQLOLEDB) • Cliente nativo de SQL Server (SQLNCLI)
	ADO.NET	• Proveedor de datos .NET Framework para SQL Server • Proveedor de datos .NET Framework para OLE DB
	JDBC	• Controlador JDBC de Microsoft para SQL Server (https://msdn.microsoft.com/library/mt484311.aspx)
	ODBC	• Cliente nativo de SQL Server
MySQL	ADO.NET	Conector/NET (https://dev.mysql.com/downloads/connector/net/)
	JDBC	Conector/JDBC (https://dev.mysql.com/downloads/connector/j/)
	ODBC	Conector/ODBC (https://dev.mysql.com/downloads/connector/odbc/)

Base de datos	Interfaz	Controladores
Oracle	ADO	• Proveedor de datos Oracle para OLE DB • Proveedor de Microsoft OLE DB para Oracle
	ADO.NET	Proveedor de datos Oracle para .NET (https://www.oracle.com/technetwork/topics/dotnet/index-085163.html)
	JDBC	• Controlador JDBC Thin • Controlador JDBC Oracle Call Interface (OCI) Estos controladores se suelen instalar durante la instalación del cliente de base de datos Oracle. Conéctese con el controlador OCI (no con Thin) si usa el componente Oracle XML DB.
	ODBC	• Microsoft ODBC para Oracle • Controlador Oracle ODBC (por lo general se instala durante la instalación del cliente de base de datos Oracle)
PostgreSQL	JDBC	Controlador PostgreSQL JDBC (https://jdbc.postgresql.org/download.html)
	ODBC	psqlODBC (https://odbc.postgresql.org/)
	Conexión nativa	Disponible. No necesita instalar ningún controladores si usa una conexión nativa.
Progress OpenEdge	JDBC	Conector JDBC (https://www.progress.com/jdbc/openedge)
	ODBC	Conector ODBC (https://www.progress.com/odbc/openedge)
SQLite	Conexión nativa	Disponible. No necesita instalar ningún controladores si usa una conexión nativa.
Sybase	ADO	Proveedor de Sybase ASE OLE DB
	JDBC	jConnect™ para JDBC
	ODBC	Controlador Sybase ASE ODBC
Teradata	ADO.NET	Proveedor de datos .NET para Teradata (https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata)
	JDBC	Controlador JDBC de Teradata (https://downloads.teradata.com/download/connectivity/jdbc-driver)
	ODBC	Controlador ODBC de Teradata para Windows (https://downloads.teradata.com/download/connectivity/odbc-driver/windows)

10.2.3 Conexiones ADO

Microsoft ActiveX Data Objects (ADO) es una tecnología de acceso de datos que permite conectarse a gran variedad de orígenes de datos con OLE DB. OLE DB es una interfaz alternativa a ODBC y JDBC. Ofrece acceso uniforme a los datos en un entorno COM (Component Object Model). ADO es el precursor del nuevo ADO.NET y suele utilizarse para conectarse a bases de datos Microsoft nativas como Microsoft Access o SQL Server, aunque también puede usar ADO para otros orígenes de datos.

Es importante saber que puede escoger entre varios proveedores ADO y que en el caso de algunos es necesario que los descargue e instale en su equipo antes de poder usarlos. Por ejemplo, para conectarse a SQL Server puede usar estos proveedores ADO:

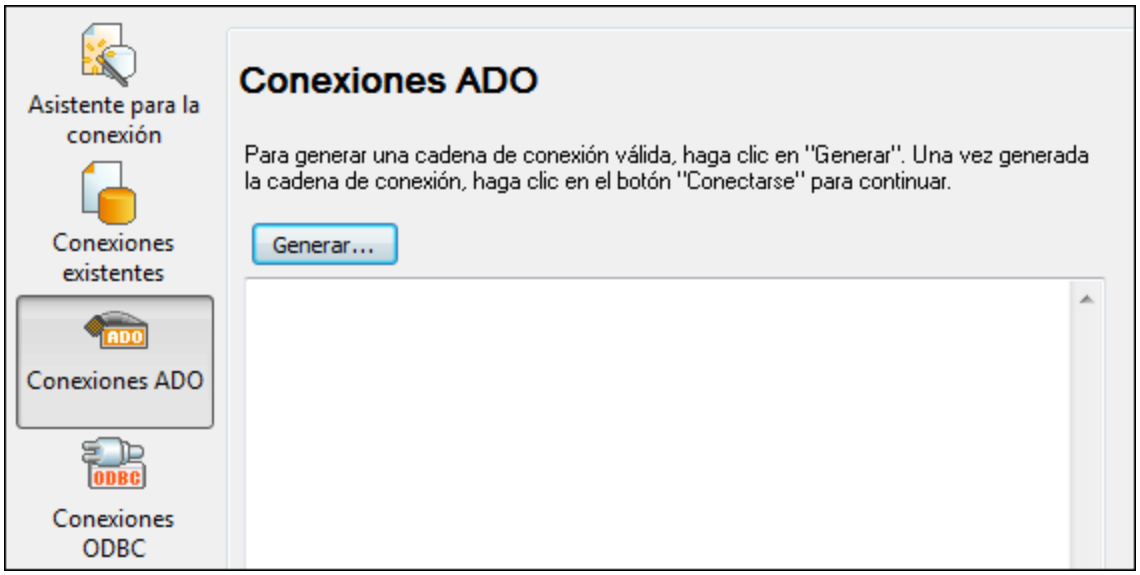
- *Controlador* de Microsoft OLE DB para SQL Server (MSOLEDBSQL)
- *Proveedor* de Microsoft OLE DB para SQL Server (SQLOLEDB)
- Cliente nativo de SQL Server (SQLNCLI)

De entre estos tres proveedores recomendamos que use MSOLEDBSQL, que puede descargar desde https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15. Tenga en cuenta que la versión que descargue debe ser coincidir con la de la plataforma de UModel (32 bits o 64 bits). Los proveedores SQLOLEDB y SQLNCLI se consideran obsoletos, por lo que no se recomiendan.

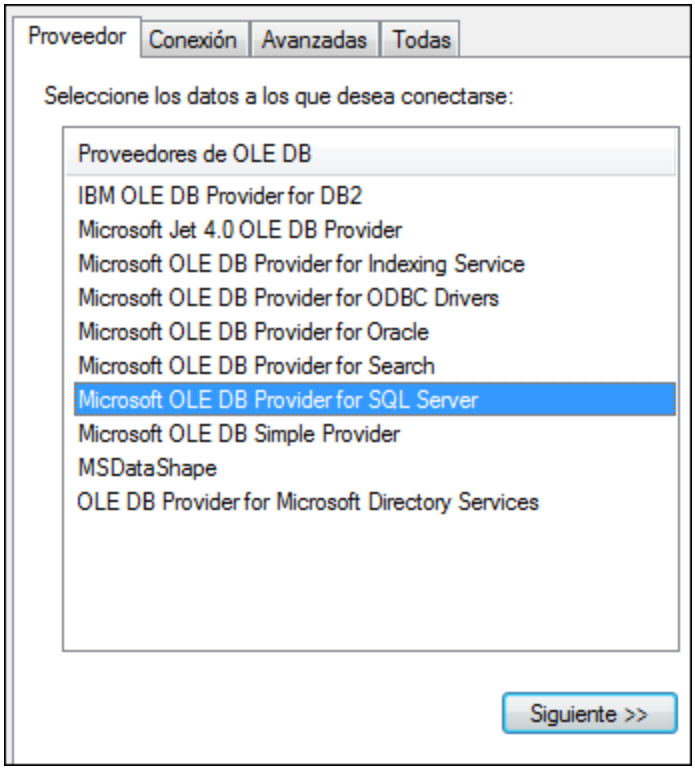
> Es un problema conocido que el proveedor de BD para SQL Server **Microsoft OLE DB Provider for SQL Server (SQLOLEDB)** tiene dificultades para enlazar parámetros de consultas complejas como las expresiones comunes de tabla (CTE) e instrucciones SELECT anidadas.

Para configurar una conexión ADO:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones ADO**.



3. Haga clic en **Generar**.



4. Seleccione el proveedor de datos que desea utilizar para establecer la conexión. La tabla que aparece a continuación enumera las combinaciones más comunes.

Para conectarse a esta BD...	Utilice este proveedor...
Microsoft Access	• **Proveedor de Microsoft Office Access Database Engine OLE DB** (recomendado) • **Proveedor de Microsoft Jet OLE DB** Si el proveedor **Microsoft Office Access Database Engine OLE DB Provider** no está disponible en la lista, asegúrese de que tiene instalados o bien Microsoft Access o el componente redistribuible del motor de base de datos de Microsoft Access (https://www.microsoft.com/en-us/download/details.aspx?id=54920) en su equipo.
SQL Server	• **Cliente nativo SQL Server** • **Proveedor para SQL Server de Microsoft OLE DB**
Otras bases de datos	Seleccione el proveedor correspondiente. Si no hay un proveedor OLE DB para su BD, instale el controlador necesario que ofrece el proveedor de la BD (consulte el Resumen de controladores de base de datos). Otra opción es configurar una conexión ADO.NET, ODBC o JDBC.

Para conectarse a esta BD...	Utilice este proveedor...
	Si el sistema operativo tiene un controlador ODBC para la BD, también puede usar **Microsoft OLE DB Provider for ODBC Drivers** o usar una conexión ODBC.

5. Haga clic en **Siguiente**.

Las instrucciones de las siguientes pantallas dependen del proveedor elegido. Para SQL Server deberá dar o seleccionar el nombre de host del servidor de BD, el método de autenticación y el nombre de usuario y la contraseña de la base de datos. Para ver un ejemplo, consulte Connecting to Microsoft SQL Server (ADO). En el caso de Microsoft Access deberá buscar o introducir la ruta de acceso al archivo de BD. Para ver un ejemplo consulte Connecting to Microsoft Access (ADO).

La lista de propiedades de inicialización (parámetros de conexión) aparece en la pestaña *Todas* del cuadro de diálogo de conexión. Estas propiedades dependen del proveedor elegido. Consulte estos apartados para aprender a configurar las propiedades básicas de inicialización para bases de datos Microsoft Access y SQL Server:

- Configurar las propiedades de vínculo de datos de SQL Server
- Configurar las propiedades de vínculo de datos de Microsoft Access

10.2.3.1 Conectarse a una base de datos Microsoft Access

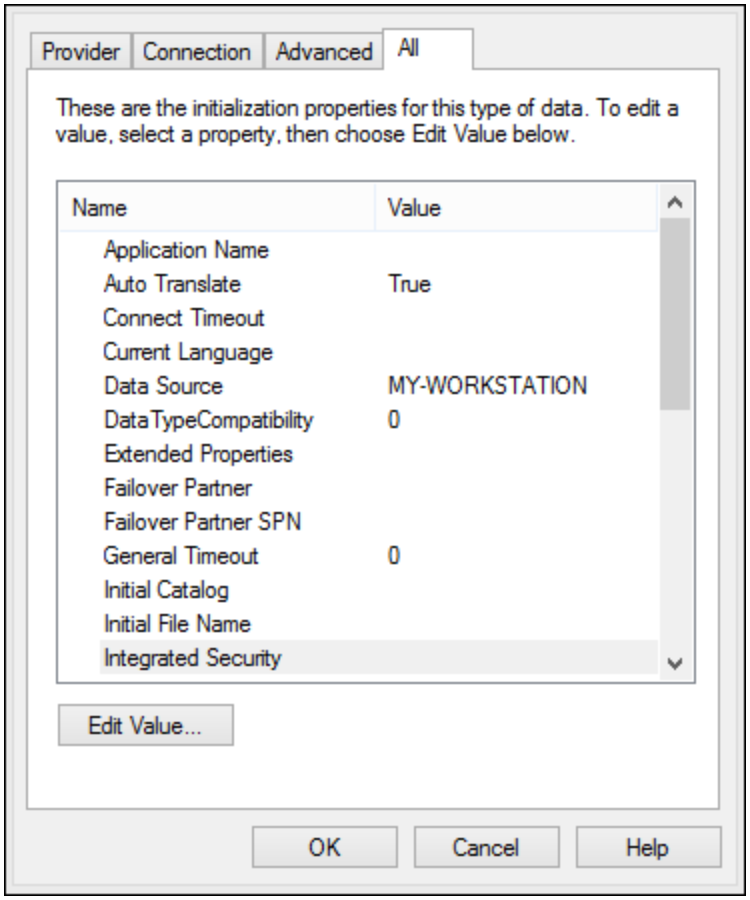
Siga estas instrucciones si desea conectarse a una base de datos Microsoft Access que no requiere contraseña. Si la base de datos está protegida con contraseña, entonces consulte el apartado Conectarse a Microsoft Access (ADO).

Para conectarse a una BD Microsoft Access:

1. Inicie el asistente para la conexión de base de datos.
2. Seleccione el botón de opción *Microsoft Access (ADO)* y haga clic en **Siguiente**.
3. Busque el archivo de BD o introduzca su ruta de acceso (relativa o absoluta).
4. Haga clic en **Conectarse**.

10.2.3.2 Configurar las propiedades de vínculo de datos de SQL Server

Cuando se conecte a una base de datos Microsoft SQL Server por ADO (consulte Conexiones ADO), asegúrese de configurar correctamente estas propiedades de vínculo de datos en la pestaña *Todas* del cuadro de diálogo "Propiedades de vínculo de datos".

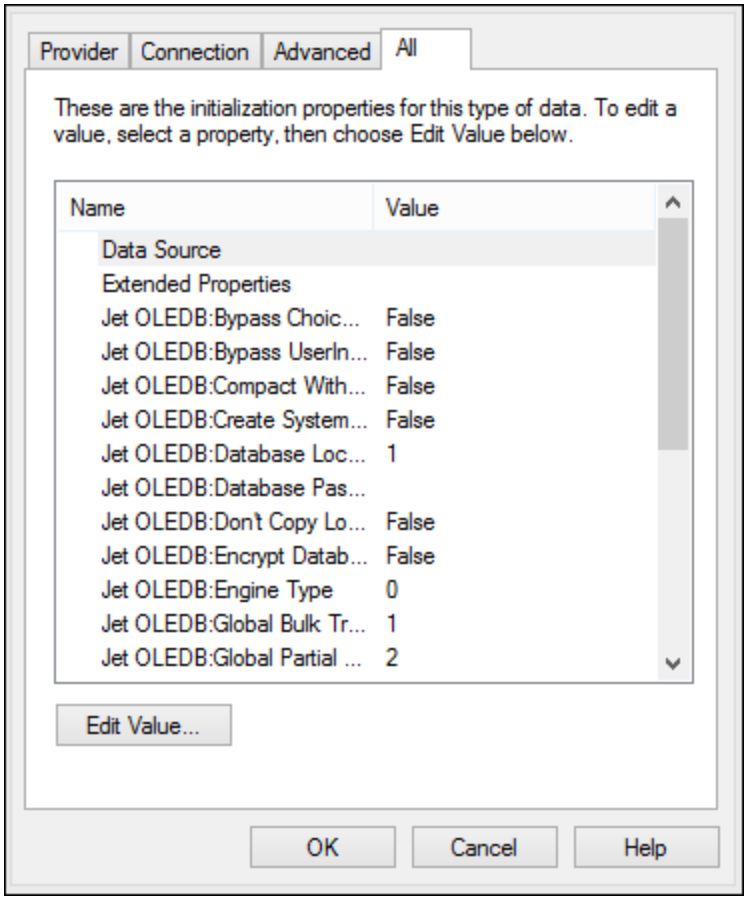


Data Link Properties dialog box

Propiedad	Notas
Seguridad integrada	Si seleccionó el proveedor de datos **SQL Server Native Client** en la pestaña *Proveedor*, esta propiedad debe ser un carácter de espacio en blanco.
Almacenar información de seguridad	Esta propiedad debe tener el valor **True**.

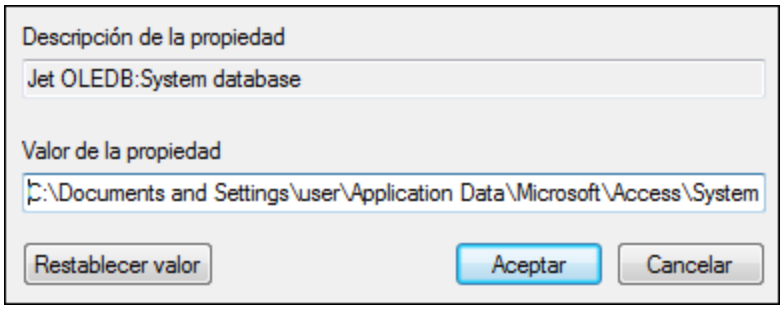
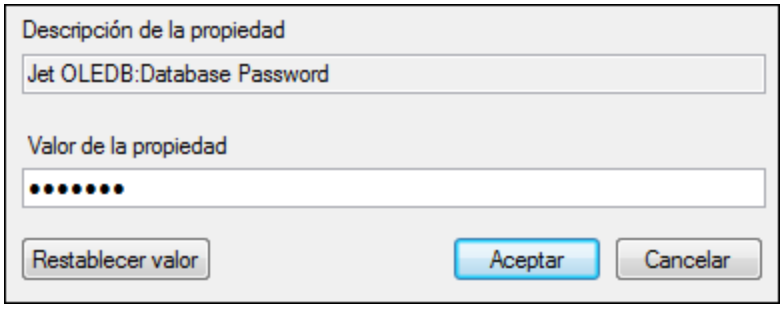
10.2.3.3 Configurar las propiedades de vínculo de datos de Microsoft Access

Cuando se conecte a una base de datos Microsoft Access por ADO (consulte Conexiones ADO), puede que deba configurar estas propiedades de vínculo de datos en la pestaña *Todas* del cuadro de diálogo "Propiedades de vínculo de datos".



Data Link Properties dialog box

Propiedad	Notas
Origen de datos	Esta propiedad almacena la ruta de acceso del archivo de BD Microsoft Access. Para evitar problemas de conexión, recomendamos usar el formato de ruta de acceso UNC. Por ejemplo: `\\servidor\compartir$\rutaArchivo`
Base de datos OLEDB:System Database	Esta propiedad almacena la ruta de acceso del archivo de información de grupo de trabajo. Quizás sea necesario configurar el valor de esta propiedad para poder establecer la conexión con la BD Microsoft Access. Si se produce un error relacionado con el archivo de información de grupo de trabajo, busque el archivo de información de grupo de trabajo (**System.MDW**) que corresponda a su perfil de usuario e introduzca la ruta de acceso del archivo **System.MDW** como valor de esta propiedad.

Propiedad	Notas
	
Contraseña Jet OLEDB:Database	Si la base de datos está protegida con contraseña, el valor de esta propiedad debe ser la contraseña de la base de datos. 

10.2.4 Conexiones ADO.NET

ADO.NET es un conjunto de bibliotecas de Microsoft .NET Framework diseñado para interactuar con datos, incluidos datos de bases de datos. Para conectarse a una base de datos desde UModel por ADO.NET es necesario tener instalado Microsoft .NET Framework 4 o superior. Como puede ver más abajo, la conexión a la base de datos a través de ADO.NET se hace seleccionando un proveedor .NET y aportando una cadena de conexión.

Un proveedor de datos .NET es una colección de clases que permite conectarse a un tipo concreto de origen de datos (p. ej. un servidor SQL Server o una base de datos Oracle), ejecutar comandos en él y recuperar sus datos. En otras palabras, con ADO .NET las aplicaciones como UModel interactúan con una base de datos a través de un proveedor de datos. Cada proveedor de datos está optimizado para poder trabajar con el tipo concreto de origen de datos para el que está diseñado. Hay dos tipos de proveedores .NET:

1. El proveedor que viene por defecto con Microsoft .NET Framework.
2. Proveedores que ofrecen los principales proveedores de bases de datos como extensión para .NET Framework. Este tipo de proveedores ADO.NET deben instalarse por separado y por lo general se pueden descargar del sitio web del correspondiente proveedor de base de datos.

Nota: ciertos proveedores ADO.NET son incompatibles o tienen una compatibilidad limitada. Consulte el apartado Notas sobre compatibilidad con ADO.NET.

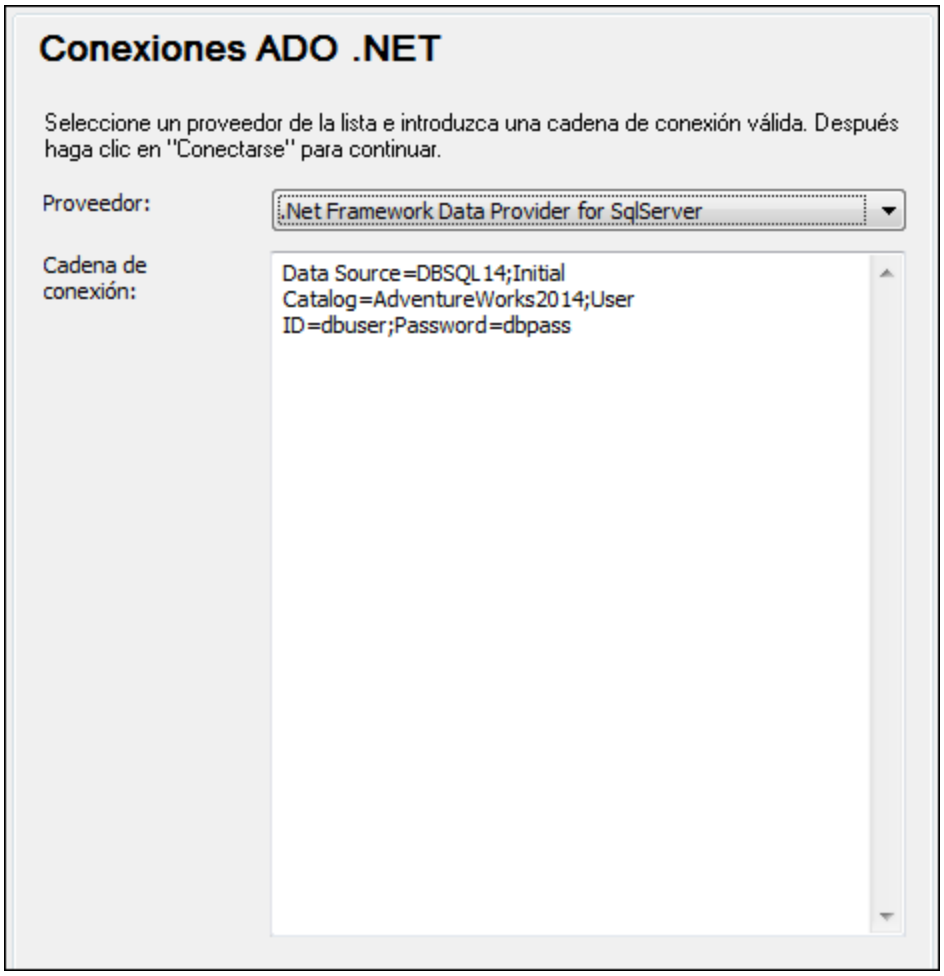
Para configurar una conexión ADO.NET:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones ADO.NET**.
3. Seleccione un proveedor de datos .NET de la lista.

> En la lista *Proveedores* aparecen todos los proveedores que vienen por defecto con .NET Framework. Los proveedores de datos .NET propios del proveedor de la base de datos solo aparecen en la lista si ya están instalados en el sistema. Deben instalarse en el caché global de ensamblados (GAC) con ayuda del archivo .msi o .exe que ofrece el proveedor de la base de datos.

4. Introduzca la cadena de conexión con la base de datos. Una cadena de conexión defina la información de conexión con la base de datos y está formada por pares clave/valor de parámetros de conexión delimitados por caracteres de punto y coma. Por ejemplo, la cadena de conexión `Data Source=DBSQLSERV;Initial Catalog=ProductsDB;User ID=dbuser;Password=dbpass` permite conectarse a la base de datos SQL Server `ProductsDB` del servidor `DBSQLSERV`, con el nombre de usuario `dbuser` y la contraseña `dbpass`. Puede crear la cadena de conexión tecleando los pares clave/valor directamente en el cuadro de texto *Cadena de conexión*, pero también puede crearla con Visual Studio (*véase* *Crear una cadena de conexión en Visual Studio*).

> La sintaxis de la cadena de conexión depende del proveedor que se seleccione en la lista *Proveedores*. Para ver un ejemplo consulte el apartado Ejemplo: cadenas de conexión ADO.NET.



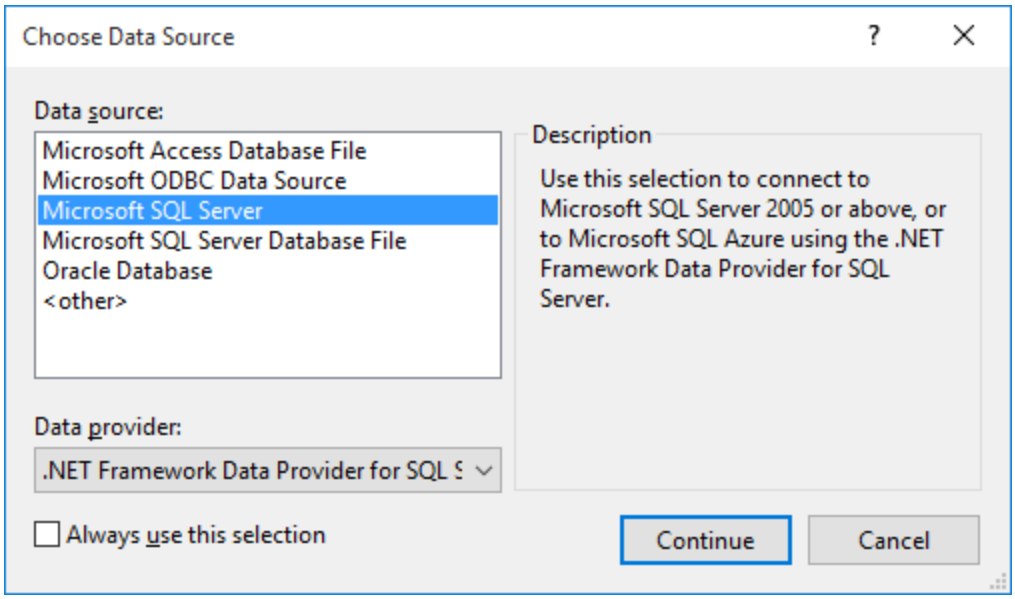
5. Haga clic en **Conectarse** para terminar.

10.2.4.1 Crear una cadena de conexión en Visual Studio

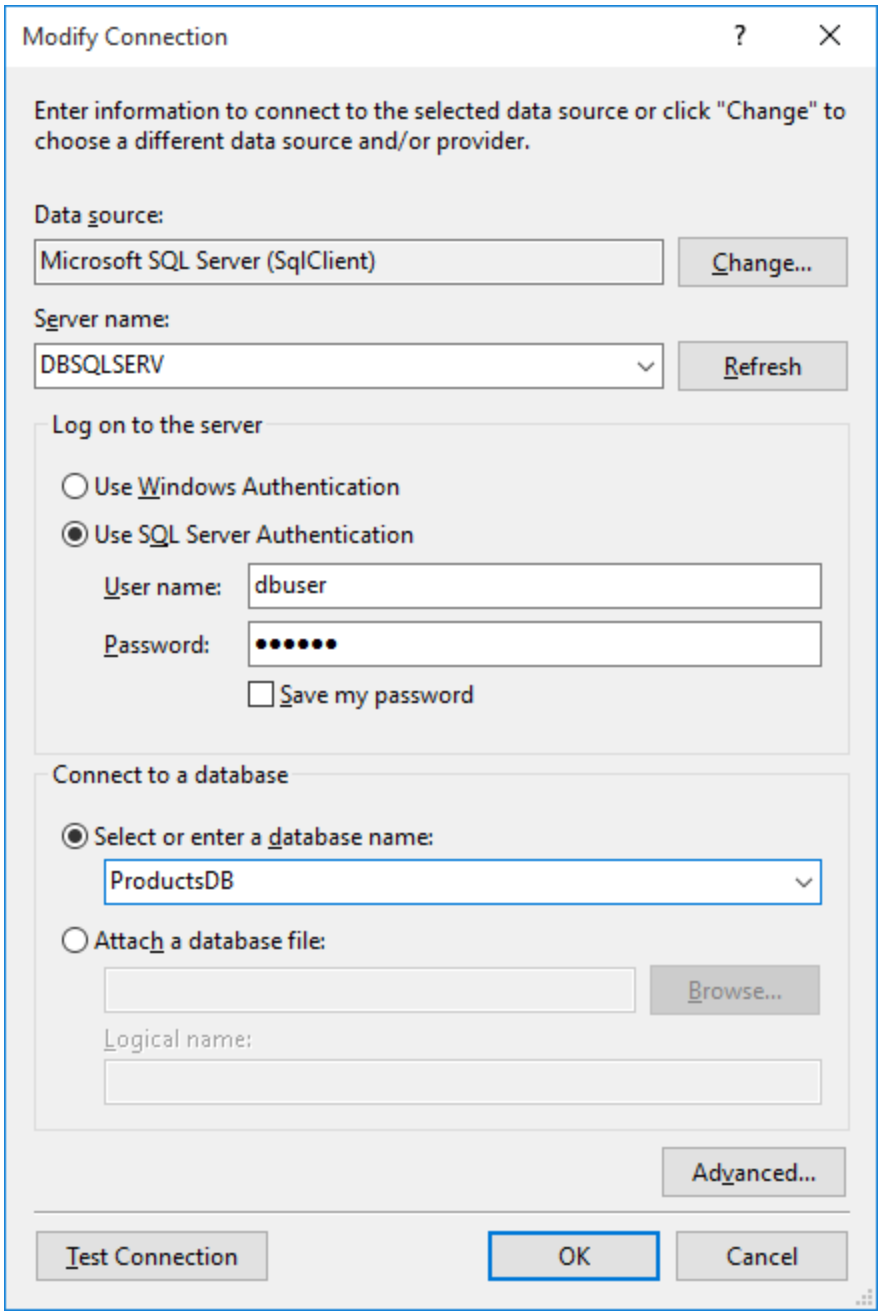
Para conectarse a un origen de datos usando ADO.NET se necesita una cadena de conexión válida. A continuación explicamos cómo crear una cadena de conexión desde Visual Studio.

Para crear una cadena de conexión en Visual Studio:

1. En el menú **Herramientas** haga clic en **Conectar a base de datos**.
2. Seleccione un origen de datos de la lista (en este ejemplo Microsoft SQL Server). El proveedor de datos se rellena automáticamente en función de la opción elegida.



3. Haga clic en **Continuar**.



4. Introduzca el nombre de host del servidor, el nombre de usuario y la contraseña de la base de datos. En este ejemplo nos conectamos a la base de datos `ProductsDB` en el servidor `DBSQLSERV`, usando SQL Server para la autenticación.
5. Para terminar haga clic en **Aceptar**.

Si la conexión se establece correctamente, aparecerá en la ventana Explorador de servidores. Para abrir esta ventana puede usar el comando **Vista | Explorador de servidores**. Para obtener la cadena de conexión con la base de datos, haga clic con el botón derecho en la conexión en la ventana Explorador de servidores y seleccione el comando **Propiedades**. Ahora aparece la cadena de conexión en la ventana Propiedades de

Visual Studio. Recuerde que debe reemplazar los asteriscos con la contraseña antes de pegar la cadena en el cuadro de texto *Cadena de conexión* de UModel.

10.2.4.2 Ejemplo: cadenas de conexión ADO.NET

Para configurar una conexión ADO.NET deberá seleccionar un proveedor ADO.NET en el cuadro de diálogo de conexión a la base de datos y deberá introducir una cadena de conexión (*véase Conexiones ADO.NET*). A continuación ofrecemos ejemplos de cadenas de conexión ADO.NET para varias bases de datos.

Proveedor de datos .NET para Teradata

Este proveedor se puede descargar del sitio web de Teradata (https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata). Un ejemplo de cadena de conexión sería:

```
Data Source=DirecciónServidor;User Id=usuario;Password=contraseña;
```

Proveedor de datos .NET Framework Data Provider para IBM i

Este proveedor se instala con *IBM i Access Client Solutions - Windows Application Package*. Un ejemplo de cadena de conexión sería:

```
DataSource=DirecciónServidor;UserID=usuario;Password=contraseña;DataCompression=True;
```

Para más información consulte el archivo de ayuda llamado ".NET Provider Technical Reference" que viene con el paquete de instalación.

Proveedor de datos .NET Framework Data Provider para MySQL

Este proveedor se puede descargar del sitio web de MySQL (https://dev.mysql.com/downloads/connector/net/). Un ejemplo de cadena de conexión sería:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

Véase también https://dev.mysql.com/doc/connector-net/en/connector-net-programming-connecting-connection-string.html

Proveedor de datos .NET Framework Data Provider para SQL Server

Un ejemplo de cadena de conexión sería:

```
Data Source=DBSQLSERV;Initial Catalog=ProductsDB;User ID=dbuser;Password=dbpass
```

Véase también https://msdn.microsoft.com/en-us/library/ms254500(v=vs.110).aspx

Proveedor de datos IBM DB2 Data Provider 10.1.2 para .NET Framework 4.0

```
Database=PRODUCTS;UID=usuario;Password=contraseña;Server=localhost:50000;
```

Nota: por lo general este proveedor se instala con el paquete de IBM DB2 Data Server Client. Si después de instalar el paquete de IBM DB2 Data Server Client el proveedor no aparece en la lista de proveedores ADO.NET, consulte esta nota técnica: https://www-01.ibm.com/support/docview.wss?uid=swg21429586.

Véase también
https://www.ibm.com/support/knowledgecenter/en/SSEPGG_10.1.0/com.ibm.swg.im.dbclient.adonet.ref.doc/doc/DB2ConnectionClassConnectionStringProperty.html

Proveedor de datos Oracle para .NET (ODP.NET)

El paquete de instalación que incluye el proveedor ODP.NET se puede descargar del sitio web de Oracle (http://www.oracle.com/technetwork/topics/dotnet/downloads/index.html). Un ejemplo de cadena de conexión sería:

```
Data Source=DSORCL;User Id=usuario;Password=contraseña;
```

En esta cadena de conexión, `DSORCL` es el nombre del origen de datos que apunta a un nombre de servicio Oracle que está definido en el archivo **tnsnames.ora** (tal y como se describe en el apartado Conectarse a Oracle (ODBC)).

Si prefiere establecer la conexión sin configurar un nombre de servicio en el archivo **tnsnames.ora**, entonces use una cadena de conexión parecida a esta:

```
Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=host)(PORT=puerto)))
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=MyOracleSID)));User
Id=usuario;Password=contraseña;
```

Véase también https://docs.oracle.com/cd/B28359_01/win.111/b28375/featConnecting.htm

10.2.4.3 Notas sobre compatibilidad con ADO.NET

En esta tabla aparecen controladores de base de datos ADO.NET conocidos que por ahora no son compatibles con UModel o que tienen una compatibilidad limitada.

Base de datos	Controlador	Notas
Todas las bases de datos	**Proveedor de datos .Net Framework para ODBC**	Compatibilidad limitada. Existen problemas conocidos con las conexiones Microsoft Access. Es mejor utilizar conexiones directas ODBC.
	Proveedor de datos .Net Framework para OleDb	Compatibilidad limitada. Existen problemas conocidos con las conexiones Microsoft Access. Es mejor utilizar conexiones directas ADO.
Firebird	**Proveedor de datos Firebird ADO.NET**	Compatibilidad limitada. Es mejor utilizar ODBC o JDBC.
Informix	**Proveedor de datos IBM Informix para .NET Framework 4.0**	Incompatible. Utilice el proveedor **DB2 Data Server Provider**.
IBM DB2 for i (iSeries)	**Proveedor de datos .Net Framework para i5/OS**	Incompatible. utilice el **proveedor de datos .Net Framework para IBM i**, que se instala con el paquete *IBM i Access Client Solutions - Windows Application Package*.
Oracle	**Proveedor de datos .Net Framework para Oracle**	Compatibilidad limitada. Aunque este controlador viene con .NET Framework, Microsoft no recomienda su uso porque es obsoleto.
PostgreSQL	-	Para este proveedor no hay controladores ADO.NET compatibles. Es mejor usar una conexión nativa.
Sybase	-	Para este proveedor no hay controladores ADO.NET compatibles.

10.2.5 Conexiones ODBC

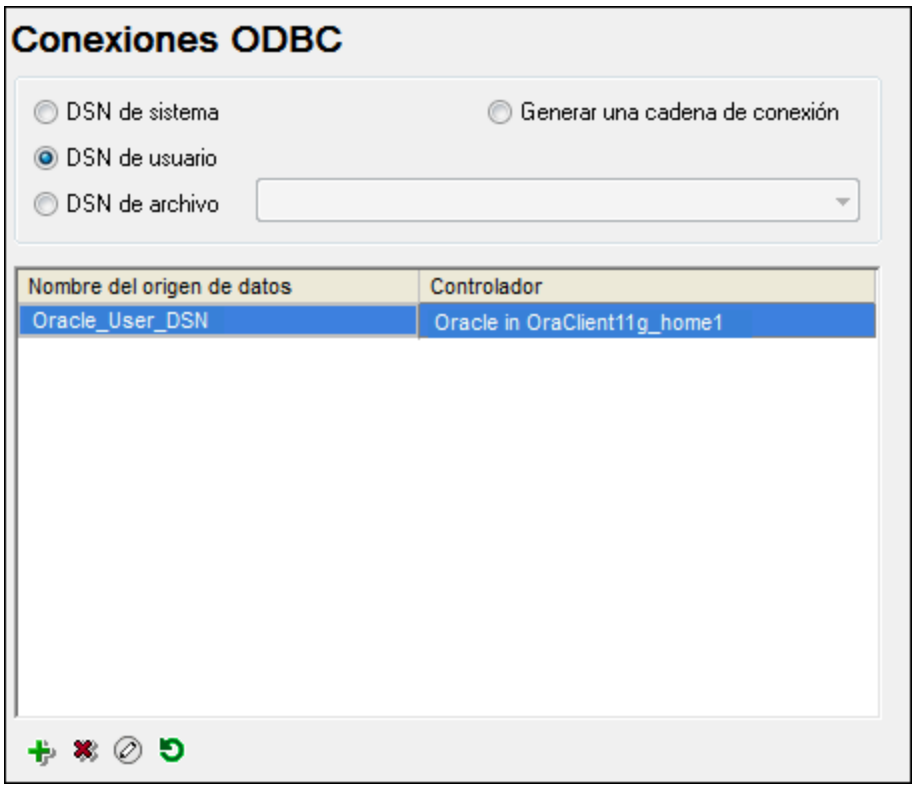
ODBC (Open Database Connectivity) es una tecnología de acceso de datos muy popular con la que se puede conectar a bases de datos desde UModel. Puede utilizarse como método principal de conexión con bases de datos o como alternativa a las conexiones OLE DB o JDBC.

Para conectarse a una base de datos por ODBC primero es necesario contar con un DSN de ODBC en el sistema operativo. El DSN describe de manera uniforme la conexión de BD a todas las aplicaciones cliente compatibles con ODBC que estén en el sistema, incluido UModel. Los DSN pueden ser de varios tipos:

- DSN de sistema
- DSN de usuario
- DSN de archivo

A un origen de datos de sistema pueden acceder todos los usuarios que tengan privilegios en el sistema operativo. A un origen de datos de usuario solo puede acceder el usuario que lo creó. Y, por último, si crea un DSN de archivo, el origen de datos se creará como archivo con extensión `.dsn` que podrá compartir con otros usuarios (siempre que tengan instalados los controladores que utiliza el origen de datos).

Los DSN que estén en el equipo aparecen en el cuadro de diálogo de conexión de base de datos al hacer clic en **Conexiones ODBC**.



Cuadro de diálogo Conexiones ODBC

Si el DSN de la base de datos no existe, el asistente para la conexión de base de datos de UModel le ayudará a crearlo. Si lo prefiere puede crearlo en Windows directamente. En ambos casos, antes de continuar, compruebe que el controlador ODBC para la base de datos está en la lista de controladores disponibles del sistema operativo (consulte el apartado Ver los controladores disponibles).

Para conectarse usando un DSN nuevo:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en el botón **Conexiones ODBC**.
3. Seleccione un tipo de origen de datos (DSN de usuario, de sistema o de archivo).

> **Nota:** para crear un DSN de sistema necesitará derechos de administrador en el sistema y UModel debe ejecutarse como administrador.

4. Haga clic en **Agregar** .
5. Seleccione un controlador y haga clic en **DSN de usuario** o **DSN de sistema**. Si el controlador correspondiente a la BD no aparece en la lista, descárguelo e instálelo.
6. En el cuadro de diálogo que aparece debe rellenar la información de conexión para terminar de configurar la conexión.

Para que la conexión se establezca correctamente deberá dar el nombre de host (o dirección IP) del servidor de la BD, así como el nombre de usuario y la contraseña. Quizás sean necesarios otros parámetros de conexión. Para más información consulte la documentación que ofrece el proveedor de la base de datos. Una vez creado, el DSN estará en la lista de nombres de orígenes de datos. Así podrá volver a usar los datos de la conexión cada vez que desee conectarse a la BD.

Para conectarse por medio de un DSN:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en el botón **Conexiones ODBC**.
3. Elija el tipo de origen de datos (DSN de usuario, de sistema o de archivo).
4. Haga clic en el DSN y después en **Conectarse**.

Para generar una cadena de conexión basada en un archivo .dsn:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en el botón **Conexiones ODBC**.
3. Seleccione *Generar una cadena de conexión* y después haga clic en **Generar**.
4. Si quiere generar la conexión a partir de un DSN de archivo, haga clic en la pestaña *Origen de datos de archivo.* Si no es así, haga clic en la pestaña *Origen de datos de equipo* (los DSN de sistema y de usuario se denominan orígenes de datos de equipo)
5. Seleccione el archivo `.dsn` correspondiente y haga clic en **Aceptar**.

Para conectarse a través de una cadena de conexión preparada previamente:

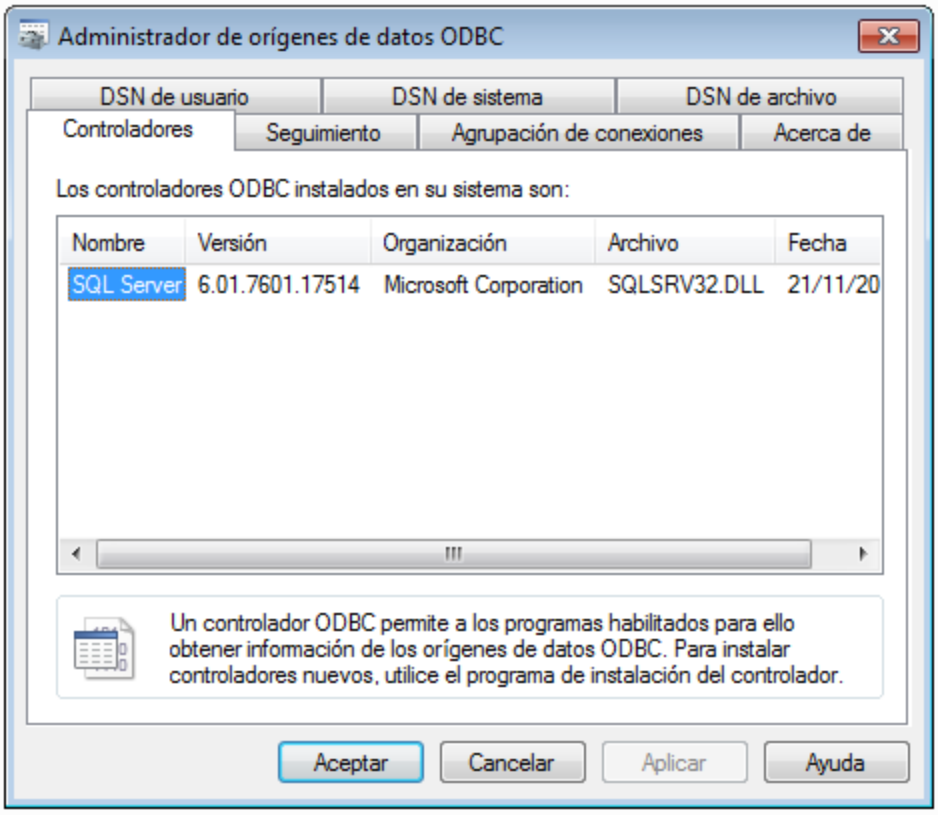
1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en el botón **Conexiones ODBC**.
3. Seleccione *Generar una cadena de conexión*.
4. Pegue la cadena de conexión en el cuadro de texto y haga clic en **Conectarse**.

10.2.5.1 Controladores ODBC disponibles

En el administrador de orígenes de datos ODBC puede ver qué controladores ODBC están disponibles en su sistema operativo. El administrador (**Odbcad32.exe**) se puede abrir desde el panel de control de Windows (desde **Herramientas administrativas**). En sistemas operativos de 64 bits encontrará dos versiones de este ejecutable:

- La versión de 32 bits está en el directorio **C:\Windows\SysWoW64** (siempre y cuando **C:** sea su unidad de sistema).
- La versión de 64 bits está en el directorio **C:\Windows\System32**.

Los controladores de BD de 43 bits aparecerán en la versión de 32 bits del administrador de orígenes de datos ODBC, mientras que los controladores de 64 bits aparecerán en la versión de 64 bits. Esto debe tenerse en cuenta a la hora de consultar los controladores en el administrador.



Si el controlador de la base de datos de destino no está en la lista del administrador o si desea agregar algún controlador, deberá descargar el controlador (Resumen de controladores de base de datos). Cuando el controlador esté disponible en el sistema podrá crear conexiones ODBC con él.

10.2.6 Conexiones JDBC

JDBC (Java Database Connectivity) es una interfaz de acceso a base de datos que forma parte de la plataforma de software Java de Oracle. Las conexiones JDBC suelen consumir más recursos que las conexiones ODBC pero pueden ofrecer más características.

Requisitos

- Tener instalado Java Runtime Environment (JRE) o Java Development Kit (JDK). Este último puede ser el JDK de Oracle o uno de código abierto, como Oracle OpenJDK. UModel identifica la ruta de acceso

al equipo virtual Java (JVM) a partir de estas ubicaciones, en este orden: a) la ruta personal de acceso al JVM que puede indicar en en las **Opciones** (véase el apartado Java); b) la ruta de acceso al JVM que se encuentra en el registro; c) la variable de entorno `JAVA_HOME`.

- Asegúrese de que la plataforma de UModel (32 o 64 bits) es la misma que la de JRE/JDK.
- Tener instalados los controladores JDBC del proveedor de la BD. Estos pueden ser los controladores JDBC que forman parte de la instalación de un cliente de BD o bibliotecas JDBC (archivos .jar) que haya descargado por separado, siempre que estos estén disponible y la BD sea compatible (*véase también Ejemplos de conexión a bases de datos*).
- La variable del entorno `CLASSPATH` debe incluir la ruta de acceso del controlador JDBC (que puede ser un archivo .jar o varios). Algunos clientes de base de datos configuran esta variable automáticamente durante la instalación. Para más información consulte el apartado Configurar la variable CLASSPATH.

Conectarse a un servidor SQL a través de JDBC con credenciales de Windows

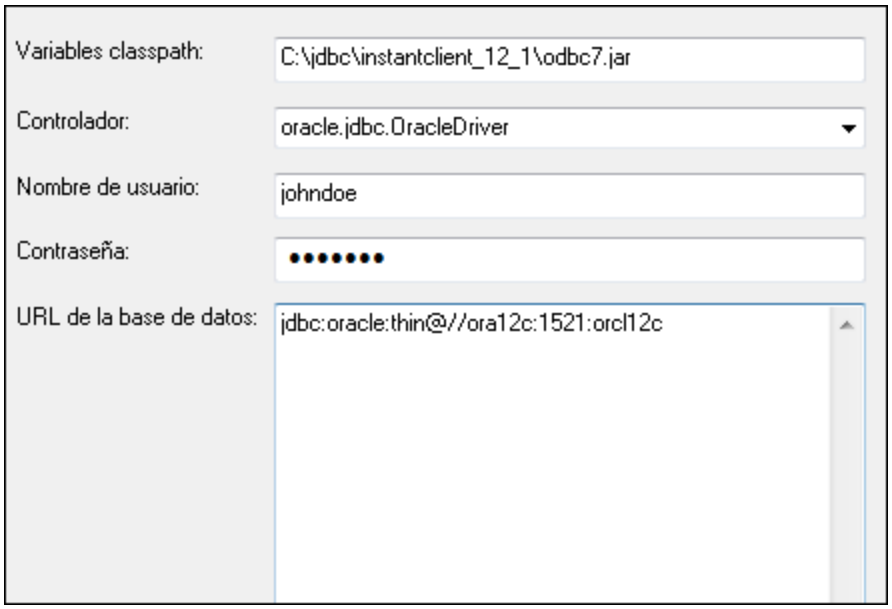
Si se conecta a un servidor SQL mediante JDBC con credenciales de Windows (seguridad integrada), tenga en cuenta lo siguiente:

- Debe copiar el archivo **sqljdbc_auth.dll** que viene con el paquete del controlador JDBC a un directorio que esté en la variable de entorno PATH del sistema. Existen dos archivos de este tipo, uno para la plataforma x86 y otro para la plataforma x64. Debe asegurarse de que añade a la ruta PATH el que corresponde con su plataforma de JDK.
- La cadena de conexión JDBC debe incluir la propiedad **`integratedSecurity=true`**.

Para más información consulte la documentación Controlador JDBC de Microsoft para SQL Server: https://docs.microsoft.com/es-es/sql/connect/jdbc/building-the-connection-url.

Para configurar una conexión JDBC:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en el botón **Conexiones JDBC**.
3. Si quiere, introduzca una lista de rutas de archivo .jar separadas por caracteres de punto y coma en el cuadro de texto *Variables classpath*. Las bibliotecas .jar que se introduzcan aquí se cargarán en el entorno junto a las que ya estén definidas en la variable de entorno `CLASSPATH`. Cuando termine de editar el cuadro de texto Variables classpath, todos los controladores JDBC encontrados en las bibliotecas .jar de origen se cargan automáticamente a la lista del cuadro combinado *Controlador*.



4. En el cuadro combinado *Controlador* seleccione un controlador JDBC de la lista o introduzca un nombre de clase Java. Observe que esta lista contiene todos los controladores JDBC configurados a través de la variable de entorno `CLASSPATH` (*véase* *Configurar la variable CLASSPATH*), así como los controladores encontrados en el campo *Variables classpath*.

> Las rutas de acceso del controlador JDBC definidas sen la variable CLASSPATH, así como las rutas de acceso de los archivos .jar introducidos directamente en el cuadro de diálogo de conexión a la base de datos se envían a Java Virtual Machine (JVM). JVM decide qué controladores se utilizan para establecer la conexión. Se recomienda realizar un seguimiento de las clases Java que se cargan en JVM para evitar conflictos y resultados inesperados a la hora de conectarse a la base de datos.

5. Introduzca el nombre de usuario y la contraseña de la BD.
6. En el cuadro de texto *URL de la base de datos:* introduzca la URL de la conexión JDBC en el formato propio del tipo de base de datos utilizado. En la siguiente tabla puede ver la sintaxis de las URL de conexión JDBC para los tipos de base de datos más frecuentes.

Base de datos	URL de conexión JDBC
Firebird	`jdbc:firebirdsql://<host>[:<puerto>]/<ruta de acceso o alias de la BD>`
IBM DB2	`jdbc:db2://nombreHost:puerto/nombreBaseDatos`
IBM DB2 for i	`jdbc:as400://[host]`
IBM Informix	`jdbc:informix-sqli://nombreHost:puerto/nombreBaseDatos:INFORMIXSERVER=miservidor`
MariaDB	`jdbc:mariadb://nombreHost:puerto/nombreBaseDatos`

Base de datos	URL de conexión JDBC
Microsoft SQL Server	jdbc:sqlserver://**nombreHost**:**puerto**;**nombreBaseDatos**=name
MySQL	jdbc:mysql://**nombreHost**:**puerto**/**nombreBaseDatos**
Oracle	jdbc:oracle:thin:@**nombreHost**:**puerto**:**SID** jdbc:oracle:thin:@//**nombreHost**:**puerto**:**servicio**
Oracle XML DB	jdbc:oracle:oci:@//**nombreHost**:**puerto**:**servicio**
PostgreSQL	jdbc:postgresql://**nombreHost**:**puerto**/**nombreBaseDatos**
Progress OpenEdge	jdbc:datadirect:openedge://**host**:**puerto**;databaseName=**nombre_bd**
Sybase	jdbc:sybase:Tds:**nombreHost**:**puerto**/**nombreBaseDatos**
Teradata	jdbc:teradata://**nombreServidorBaseDatos**

Nota: en algunos casos puede modificarse el formato sintáctico (p. ej. la URL de la base de datos puede excluir el puerto o puede incluir el nombre de usuario y la contraseña). Consulte la documentación del proveedor de BD para obtener más información.

7. Haga clic en **Conectarse**.

10.2.6.1 Configurar la variable CLASSPATH

La variable de entorno `CLASSPATH` es utilizada por Java Runtime Environment (JRE) o por Java Development Kit (JDK) para encontrar las clases Java y otros archivos de recursos del sistema operativo. Cuando se conecte a una BD con JDBC, esta variable debe incluir la ruta de acceso del controlador JDBC del sistema y, en algunos casos, la ruta de acceso de otros archivos de biblioteca relacionados con el tipo de BD que esté utilizando.

En la tabla que aparece a continuación puede ver ejemplos de variables `CLASSPATH`. Lo más importante que debe tener en cuenta es que quizás deba adaptar esta información dependiendo de la ubicación y del nombre del controlador JDBC y de la versión JRE/JDK que esté en su sistema operativo. Para evitar problemas de conexión recomendamos leer detenidamente las instrucciones de instalación del controlador JDBC que esté instalado en el sistema.

Base de datos	Ejemplo de variable CLASSPATH
Firebird	C:\Archivos de programa\Firebird\Jaybird-2.2.8-JDK_1.8\jaybird-full-2.2.8.jar
IBM DB2	C:\Archivos de programa (x86)\IBM\SQLLIB\java\db2jcc.jar;C:\Archivos de programa (x86)\IBM\SQLLIB\java\db2jcc_license_cu.jar;
IBM DB2 para i	C:\jt400\jt400.jar;

Base de datos	Ejemplo de variable CLASSPATH
IBM Informix	`C:\Informix_JDBC_Driver\lib\ifxjdbc.jar;`
MariaDB	**<directorio de instalación>**\mariadb-java-client-2.2.0.jar
Microsoft SQL Server	`C:\Archivos de programa\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_4.0\enu\sqljdbc.jar`
MySQL	**<directorio de instalación>**\mysql-connector-java-*version*-bin.jar;
Oracle	**ORACLE_HOME**\jdbc\lib\ojdbc6.jar;
Oracle (with XML DB)	**ORACLE_HOME**\jdbc\lib\ojdbc6.jar;**ORACLE_HOME**\LIB\xmlparserv2.jar; **ORACLE_HOME**\RDBMS\jlib\xdb.jar;
PostgreSQL	**<directorio de instalación>**\postgresql.jar
Progress OpenEdge	`%DLC%\java\openedge.jar;%DLC%\java\pool.jar;` **Nota:** siempre y cuando Progress OpenEdge SDK esté instalado en el equipo, `%DLC%` es el directorio donde está instalado OpenEdge.
Sybase	`C:\sybase\jConnect-7_0\classes\jconn4.jar`
Teradata	**<directorio de instalación>**\tdgssconfig.jar;**<directorio de instalación>**\terajdbc4.jar

- Los cambios en la configuración de la variable CLASSPATH pueden afectar al comportamiento de las aplicaciones Java del equipo. Consulte la documentación de Java antes de continuar.
- Las variables de entorno pueden ser del sistema o del usuario. Para cambiar las variables de entorno del sistema es necesario tener derechos de administrador.
- Tras modificar la variable de entorno, reinicie los programas que estén en ejecución para que los cambios surtan efecto. También puede cerrar sesión o reiniciar el sistema.

Para configurar CLASSPATH en Windows 7:

1. Abra el menú **Inicio** y haga clic con el botón derecho en **Equipo**.
2. Haga clic en **Propiedades**.
3. Haga clic en **Configuración avanzada del sistema**.
4. En la pestaña *Avanzadas*, haga clic en **Variables de entorno**.
5. Busque la variable `CLASSPATH` del sistema o del usuario y haga clic en **Editar**. Si no existe, haga clic en **Nueva** para crearla.

6. Edite el valor de la variable e incluya la ruta de acceso del controlador JDBC. Utilice un punto y coma para separar la ruta de acceso del controlador de las demás partes de la variable.

Para configurar CLASSPATH en Windows 10:

1. Pulse la tecla Windows y teclee "variables de entorno".
2. Haga clic en la sugerencia **Editar las variables de entorno del sistema**.
3. Haga clic en **Variables de entorno**.
4. Busque la variable `CLASSPATH` del sistema o del usuario y haga clic en **Editar**. Si no existe, haga clic en **Nueva** para crearla.
5. Edite el valor de la variable e incluya la ruta de acceso del controlador JDBC. Utilice un punto y coma para separar la ruta de acceso del controlador de las demás partes de la variable.

10.2.7 Conexiones PostgreSQL

Las conexiones a bases de datos PostgreSQL pueden configurarse como conexiones nativas o a través de ODBC, JDBC y otros controladores. La ventaja de configurar una conexión nativa es que no es necesario tener controladores instalados en el sistema.

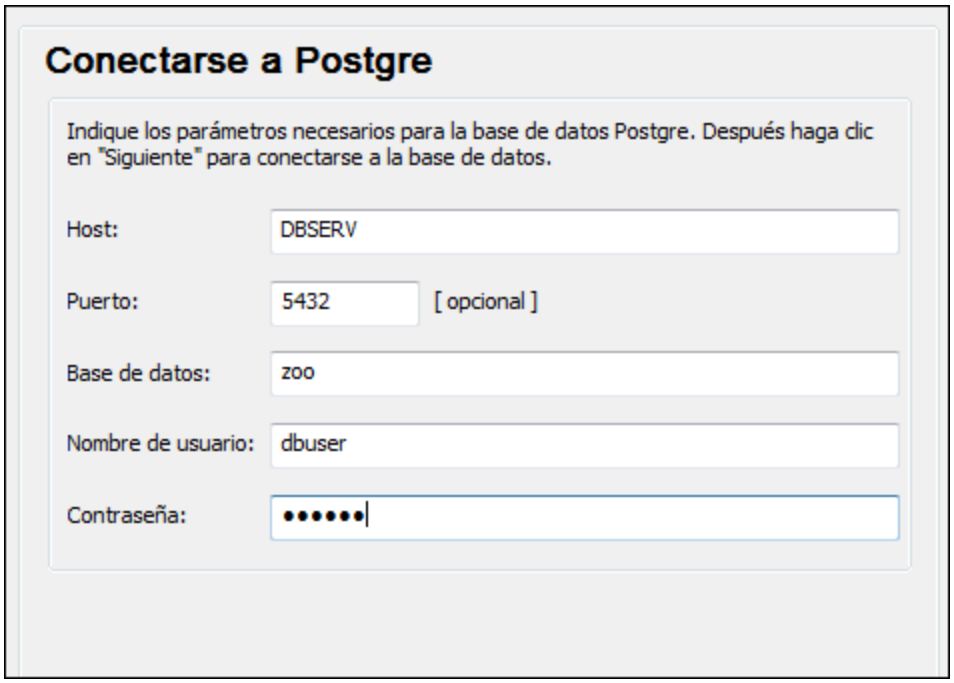
Si prefiere establecer la conexión por medio de un controlador no nativo, consulte estos temas de la documentación:

- Configurar una conexión JDBC
- Conectarse a PostgreSQL (ODBC)

Si lo que quiere es configurar una conexión nativa con PostgreSQL, siga las instrucciones que aparecen más abajo. El único requisito es conocer el nombre de host, el puerto, el nombre de la base de datos, el nombre de usuario y la contraseña.

Para configurar una conexión PostgreSQL nativa:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en el botón **Conexiones PostgreSQL**.
3. Ahora introduzca el host (localhost si PostgreSQL se ejecuta en el mismo equipo), el puerto (es opcional, pero suele ser 5432), el nombre de la BD, el nombre de usuario y la contraseña.



4. Para terminar haga clic en **Conectarse**.

Si el servidor de base de datos PostgreSQL está en otro equipo, debe tener en cuenta lo siguiente:

- El servidor de base de datos PostgreSQL debe estar configurado para aceptar conexiones de clientes. Concretamente, debe configurar el archivo **pg_hba.conf** para que permita conexiones no locales. Además debe configurar el archivo **postgresql.conf** para que escuche determinadas direcciones IP y puertos. Para más información consulte la documentación de PostgreSQL (https://www.postgresql.org/docs/9.5/static/client-authentication-problems.html).
- El equipo servidor debe estar configurado para aceptar conexiones en el puerto correspondiente (suele ser 5432) a través del servidor de seguridad. Por ejemplo, imagine que tiene un servidor de BD que se ejecuta en un equipo Windows. Deberá crear una regla que permita conexiones en el puerto 5432 a través del servidor de seguridad (desde **Panel de control > Firewall de Windows > Configuración avanzada > Reglas de entrada**).

10.2.8 Conexiones SQLite

SQLite (https://www.sqlite.org/index.html) es un tipo de base de datos basado en archivos y con almacenamiento. Las bases de datos SQLite son compatibles de forma nativa con UModel, así que no es necesario instalar ningún controlador para poder conectarse a este tipo de BD.

Notas sobre compatibilidad con bases de datos SQLite

- En Linux no hay un tiempo de espera de ejecución de instrucciones para bases de datos SQLite.
- No son compatibles con búsquedas de texto completo en tablas.
- SQLite admite valores de diferentes tipos de datos en cada fila de una tabla. En MapForce todos los valores procesados deben ser compatibles con el tipo de columna declarado. Por tanto, pueden darse

errores en tiempo de ejecución si la base de datos SQLite tiene valores de fila que no coinciden con el tipo de columna declarado.

Importante

Es recomendable que use la palabra clave `STRICT` al crear tablas para asegurarse de que el comportamiento de los datos es más predecible. De lo contrario, es posible que los datos no se lean o escriban correctamente si en una misma columna hay valores de distintos tipos mezclados. Para saber más sobre las tablas `STRICT` consulte la documentation SQLite.

10.2.8.1 Conectarse a una base de datos SQLite

Para conectarse a una base de datos SQLite:

1. Inicie el asistente para la conexión de base de datos
2. Seleccione el botón de opción *SQLite* y después haga clic en **Siguiente**.
3. Busque el archivo de BD o introduzca su ruta de acceso (relativa o absoluta). El botón **Conectarse** se habilita una vez introducida la ruta de acceso del archivo.
4. Ahora haga clic en **Conectarse**.

10.2.9 Ejemplos de conexión a bases de datos

Esta sección incluye ejemplos de cómo conectarse a una base de datos desde UModel con ADO, ODBC o JDBC. Para ver los ejemplos de ADO.NET consulte el apartado Ejemplo: cadenas de conexión ADO.NET. Para ver cómo establecer una conexión nativa a PostgreSQL y SQLite (véanse respectivamente Conexiones PostgreSQL y Conexiones SQLite).

Tenga en cuenta lo siguiente:

- Puede que las instrucciones sean distintas si su configuración de Windows, entorno de red y cliente de BD o software de servidor no son los mismos que los descritos en el ejemplo correspondiente.
- En la mayoría de los casos se puede establecer la conexión por medio de varias tecnologías de acceso (ADO, ADO.NET, ODBC, JDBC) o controladores. El rendimiento de la conexión de BD, así como sus características y limitaciones, dependerá del controlador, software cliente y parámetros de conexión seleccionados.

10.2.9.1 Firebird (JDBC)

Este ejemplo explica cómo conectarse a una base de datos Firebird por JDBC.

Requisitos:

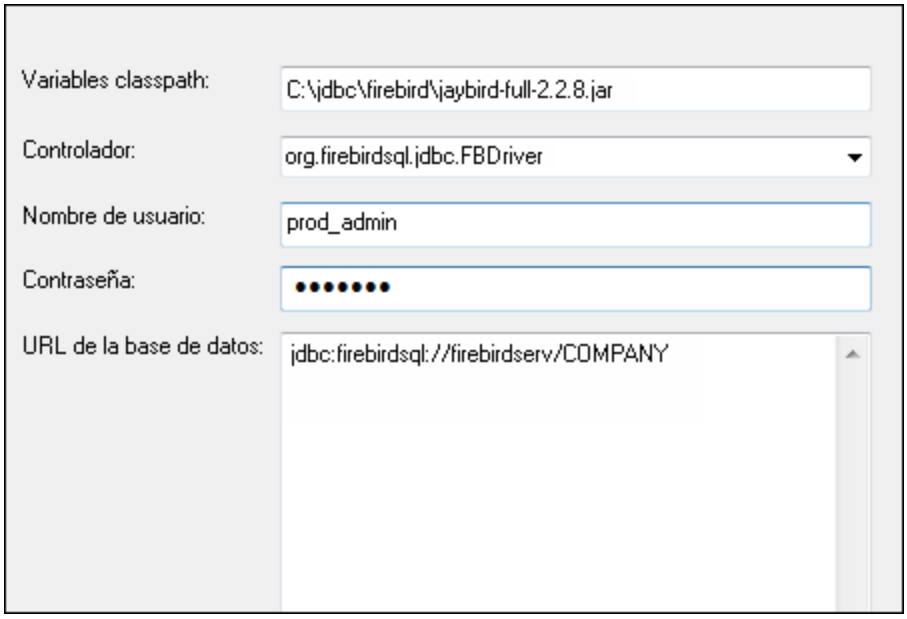
- JRE (Java Runtime Environment) o Java Development Kit (JDK) está instalado. Este último puede ser el JDK de Oracle o uno de código abierto, como Oracle OpenJDK. UModel identifica la ruta de acceso

al equipo virtual Java (JVM) a partir de estas ubicaciones, en este orden: a) la ruta personal de acceso al JVM que puede indicar en en las **Opciones** (véase el apartado Java); b) la ruta de acceso al JVM que se encuentra en el registro; c) la variable de entorno `JAVA_HOME`.

- Asegúrese de que la plataforma de UModel (32 o 64 bits) es la misma que la de JRE/JDK.
- El controlador Firebird JDBC debe estar disponible en el sistema operativo (se trata de un archivo .jar que ofrece conectividad con la base de datos). El controlador se puede descargar del sitio web de Firebird (https://www.firebirdsql.org/). En este ejemplo usamos el controlador *Jaybird 2.2.8*.
- Disponer de los datos de conexión: host, ruta de acceso (o alias) de la base de datos, nombre de usuario y contraseña.

Para conectarse a Firebird por JDBC:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones JDBC**.
3. En el campo *Variables classpath* debe introducir la ruta de acceso del archivo .jar que ofrece conectividad con la base de datos. Si fuera necesario, también puede introducir una lista de rutas de archivo .jar separadas por caracteres de punto y coma. En este ejemplo, el archivo .jar está ubicado en esta ruta de acceso: `C:\jdbc\firebird\jaybird-full-2.2.8.jar`. Tenga en cuenta que este campo puede dejarse en blanco si añadió la ruta de acceso de los archivos .jar a la variable de entorno CLASSPATH del sistema operativo (*véase Configurar la variable CLASSPATH*).
4. En el campo *Controlador* seleccione **org.firebirdsql.jdbc.FBDriver**. Recuerde que esta entrada solo estará disponible si se encuentra una ruta de archivo .jar válida en el campo Variables *classpath* o en la variable de entorno CLASSPATH del sistema operativo.



5. Introduzca el nombre de usuario y la contraseña de la base de datos.
6. Introduzca la cadena de conexión para el servidor de BD en el cuadro de texto *URL de la base de datos* (reemplace lo valores resaltados con los de su base de datos).

```
jdbc:firebirdsql://<host>[:<puerto>]/<ruta o alias de la BD>
```

7. Haga clic en **Conectarse**.

10.2.9.2 Firebird (ODBC)

Este ejemplo explica cómo conectarse a una base de datos Firebird 2.5.4 de un servidor Linux.

Requisitos:

- El servidor de BD Firebird está configurado para aceptar conexiones TCP/IP desde clientes.
- El controlador ODBC de Firebird está instalado en el sistema. Este ejemplo usa la versión 2.0.3.154 del controlador (https://www.firebirdsql.org/).
- El cliente Firebird está instalado en el sistema. Recuerde que el cliente forma parte del paquete de instalación del servidor Firebird. Puede descargar este paquete del sitio web de Firebird (https://www.firebirdsql.org/) y buscar el instalador para Windows (Windows executable installer for full Superclassic/Classic or Superserver). Para instalar los archivos del cliente solamente elija la opción **Minimum client install - no server, no tools** del asistente para la instalación.

  Nota:

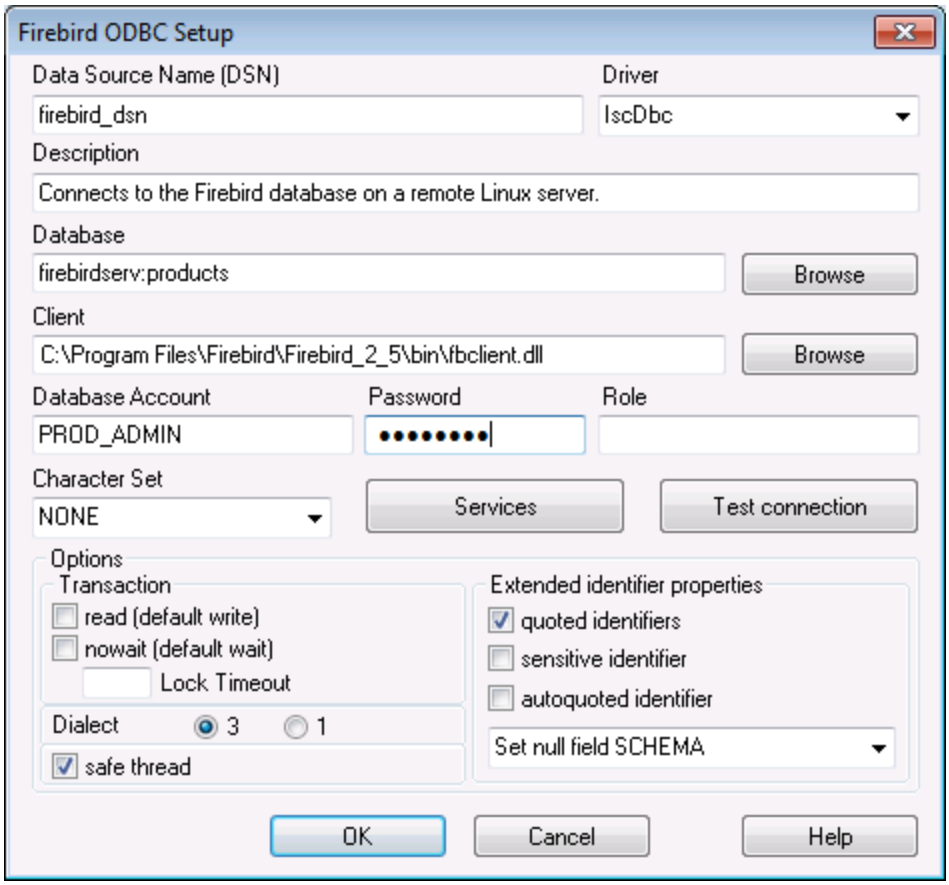
  - La plataforma del controlador ODBC de Firebird y del cliente (de 32 o 64 bits) debe coincidir con la plataforma de UModel.
  - La versión del cliente Firebird debe coincidir con la versión del servidor Firebird al que desea conectarse.

- Disponer de los datos de conexión: nombre de host o dirección IP, ruta de acceso (o alias) de la base de datos en el servidor, nombre de usuario y contraseña.

Para conectarse a Firebird por ODBC:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones ODBC**.
3. Seleccione la opción *DSN de usuario* (o *DSN de sistema* si tiene privilegios de administrador) y después haga clic en **Agregar** .

4. Seleccione el controlador Firebird y después haga clic en *DSN de usuario* (o *DSN de sistema* dependiendo de la opción seleccionada en el paso anterior). Si el controlador Firebird no aparece en la lista, compruebe que está instalado en el sistema.



5. Introduzca los datos de conexión:

Nombre del origen de datos (DSN)	Introduzca un nombre para el origen de datos que desea crear.

Base de datos	Introduzca el nombre de host o dirección IP del servidor, seguida de dos puntos, seguido del alias (o ruta de acceso) de la base de datos. En este ejemplo el nombre de host es `firebirdserv` y el alias de la BD es `products`:
	
	`firebirdserv:products`
	
	Usamos el alias de la BD porque damos por hecho que en el lado servidor el administrador de la BD ha configurado el alias *products* para apuntar al archivo de BD Firebird (.fdb) en el servidor.
	
	En lugar del nombre de host también puede usar la dirección IP del servidor. Y en lugar del alias puede usar una ruta de acceso. Por tanto, también podría usar estas otras dos cadenas de conexión:
	
	`firebirdserver:/var/Firebird/databases/butterflies.fdb`
	`127.0.0.1:D:\Misc\Lenders.fdb`

	Si la base de datos está en el equipo Windows local, haga clic en **Examinar** y seleccione el archivo de base de datos directamente.
Cliente	Introduzca la ruta de acceso del archivo `fbclient.dll`. Su ubicación predeterminada es el subdirectorio `bin` del directorio de instalación de Firebird.
Cuenta de base de datos	Introduzca el nombre de usuario que recibió del administrador de la BD (en este ejemplo es `PROD_ADMIN`).
Contraseña	Introduzca la contraseña de la BD que recibió del administrador.

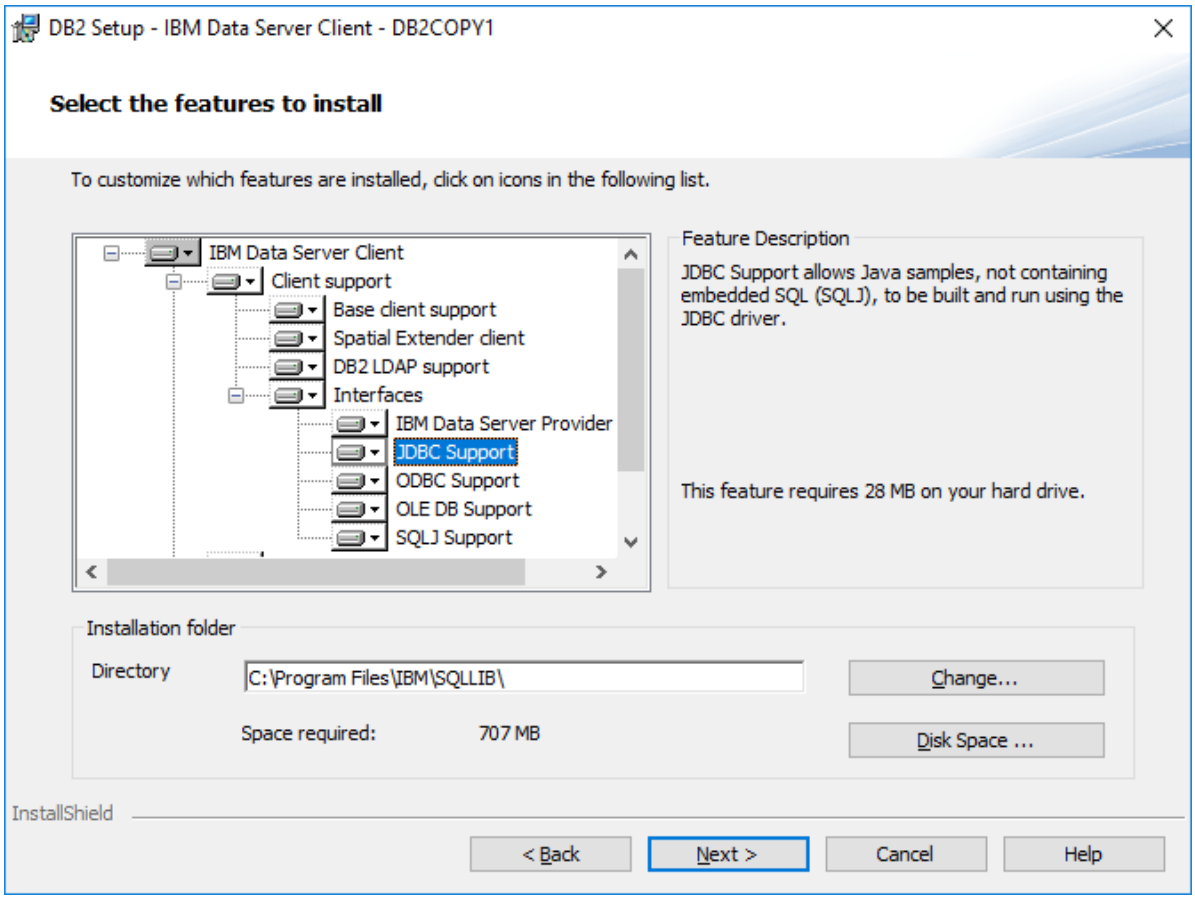
6. Haga clic en **Aceptar**.

10.2.9.3 IBM DB2 (JDBC)

Este ejemplo explica cómo conectarse a una base de datos IBM DB2 por JDBC.

Requisitos:

- JRE (Java Runtime Environment) o Java Development Kit (JDK) está instalado. Este último puede ser el JDK de Oracle o uno de código abierto, como Oracle OpenJDK. UModel identifica la ruta de acceso al equipo virtual Java (JVM) a partir de estas ubicaciones, en este orden: a) la ruta personal de acceso al JVM que puede indicar en en las **Opciones** (véase el apartado Java); b) la ruta de acceso al JVM que se encuentra en el registro; c) la variable de entorno `JAVA_HOME`.
- Asegúrese de que la plataforma de UModel (32 o 64 bits) es del mismo tipo que la de JRE/JDK. Este ejemplo usa Oracle OpenJDK 11.0 de 64 bits, por lo que usa la versión de 64 bits de UModel.
- El controlador JDBC (uno o más archivos .jar que permiten conectarse a la BD) debe estar disponible en el sistema operativo. Este ejemplo usa el controlador JDBC que está disponible tras instalar la versión 10.1 del **IBM Data Server Client** (64 bits). Al instalar los controladores, elija la instalación típica o seleccione esa opción en el asistente de instalación.

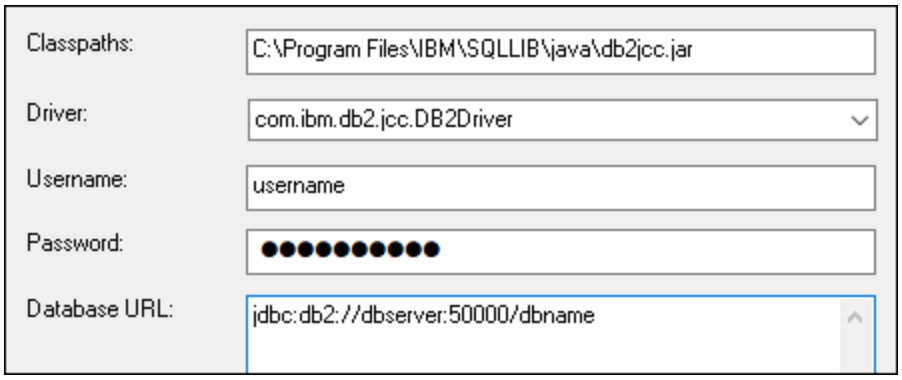


Si no ha modificado la ruta predeterminada de la instalación, una vez esta haya finalizado los archivos .jar requeridos están en el directorio **C:\Program Files\IBM\SQLLIB\java**.

- Necesitará esta información sobre la conexión a la BD: host, puerto, nombre de la BD, nombre de usuario y contraseña.

Para conectarse a IBM DB2 for i por JDBC:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones JDBC**.
3. Junto a "Variables classpath" introduzca la ruta al archivo .jar que permite conectarse a la BD. En este ejemplo la ruta es **C:\Program Files\IBM\SQLLIB\java\db2jcc.jar**. Puede que necesite hacer referencia al controlador **db2jcc4.jar** en función de la versión del servidor de BD. Si tiene dudas sobre si el controlador es compatible consulte la documentación de IBM (http://www-01.ibm.com/support/docview.wss?uid=swg21363866). Puede dejar el campo "Variables classpath" vacío si ha añadido la ruta de acceso del archivo .jar (también pueden ser varios) a la variable de entorno CLASSPATH del sistema operativo (*véase también el apartado Configurar la variable CLASSPATH*).
4. En el campo "Controlador" seleccione **com.ibm.db2.jcc.DB2Driver**. Esta entrada solo está disponible si se encuentra una ruta de acceso válida a un archivo .jar en el campo "Variables classpath" o en la variable de entorno CLASSPATH del sistema operativo (*véase el paso anterior*).



5. Introduzca el nombre de usuario y la contraseña del usuario de la BD en los campos correspondientes.
6. Introduzca la cadena de conexión JDBC en el campo **URL de la BD**. Asegúrese de reemplazar los detalles de la conexión con los de su servidor de BD.

```
jdbc:db2://hostName:port/databaseName
```

7. Haga clic en **Conectarse**.

10.2.9.4 IBM DB2 (ODBC)

Este ejemplo explica cómo conectarse a una base de datos IBM DB2 por ODBC.

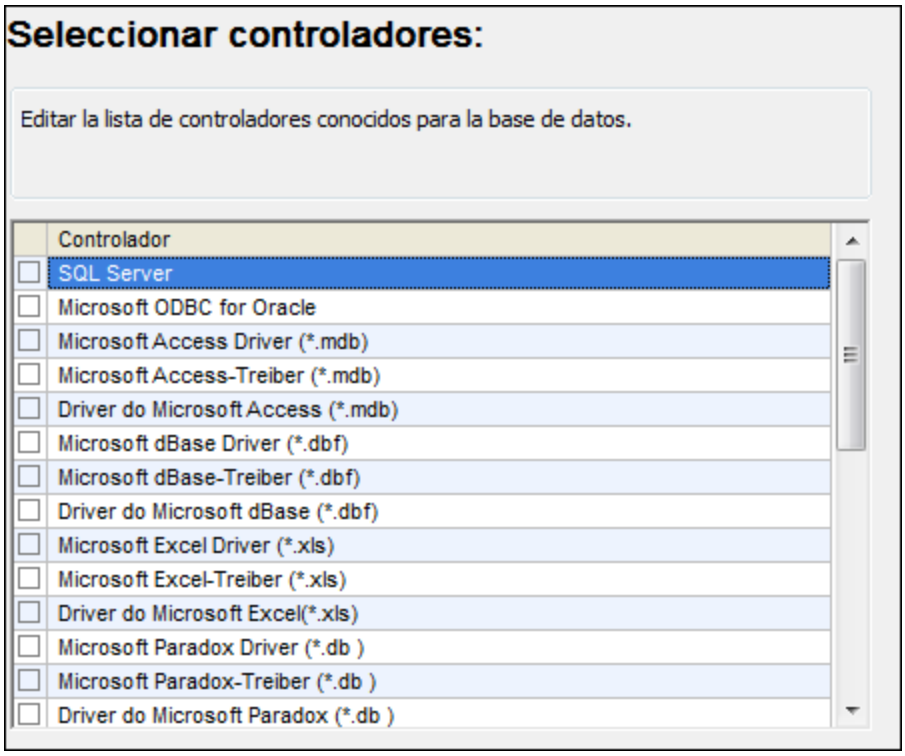
Requisitos:

- El cliente IBM Data Server está instalado y configurado en el sistema (en este ejemplo usamos IBM Data Server Client 9.7). Después de instalar el cliente IBM Data Server compruebe que los controladores ODBC están disponibles en el equipo (consulte Ver los controladores ODBC disponibles).
- Tiene un alias para la base de datos. El alias se puede crear de varias maneras:
  - Con el asistente de configuración de IBM DB2
  - Con el procesador de línea de comandos de IBM DB2
  - Con el asistente para orígenes de datos ODBC (instrucciones más abajo).
- Disponer de los datos de conexión: host, base de datos, puerto, nombre de usuario y contraseña.

Para conectarse a IBM DB2:

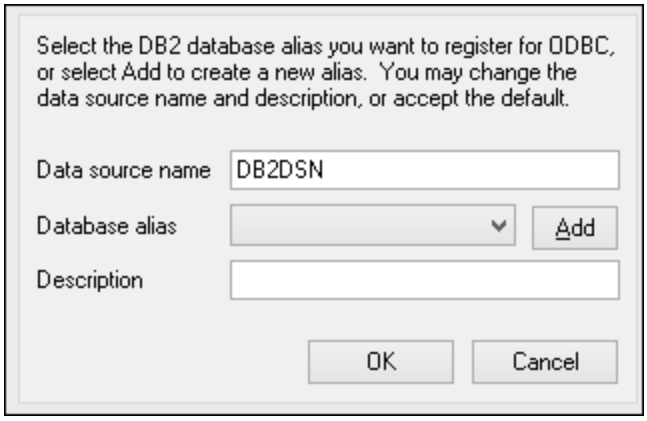
1. Inicie el asistente para la conexión de base de datos y seleccione la opción *IBM DB2 (ODBC/JDBC)*.
2. Haga clic en **Siguiente**.

3. Seleccione la opción *ODBC* y haga clic en **Siguiente**. Si necesita editar la lista de controladores conocidos para la base de datos, seleccione los controladores que corresponden a IBM DB2 (ver Requisitos) y haga clic en **Siguiente**.

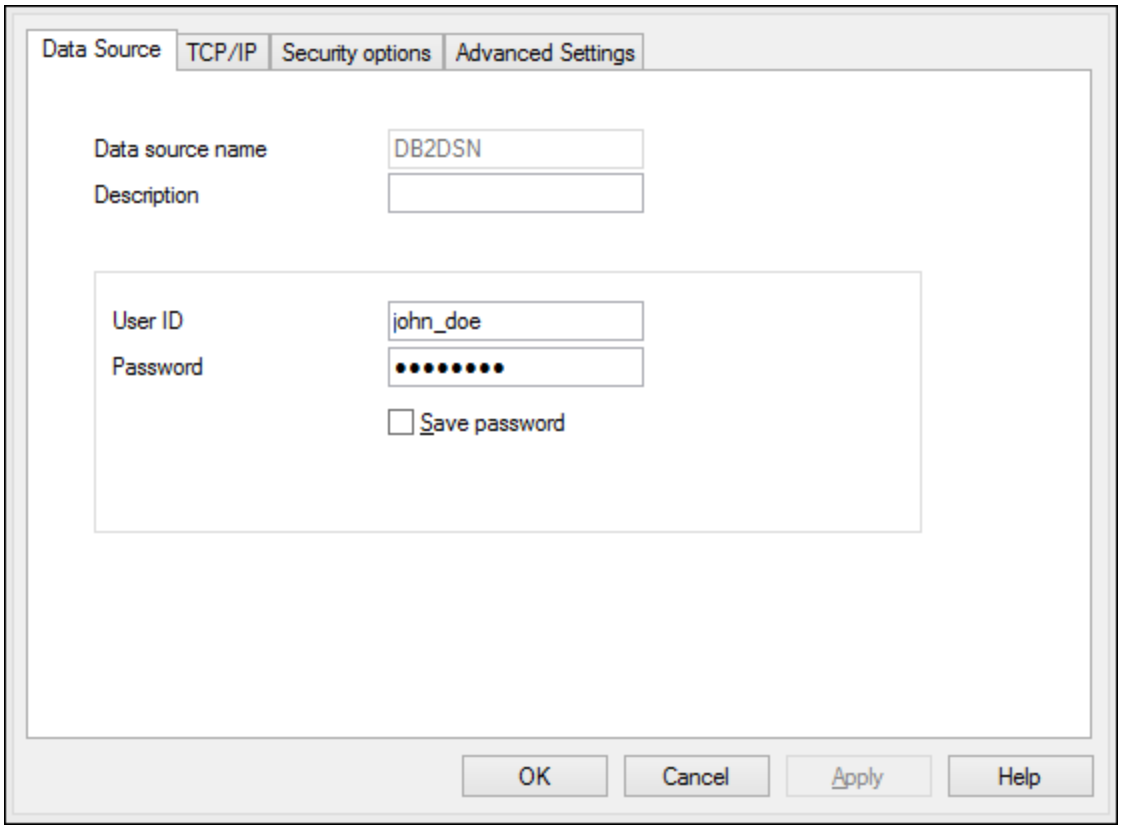


4. Seleccione el controlador de la lista y haga clic en **Conectarse**. (Para editar la lista de controladores disponibles haga clic en **Editar controladores** y active/desactive los controladores que desea agregar o eliminar)

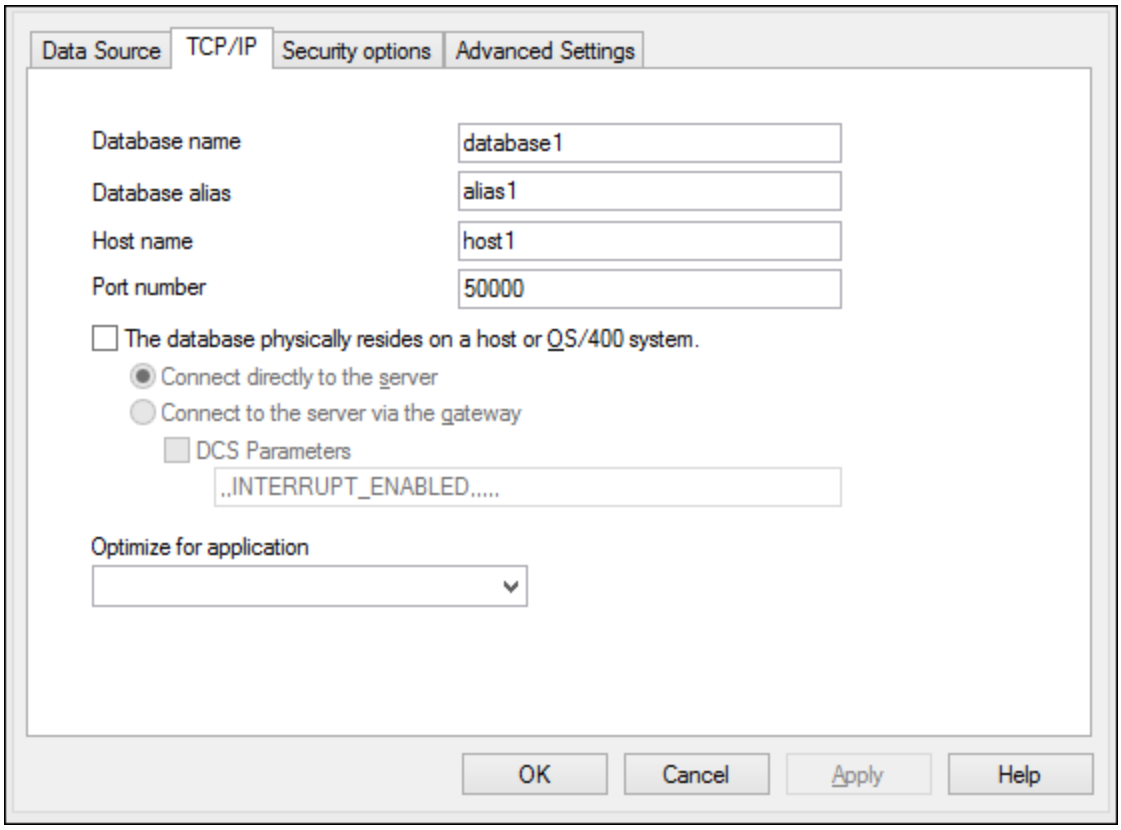
5. Introduzca el DSN (**DB2DSN**) y haga clic en **Agregar**.



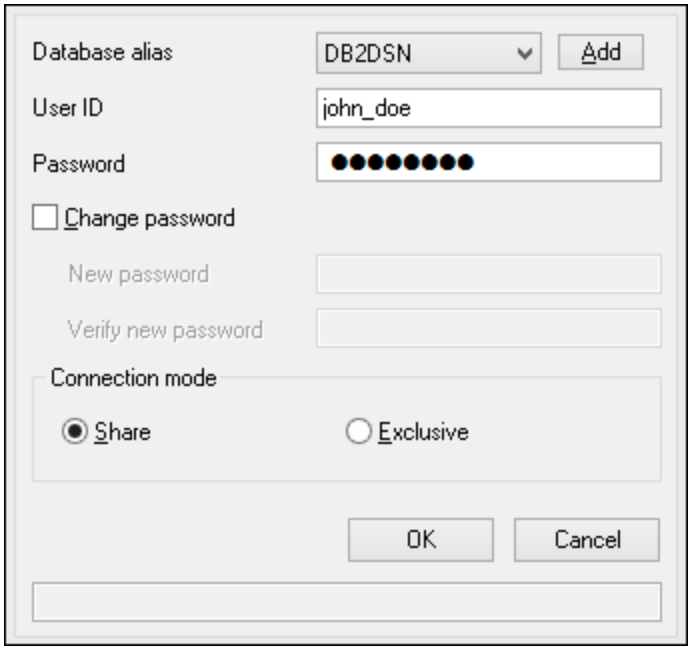
6. En la pestaña *Origen de datos* introduzca el nombre de usuario y la contraseña de la base de datos.



7. En la pestaña **TCP/IP** introduzca el nombre de la base de datos, un nombre para el alias, el nombre de host y el número de puerto. Después haga clic en **Aceptar**.



8. Vuelva a introducir el nombre de usuario y la contraseña y haga clic en **Aceptar**.



10.2.9.5 IBM DB2 para i (JDBC)

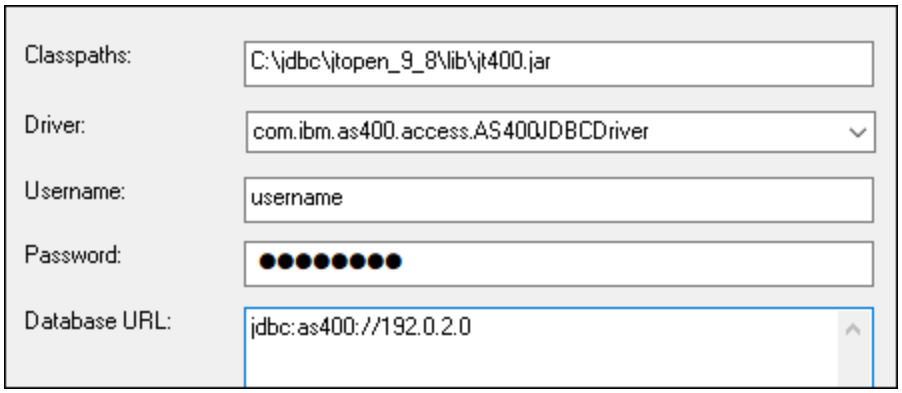
Este ejemplo explica cómo conectarse a una base de datos IBM DB2 for i por JDBC.

Requisitos:

- JRE (Java Runtime Environment) o Java Development Kit (JDK) está instalado. Este último puede ser el JDK de Oracle o uno de código abierto, como Oracle OpenJDK. UModel identifica la ruta de acceso al equipo virtual Java (JVM) a partir de estas ubicaciones, en este orden: a) la ruta personal de acceso al JVM que puede indicar en en las **Opciones** (véase el apartado Java); b) la ruta de acceso al JVM que se encuentra en el registro; c) la variable de entorno `JAVA_HOME`.
- Asegúrese de que la plataforma de UModel (32 o 64 bits) es del mismo tipo que la de JRE/JDK. Este ejemplo usa Oracle OpenJDK 11.0 de 64 bits, por lo que usa la versión de 64 bits de UModel.
- El controlador JDBC (uno o más archivos .jar que permitan conectarse a la BD) debe estar disponible en su sistema operativo. En este ejemplo se usa la biblioteca de código abierto **Toolbox for Java/JTOpen** versión 9.8 (http://jt400.sourceforge.net/). Una vez haya descargado el paquete y lo haya desempaquetado en un directorio local, los archivos .jar necesarios estarán disponibles en el subdirectorio **lib**.
- Necesitará esta información sobre la conexión a la BD: host, nombre de usuario y contraseña.

Para conectarse a IBM DB2 for i con JDBC:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **JDBC Connections**.
3. Junto a "Variables classpath" introduzca la ruta de acceso al archivo .jar que permite conectarse a la BD. En este ejemplo la ruta es **C:\jdbc\jtopen_9_8\jt400.jar**. Tenga en cuenta que puede dejar vacío el campo "Variables classpath" si ha añadido la ruta de acceso al archivo .jar (también pueden ser varios) a la variable de entorno CLASSPATH del sistema operativo (*véase también Configurar la variable CLASSPATH*).
4. En el campo "Controlador" seleccione **com.ibm.as400.access.AS400JDBCDriver**. Esta entrada solo está disponible si se encuentra una ruta de acceso válida a un archivo .jar en el campo "Variables classpath" o en la variable de entorno CLASSPATH del sistema operativo (*véase el paso anterior*).



5. Introduzca el nombre de usuario y la contraseña del usuario de la BD en los campos correspondientes.
6. Introduzca la cadena de conexión JDBC en el campo **URL de la BD**. Asegúrese de reemplazar `host` con el nombre de host o la dirección IP de su servidor de BD.


```
jdbc:as400://host
```

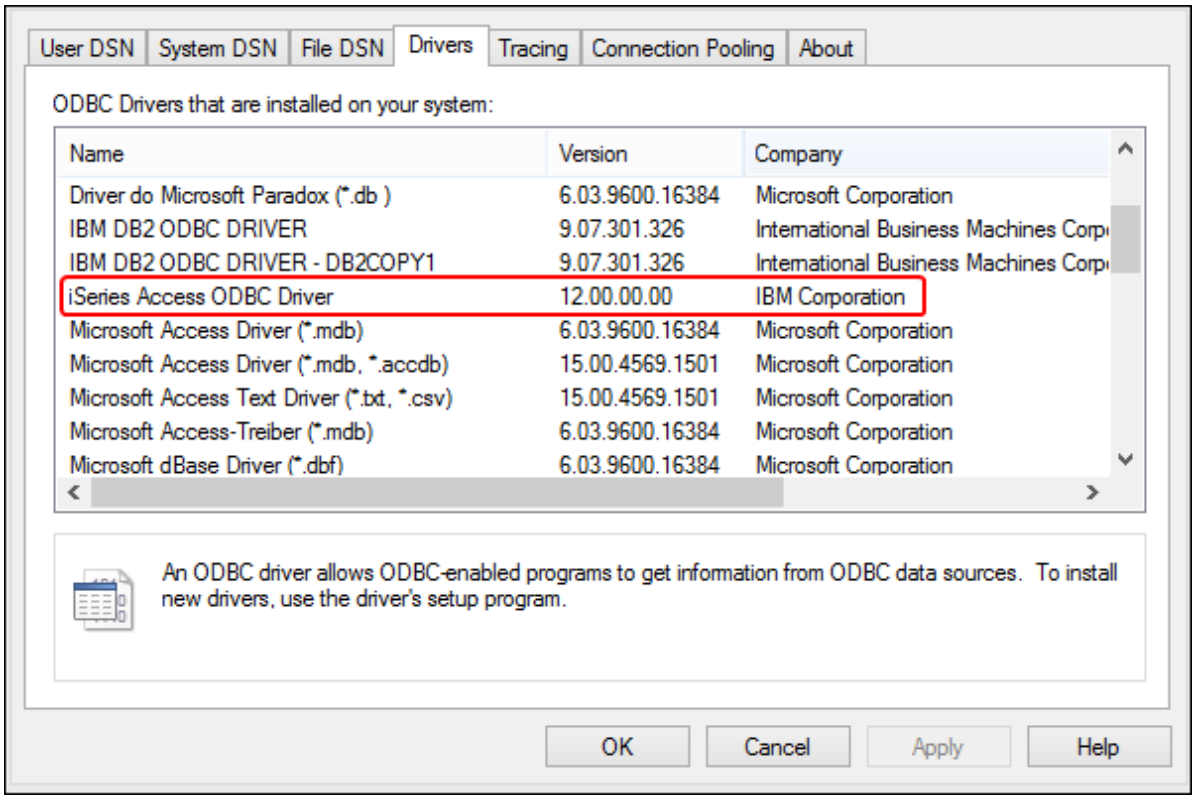
7. Haga clic en **Conectarse**.

10.2.9.6 IBM DB2 para i (ODBC)

Este ejemplo explica cómo conectarse a una base de datos IBM DB2 for i por ODBC.

Requisitos:

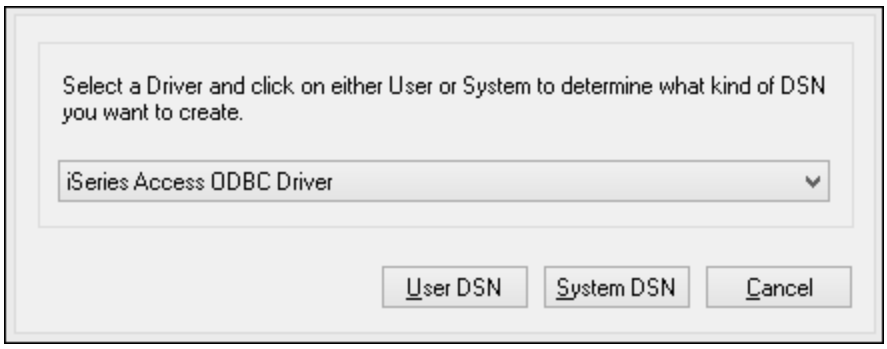
- *IBM System i Access for Windows* está instalado en el sistema (para este ejemplo usamos *IBM System i Access for Windows V6R1M0*). Compruebe que el controlador ODBC está en el equipo (consulte Ver los controladores ODBC disponibles).



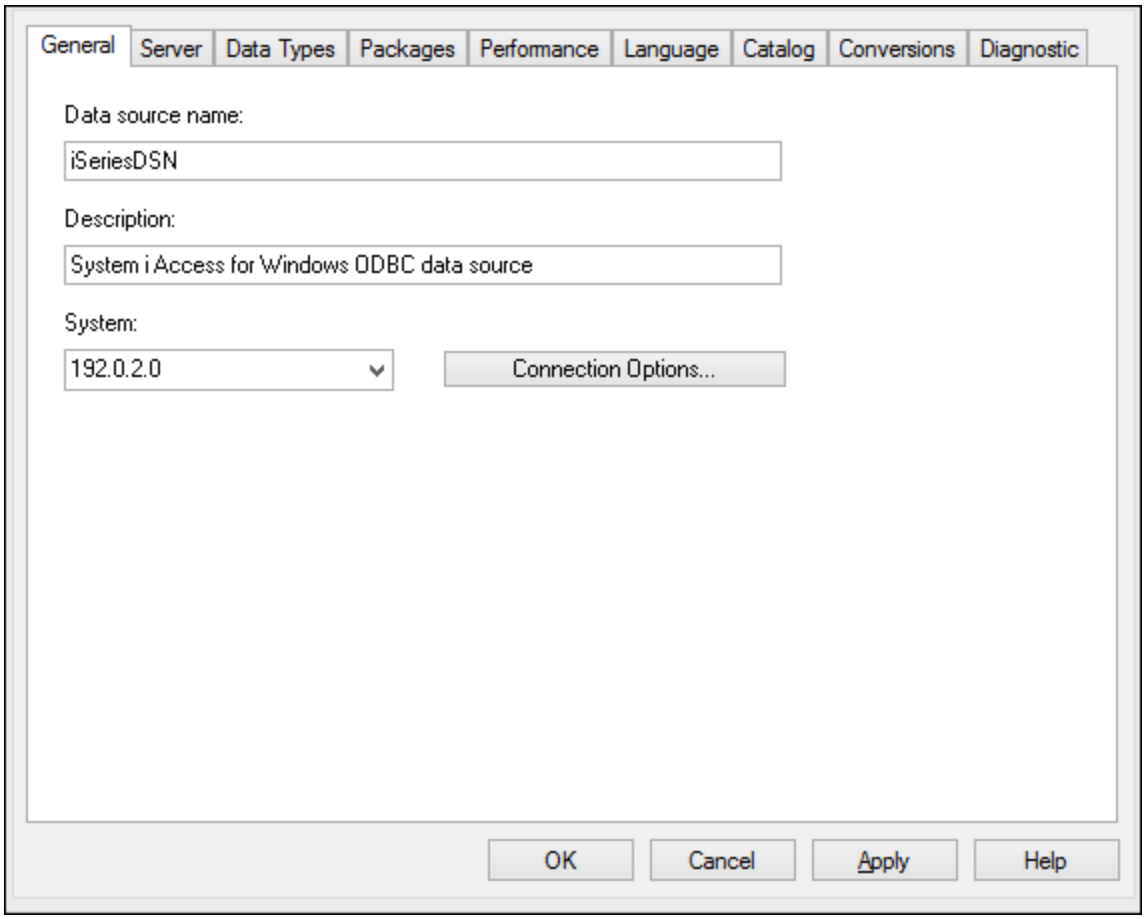
- Disponer de los datos de conexión: dirección IP del servidor de BD, nombre de usuario y contraseña.
- Ejecute *System i Navigator* y siga las instrucciones que aparecen en pantalla para crear una conexión nueva. Llegado el momento de especificar un sistema, introduzca la dirección IP del servidor de BD. Tras crear la conexión se recomienda verificarla (haga clic en la conexión y seleccione **Archivo** > **Diagnóstico** > **Verificar conexión**). Si recibe errores de conexión, póngase en contacto con el administrador del servidor de BD.

Para conectarse a IBM DB2 for i:

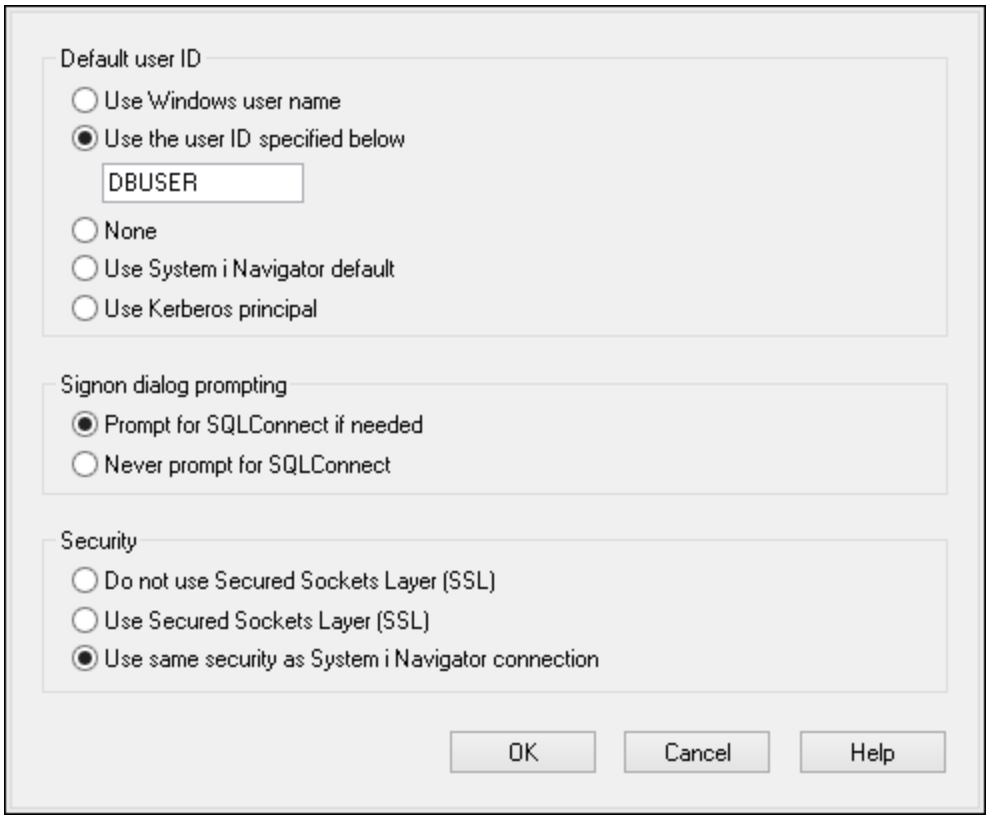
1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones ODBC**.
3. Seleccione *DSN de usuario* (o *DSN de sistema* o *DSN de archivo*).
4. Haga clic en **Agregar** .
5. Seleccione el controlador **iSeries Access ODBC Driver** de la lista y después haga clic en **DSN de usuario** (o **DSN de sistema**).



6. Introduzca el DSN y seleccione la conexión del cuadro combinado *Sistema*. En este ejemplo el DSN es **iSeriesDSN** y el sistema es **192.0.2.0**.



7. Haga clic en el botón **Opciones de conexión** y seleccione *Utilizar el Id. de usuario indicado* y escriba el nombre del usuario de la BD (en este ejemplo es **DBUSER**).



8. Haga clic en **Aceptar**. El origen de datos nuevo aparece ahora en la lista de DSN.
9. Haga clic en **Conectarse**.
10. Introduzca el nombre de usuario y la contraseña de la BD cuando sea necesario y después haga clic en **Aceptar**.

10.2.9.7 IBM Informix (JDBC)

Este ejemplo explica cómo conectarse a una base de datos IBM por JDBC.

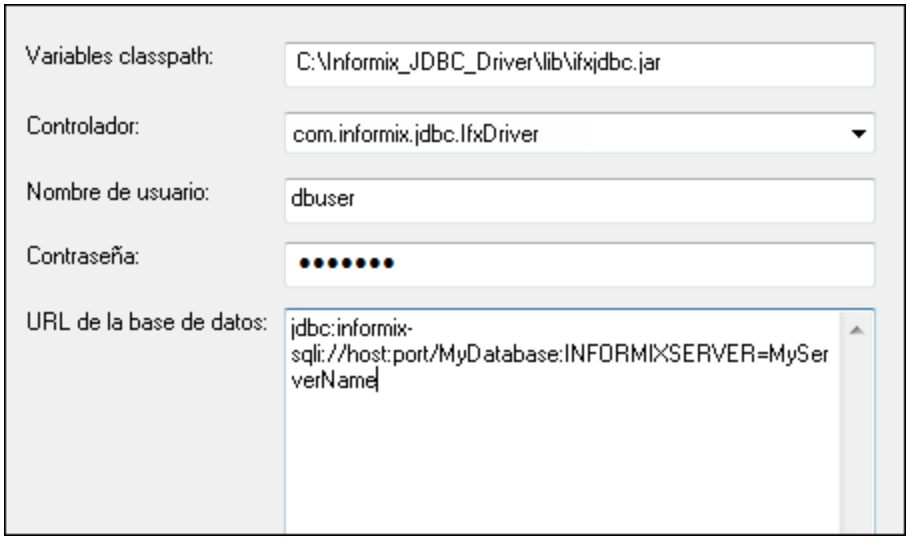
Requisitos:

- Debe tener instalado Java Runtime Environment (JRE) o Java Development Kit (JDK). Este último puede ser el JDK de Oracle o uno de código abierto, como Oracle OpenJDK. UModel identifica la ruta de acceso al equipo virtual Java (JVM) a partir de estas ubicaciones, en este orden: a) la ruta personal de acceso al JVM que puede indicar en en las **Opciones** (véase el apartado Java); b) la ruta de acceso al JVM que se encuentra en el registro; c) la variable de entorno `JAVA_HOME`.
- Asegúrese de que la plataforma de UModel (32 o 64 bits) es la misma que la de JRE/JDK.
- El controlador JDBC (se trata de un archivo .jar o varios archivos .jar que ofrecen conectividad con la base de datos) debe estar disponible en el sistema operativo. En este ejemplo se utiliza el controlador JDBC IBM Informix versión 3.70. Consulte la documentación del controlador para ver las instrucciones de instalación o la guía *IBM Informix JDBC Driver Programmer's Guide*.

- Disponer de los datos de conexión: host, nombre del servidor Informix, base de datos, puerto, nombre de usuario y contraseña.

Para conectarse a IBM Informix por JDBC:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones JDBC**.
3. En el campo *Variables classpath* debe introducir la ruta de acceso del archivo .jar que ofrece conectividad con la base de datos. Si fuera necesario, también puede introducir una lista de rutas de archivo .jar separadas por caracteres de punto y coma. En este ejemplo, el archivo .jar está ubicado en esta ruta de acceso: `C:\Informix_JDBC_Driver\lib\ifxjdbc.jar`. Tenga en cuenta que este campo puede dejarse en blanco si añadió la ruta de acceso de los archivos .jar a la variable de entorno CLASSPATH del sistema operativo (*véase Configurar la variable CLASSPATH*).
4. En el campo *Controlador* seleccione **com.informix.jdbc.IfxDriver**. Recuerde que esta entrada solo estará disponible si se encuentra una ruta de archivo .jar válida en el campo Variables *classpath* o en la variable de entorno CLASSPATH del sistema operativo.



5. Introduzca el nombre de usuario y la contraseña de la base de datos.
6. Introduzca la cadena de conexión para el servidor de BD en el cuadro de texto *URL de la base de datos* (reemplace lo valores resaltados con los de su base de datos).

```
jdbc:informix-sqli://nombreHost:puerto/nombreBD:INFORMIXSERVER=myserver;
```

7. Haga clic en **Conectarse**.

10.2.9.8 MariaDB (ODBC)

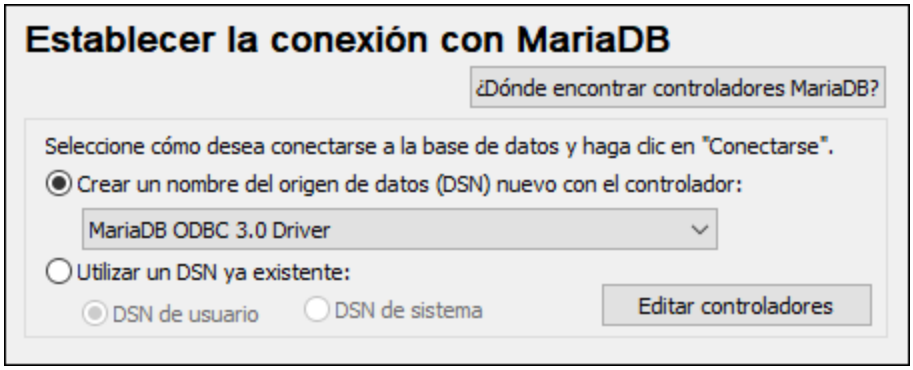
Este ejemplo explica cómo conectarse a un servidor de base de datos MariaDB por ODBC.

Requisitos:

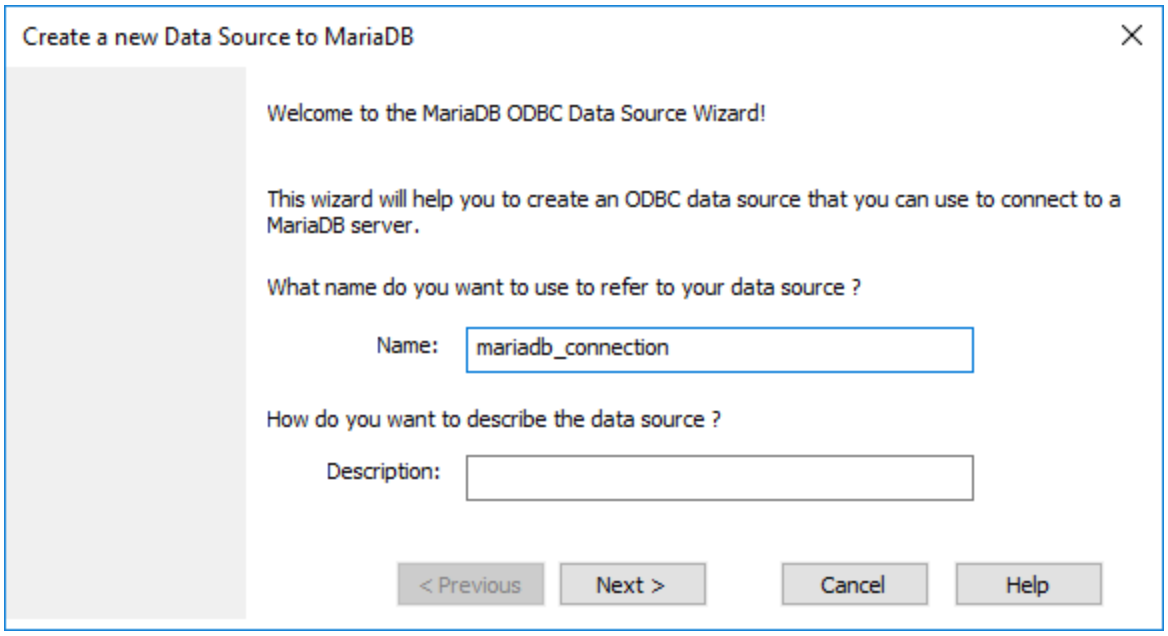
- Debe tener instalado el conector de MariaDB para ODBC (https://downloads.mariadb.org/connector-odbc/).
- Disponer de los datos de conexión: host, base de datos, puerto, nombre de usuario y contraseña.

Para conectarse a MariaDB por ODBC:

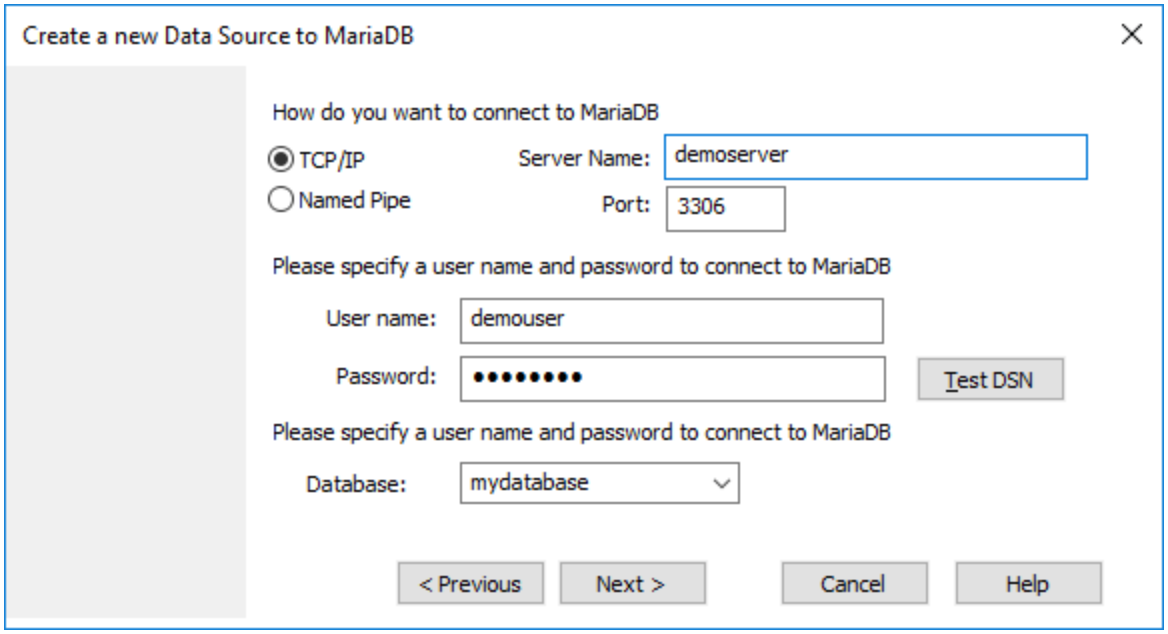
1. Inicie el asistente para la conexión de base de datos.
2. Seleccione *MariaDB (ODBC)* y después haga clic en **Siguiente**.



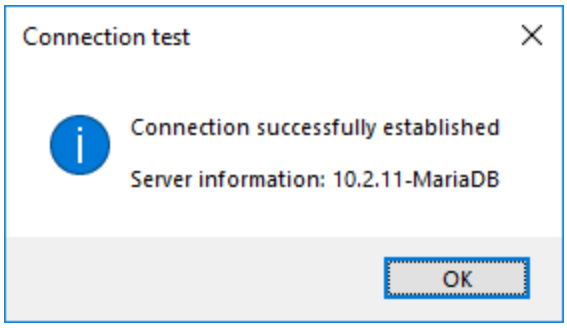
3. Seleccione *Crear un nombre del origen de datos (DSN) nuevo con el controlador* y elija el controlador **MariaDB ODBC 3.0 Driver**. Si este controlador no aparece en la lista, haga clic en **Editar controladores** y seleccione cualquier controlador MariaDB (la lista contiene todos los controladores ODBC que están instalados en el sistema operativo).
4. Haga clic en **Conectarse**.



5. Introduzca el nombre y, si quiere, una descripción que le ayude a identificar este origen de datos ODBC más adelante.



6. Rellene las credenciales de la conexión de base de datos (servidor TCP/IP, usuario, contraseña), seleccione una base de datos y después haga clic en **Probar DSN**. Cuando se establezca la conexión aparecerá este mensaje:



7. Haga clic en **Siguiente** y siga los pasos del asistente hasta el final. Dependiendo del caso, puede que sean necesarios más parámetros (p. ej. certificados SSL si se conecta a MariaDB a través de una conexión segura).

Nota: si el servidor de base de datos es remoto, deberá estar configurado por el administrador para que acepte conexiones remotas desde la dirección IP de su equipo.

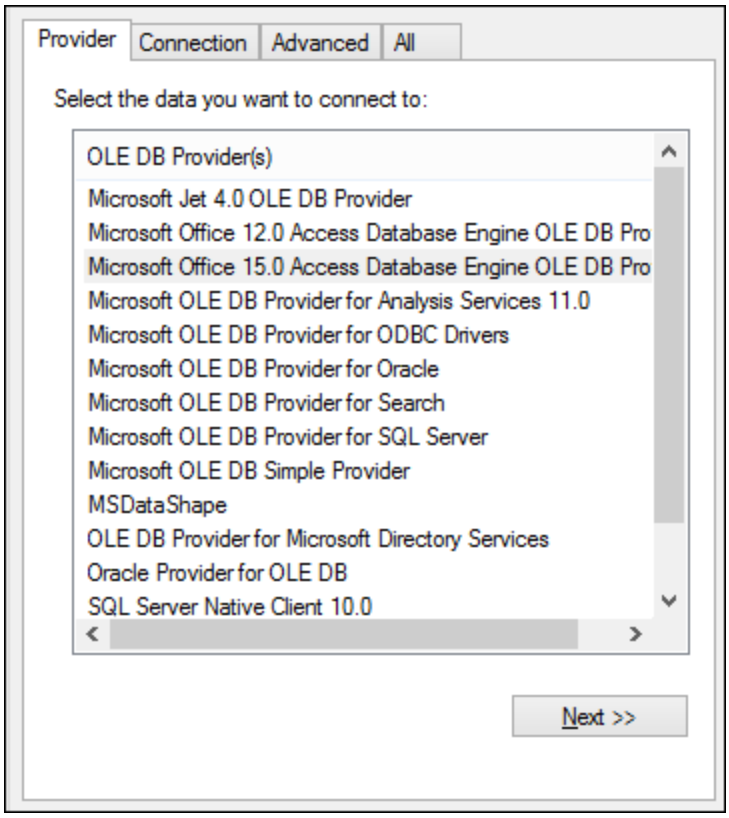
10.2.9.9 Microsoft Access (ADO)

Lo más sencillo para conectarse a una base de datos Microsoft Access es seguir las instrucciones del asistente para la conexión de base de datos y buscar el archivo de base de datos, como se muestra en el apartado Conectarse a una base de datos Microsoft Access. También puede configurar explícitamente una conexión ADO, como muestra este ejemplo. Esta segunda opción se recomienda si la base de datos está protegida con contraseña.

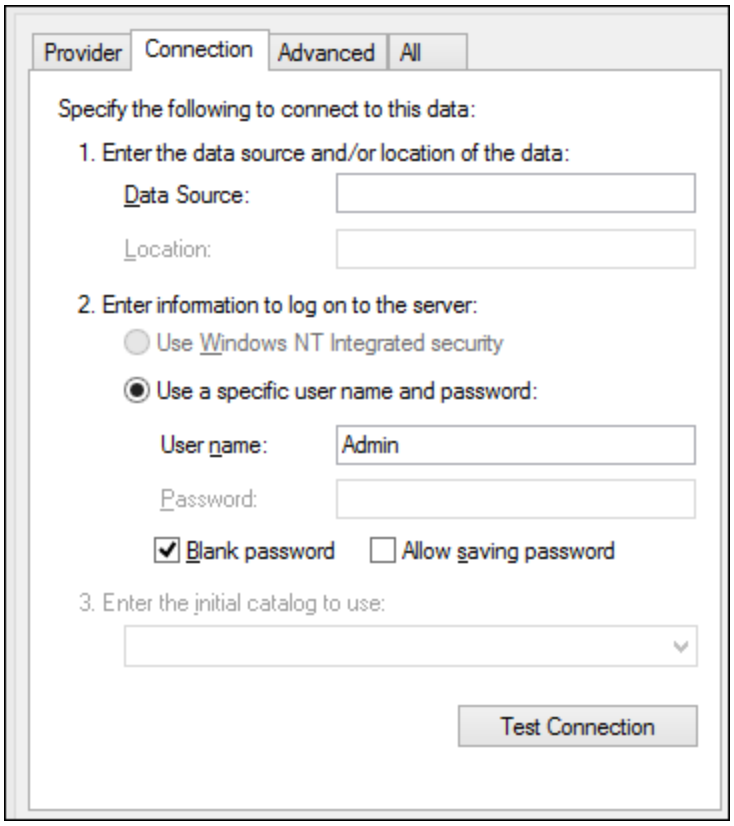
También puede conectarse a Microsoft Access por ODBC, pero esto implica algunas restricciones por lo que recomendamos evitar este tipo de conexión.

Para conectarse a una base de datos Microsoft Access protegida con contraseña:

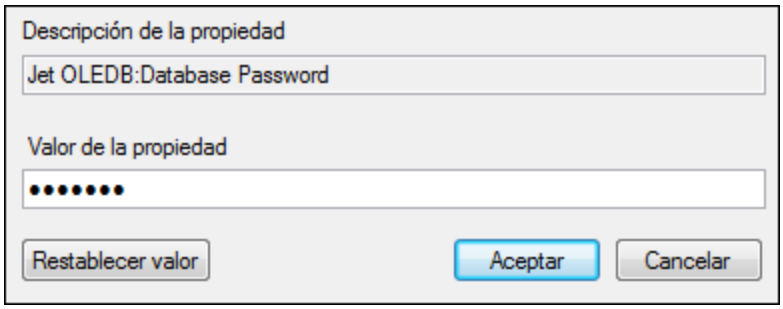
1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones ADO**.
3. Ahora haga clic en **Generar**.



4. Seleccione **Microsoft Office 15.0 Access Database Engine OLE DB Provider** y haga clic en **Siguiente**.



5. En el cuadro de texto Origen de datos introduzca la ruta de acceso del archivo de Microsoft Access en formato UNC, por ejemplo, `\\myserver\\mynetworkshare\Reports\Revenue.accdb`, donde `myserver` es el nombre del servidor y `mynetworkshare` el nombre del recurso compartido de red.
6. En la pestaña *Todos* haga doble clic en la propiedad **Jet OLEDB:Database Password** e introduzca la contraseña de la base de datos.



Nota: si no es capaz de establecer la conexión, busque el archivo de información del grupo de trabajo (`System.MDW`) de su perfil de usuario y establezca el valor de la propiedad **Jet OLEDB: System database** en la ruta de acceso del archivo `System.MDW`.

10.2.9.10 Microsoft SQL Server (ADO)

En este ejemplo explicamos cómo conectarse a una base de datos SQL Server con ADO. Estas instrucciones asumen que usa el driver recomendado para SQL Server **Microsoft OLE DB Driver for SQL Server**

(MSOLEDBSQL), que puede descargar en https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15.

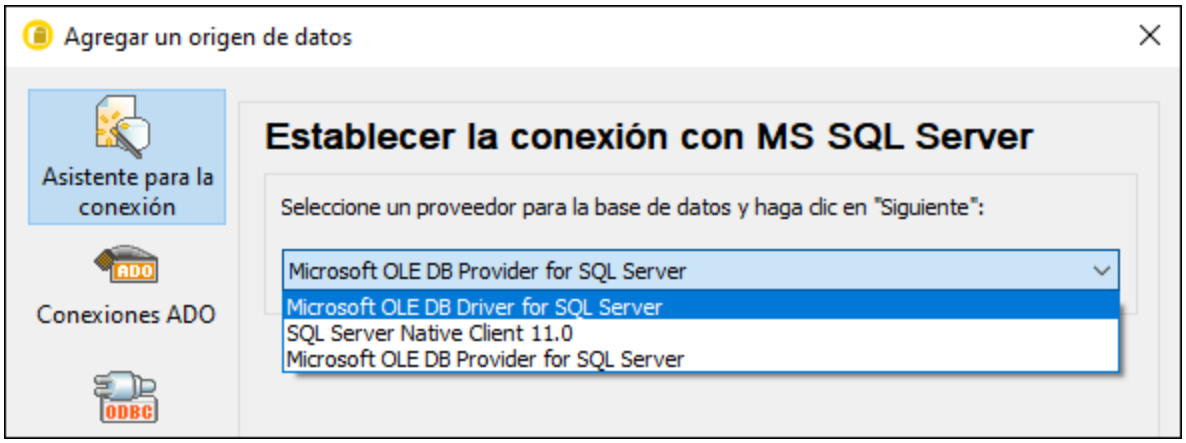
Antes se seguir estas instrucciones, asegúrese de que descarga e instala el proveedor que mencionado más arriba en su equipo de trabajo. El proveedor ADO debe ser de la misma versión de plataforma que UModel (32 bits o 64 bits).

Si quiere usar otro proveedor ADO, como **QL Server Native Client (SQLNCLI)** or **Microsoft OLE DB Provider for SQL Server (SQLOLEDB)**, las instrucciones son parecidas; sin embargo, estos proveedores están obsoletos, por lo que no se recomiendan. Además, para poder conectarse correctamente con un proveedor obsoleto puede que necesite configurar otras propiedades de conexión, como se describe en Configurar las propiedades de vínculo de datos de SQL Server.

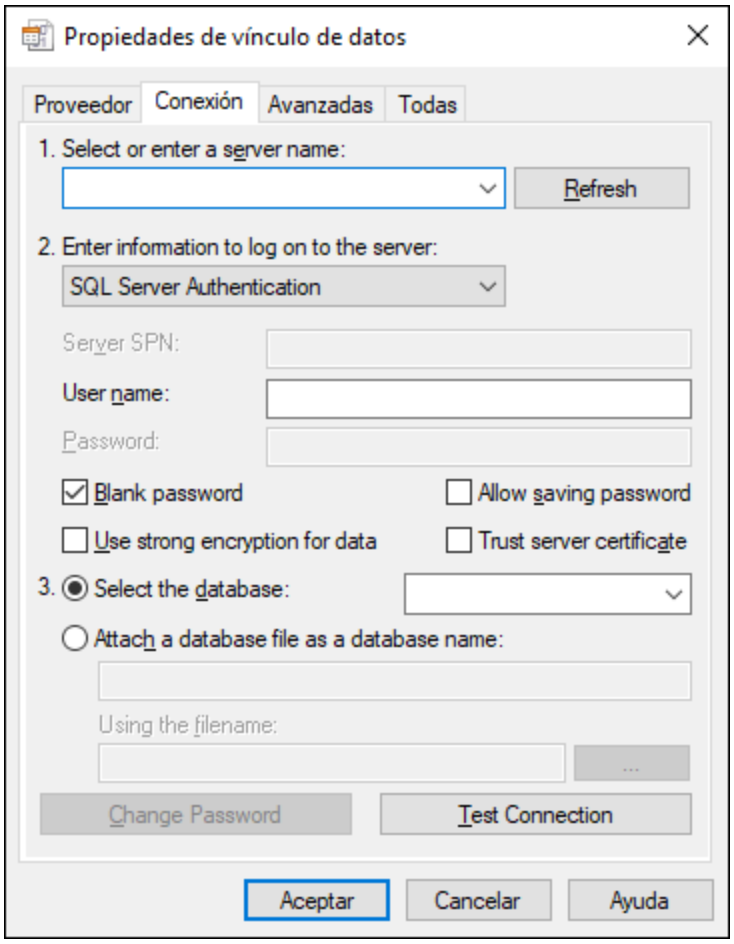
Es un problema conocido que el proveedor de BD para SQL Server **Microsoft OLE DB Provider for SQL Server (SQLOLEDB)** tiene dificultades para enlazar parámetros de consultas complejas como las expresiones comunes de tabla (CTE) e instrucciones SELECT anidadas.

Para conectarse a SQL Server:

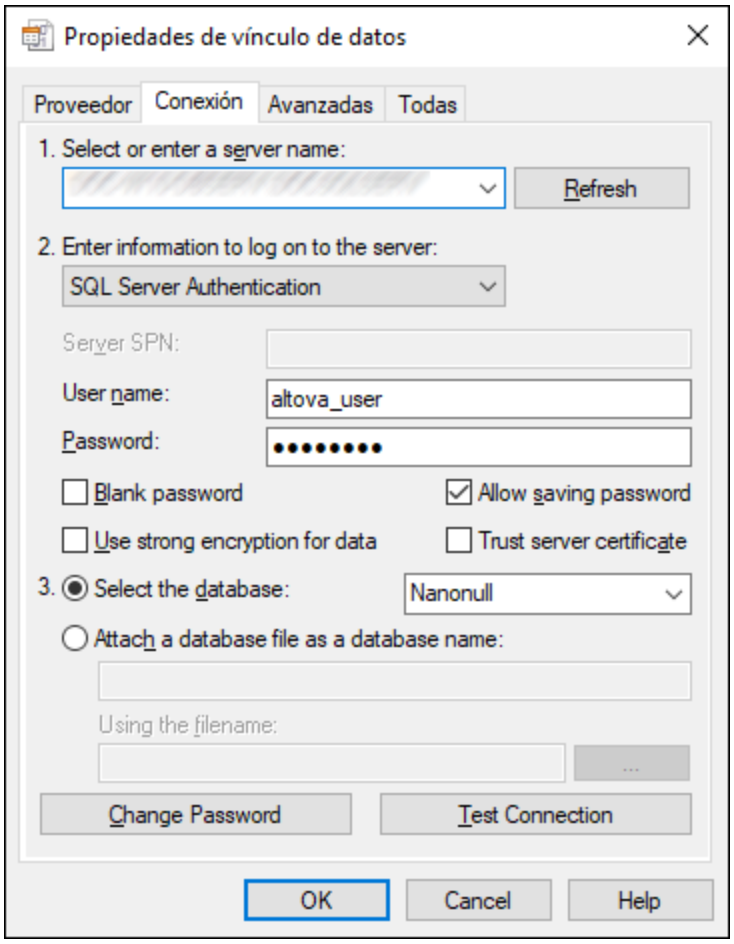
1. Inicie el asistente de conexión.
2. Seleccione **Microsoft SQL Server (ADO)** y después haga clic en **Siguiente**. Verá que aparece la lista de proveedores ADO disponibles. En este ejemplo usaremos **Microsoft OLE DB Driver for SQL Server**. Si no está en la lista, asegúrese de que está instalado en su equipo, como hemos mencionado antes.



3. Haga clic en **Siguiente**. Se abre el cuadro de diálogo "Propiedades de enlace de datos".



4. Seleccione o introduzca el nombre del servidor de BD, por ejemplo **SQLSERV01**. Si se conecta a una instancia de SQL Server con nombre, el nombre del servidor se parecerá a: **SQLSERV01\SOMEINSTANCE**.
5. Si configuró el servidor de BD para que permita conexiones de usuarios autenticados en el dominio Windows, seleccione **Autenticación Windows**. De lo contrario seleccione **Autenticación SQL Server**, desmarque la casilla *Contraseña en blanco* e introduzca las credenciales de BD en los campos correspondientes.
6. Marque la casilla *Permitir guardar contraseña* y seleccione la BD a la que se quiere conectar (en este ejemplo, "Nanonull").



7. Para comprobar la conexión haga clic en **Comprobar conexión**. Este paso es opcional, pero recomendamos no saltárselo.
8. Haga clic en **Aceptar**.

10.2.9.11 Microsoft SQL Server (ODBC)

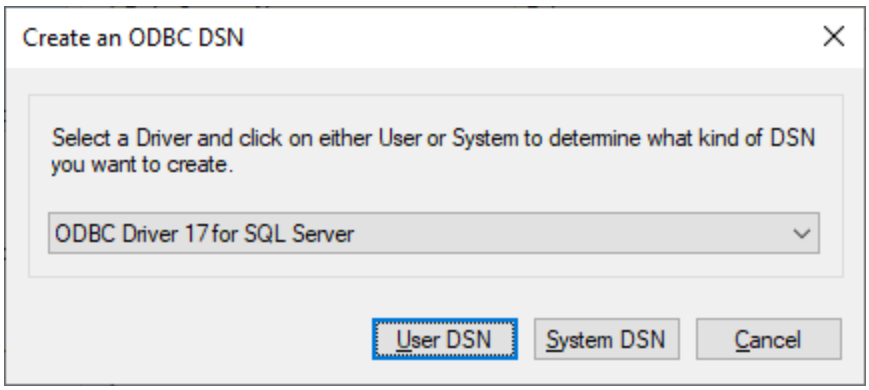
En este ejemplo aprenderá a conectar una base de datos a SQL Server con ODBC.

Requisitos previos:

- Descargue e instale el controlador **Microsoft ODBC Driver for SQL Server**, que encontrará en el sitio web de Microsoft (*véase* *https://docs.microsoft.com/en-us/SQL/connect/odbc/download-odbc-driver-for-sql-server*).Este ejemplo usa **Microsoft ODBC Driver 17 for SQL Server** para conectarse a la base de datos **SQL Server 2016**. Puede descargar un controlador distinto en función de la versión de SQL Server a la que se quiera conectar. Para información sobre las versiones del controlador ODBC compatibles con su base de datos SQL Server consulte los requisitos del sistema del controlador.

Para conectarse a SQL Server con ODBC:

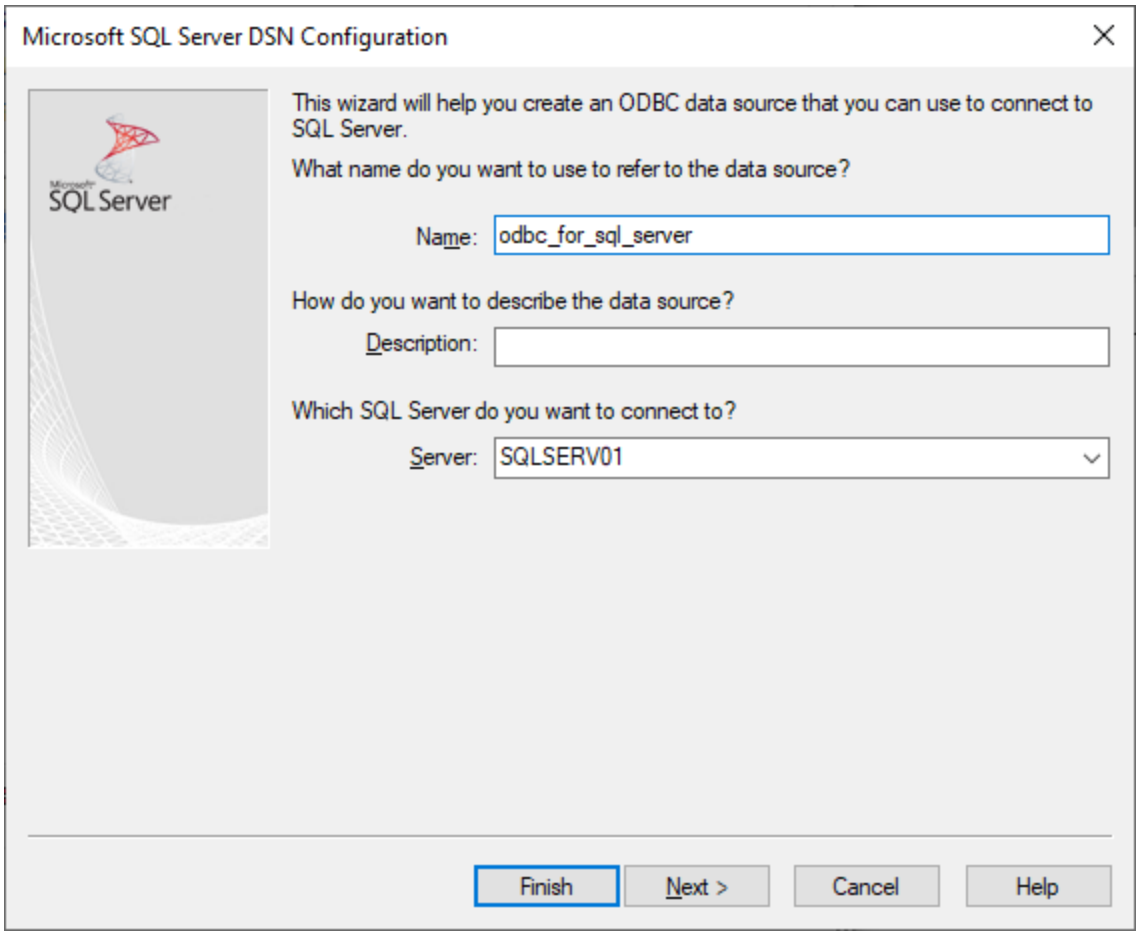
1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones ODBC**.
3. Seleccione **DSN de usuario** (o **DSN de sistema** si tiene privilegios de administrador) y haga clic en **Agregar** .
4. Seleccione el controlador de la lista. Tenga en cuenta que este solo aparece en esta lista una vez se ha instalado.



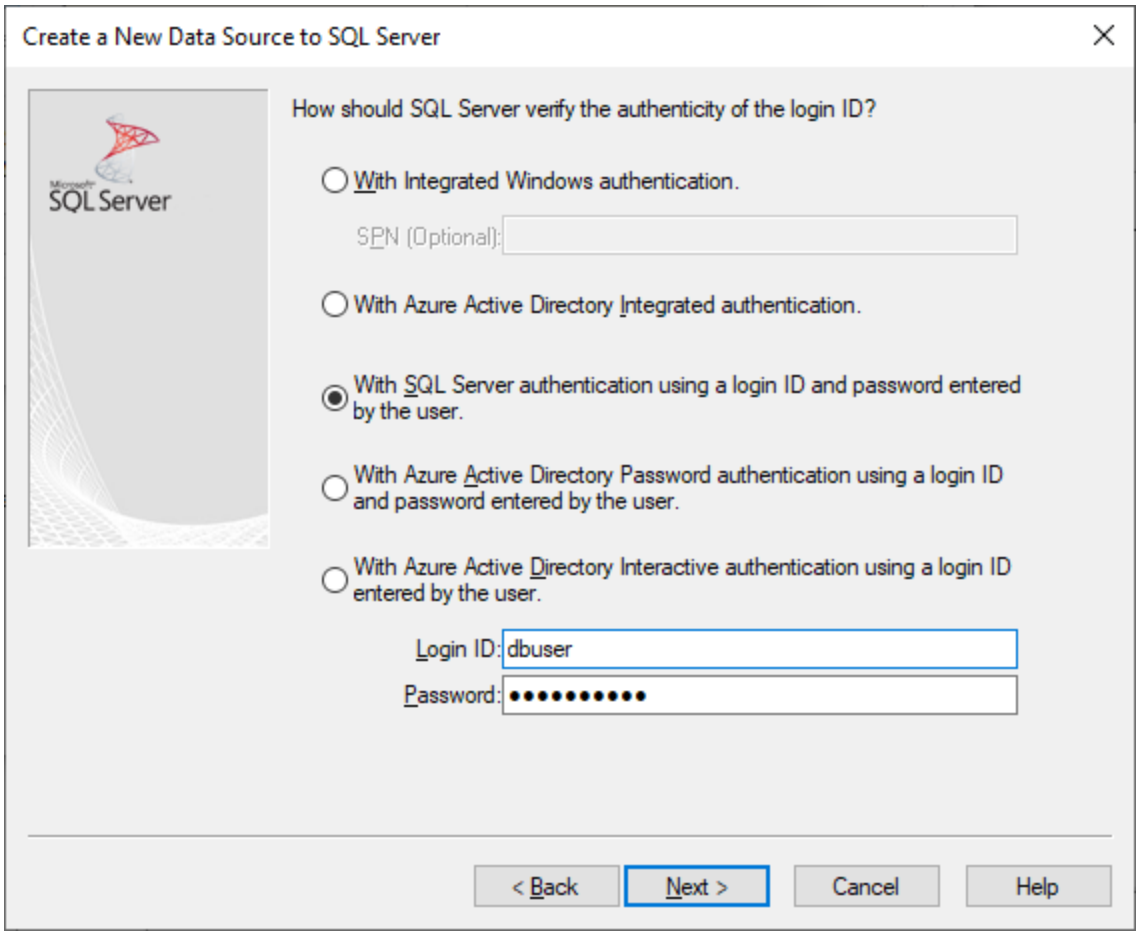
5. Haga clic en **DSN de usuario** (o en **DSN de sistema** si está creando un sistema DNS).

> Para crear un **Sistema DSN** necesita ejecutar UModel como administrador. Si lo que quiere es crear este sistema, salga del asistente y vuelva a ejecutar UModel como administrador y siga los pasos desde el principio.

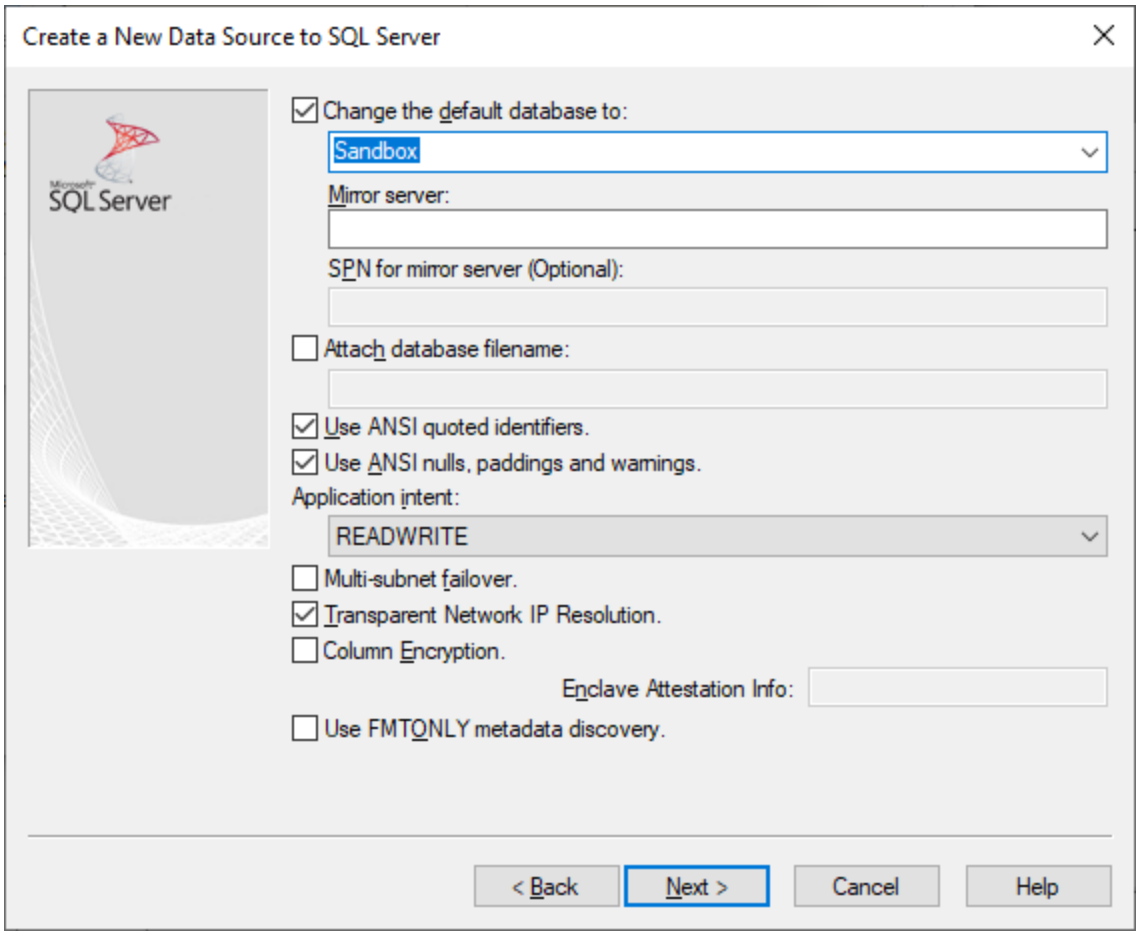
6. Introduzca un nombre y, si quiere, una descripción que identifique esta conexión. A continuación seleccione en la lista el SQL Server al que se quiere conectar (en este ejemplo, **SQLSERV01**).



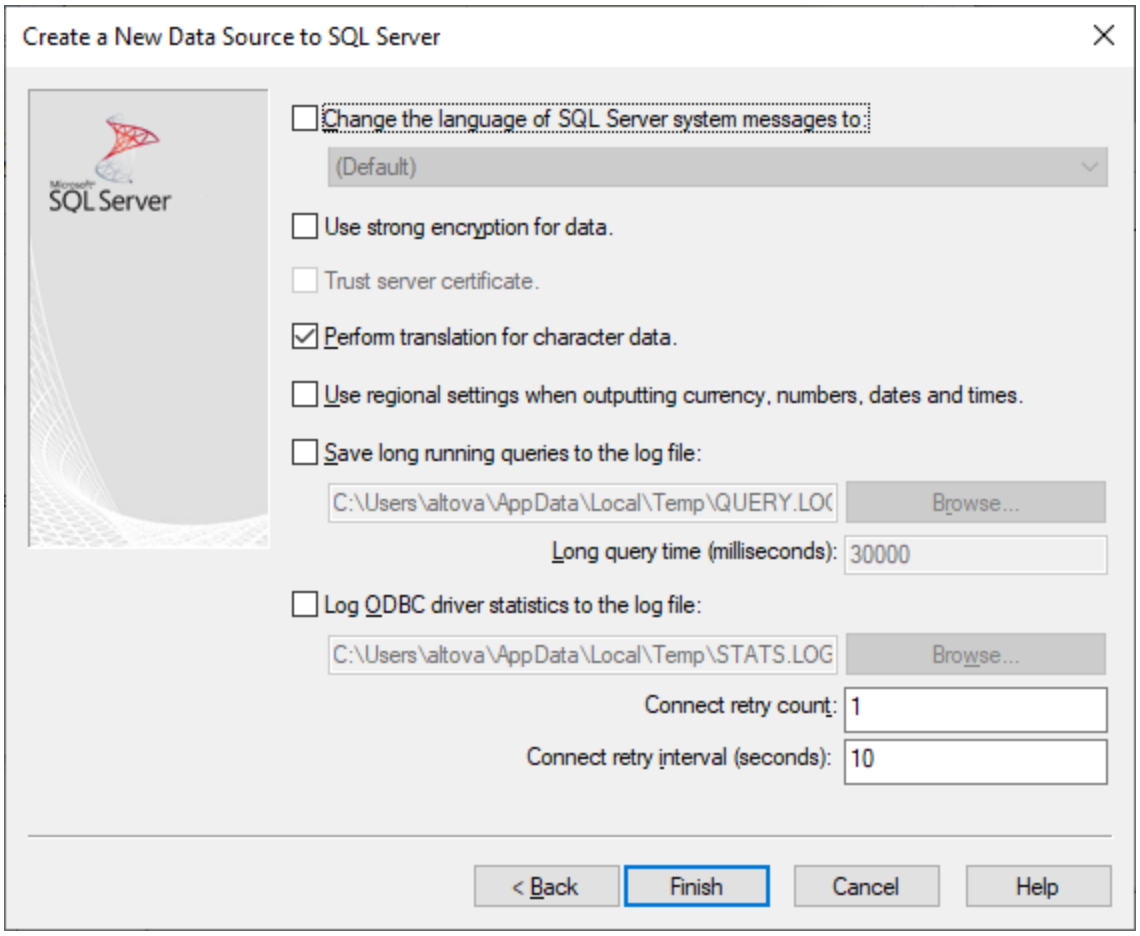
7. Si el servidor de BD se configuró para que permita conexiones de usuarios autenticados en el dominio Windows, seleccione **Con autenticación integrada de Windows**. De lo contrario, seleccione la opción que necesite. En este ejemplo usamos **Con autenticación de SQL Server...**, que necesita que se introduzcan el nombre de usuario y la contraseña en los campos correspondientes.



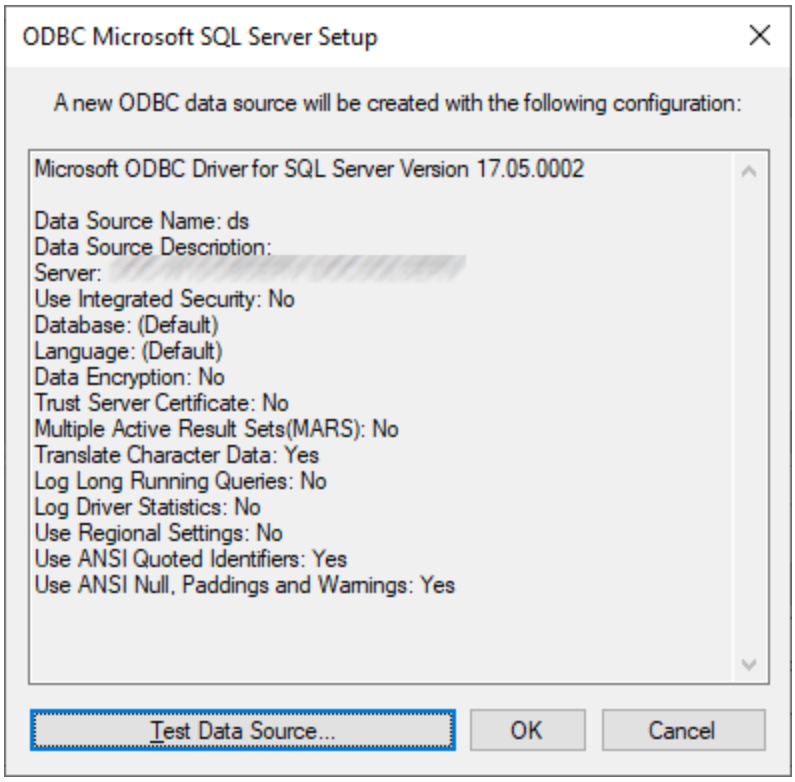
8. También puede marcar la casilla **Cambiar la base de datos predeterminada a** e introducir el nombre de la base de datos a la que se va a conectar (en este ejemplo, **Sandbox**).



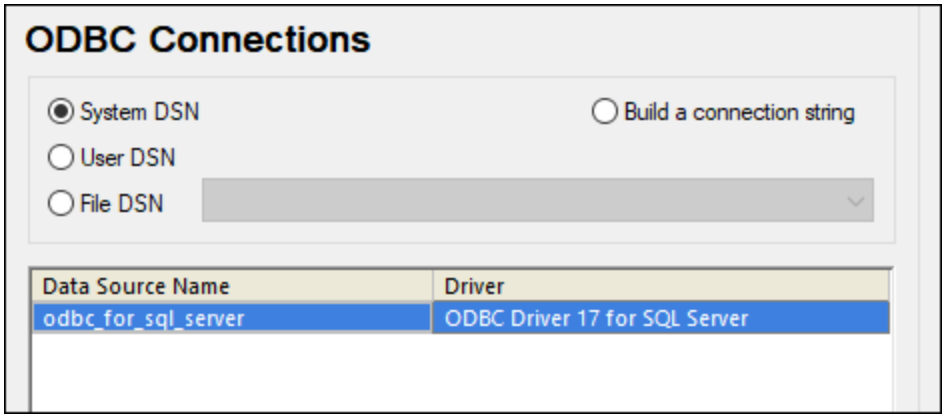
9. Haga clic en **Siguiente** y, si quiere, configure el resto de los parámetros de esta conexión.



10. Haga clic en **Finalizar**. Aparece un cuadro de diálogo de conformación que contiene los detalles de la conexión.



11. Haga clic en **Aceptar**. Ahora la fuente de datos aparece en la lista de fuentes de datos de **Usuario** o **Sistema**, según la configuración; por ejemplo:



10.2.9.12 MySQL (ODBC)

Este ejemplo explica cómo conectarse a un servidor de base de datos MySQL desde un equipo Windows con el controlador ODBC. El controlador ODBC MySQL no está disponible en Windows así que deberá descargarlo e instalarlo por separado. En este ejemplo usamos el conector MySQL/ODBC 8.0.

Requisitos previos:

- El controlador MySQL ODBC está instalado en el sistema. Consulte la documentación de MySQL para conocer qué controlador debe usar para su versión del servidor de base de datos (*véase https://dev.mysql.com/downloads/connector/odbc/*).
- Disponer de los datos de conexión: host, base de datos, puerto, nombre de usuario y contraseña.

Conector MySQL/ODBC 8.0
Si instala el controlador MySQL ODBC para plataformas de 64 bits, asegúrese de que también instala la versión de UModel para plataformas de 64 bits.

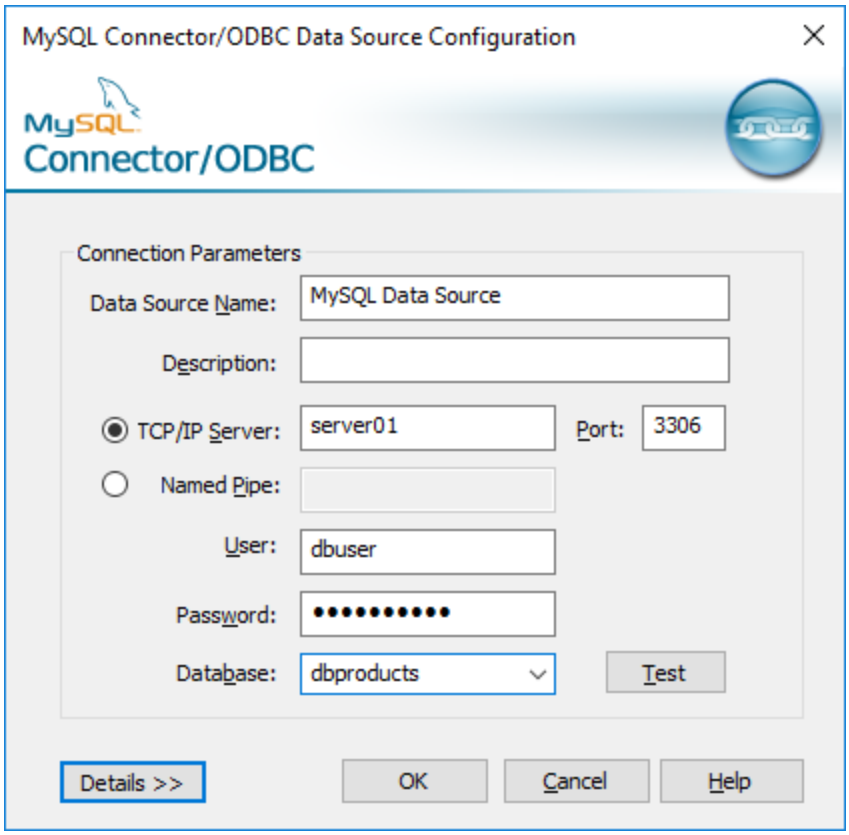
Para conectarse a MySQL por ODBC:

1. Inicie el asistente para la conexión de base de datos.
2. Seleccione la opción *MySQL (ODBC)* y después haga clic en **Siguiente**.

3. Seleccione la opción *Crear un DNS nuevo con el controlador* y seleccione un controlador MySQL. Si no hay ningún controlador MySQL en la lista, haga clic en **Editar controladores** y seleccione uno (la lista contiene todos los controladores ODBC que están instalados en el sistema).

   Si instaló la versión de UModel para plataformas de 64 bits se mostrarán los controladores ODBC en la lista. De lo contrario se mostrarán los controladores para la versión de 32 bits. Consulte también el apartado Ver los controladores ODBC disponibles.

4. Haga clic en **Conectarse**.



5. En el cuadro de texto *Nombre del origen de datos* introduzca un nombre que le ayude a identificar este origen de datos ODBC más adelante.
6. Rellene las credenciales de la conexión de BD (servidor TCP/IP, usuario, contraseña), seleccione una base de datos y haga clic en **Aceptar**.

Nota: si el servidor de BD es remoto, el administrador del servidor debe configurarlo para que acepte conexiones remotas desde la dirección IP de su equipo. Además, si hace clic en **Detalles>>**, podrá configurar algunos parámetros más. Consulte la documentación del controlador antes de cambiar los valores predeterminados.

10.2.9.13 Oracle (JDBC)

Este ejemplo explica cómo conectarse a un servidor de base de datos Oracle desde un equipo cliente, usando la interfaz JDBC. La conexión se crea como una conexión Java pura, usando el paquete **Oracle Instant Client Package (Basic)** que se puede descargar del sitio web de Oracle. La ventaja de este tipo de conexión es que solamente exige el entorno Java y las bibliotecas .jar que vienen con el paquete Oracle Instant Client Package. Es decir, no es necesario instalar ni configurar clientes de base de datos más complejos.

Requisitos:

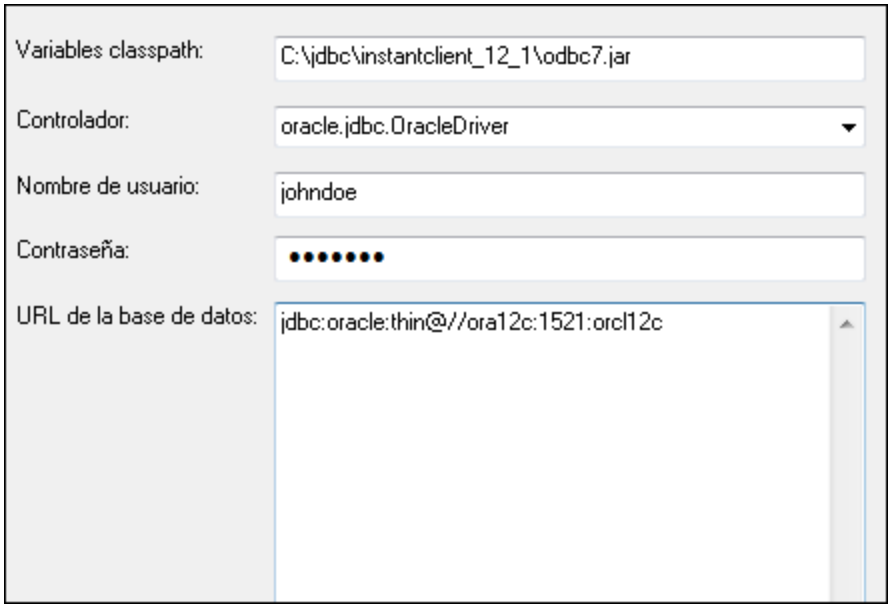
- JRE (Java Runtime Environment) o Java Development Kit (JDK) está instalado. Este último puede ser el JDK de Oracle o uno de código abierto, como Oracle OpenJDK. UModel identifica la ruta de acceso

al equipo virtual Java (JVM) a partir de estas ubicaciones, en este orden: a) la ruta personal de acceso al JVM que puede indicar en en las **Opciones** (véase el apartado Java); b) la ruta de acceso al JVM que se encuentra en el registro; c) la variable de entorno `JAVA_HOME`.

- Asegúrese de que la plataforma de UModel (32 o 64 bits) es la misma que la de JRE/JDK.
- El paquete **Oracle Instant Client Package (Basic)** está disponible en el sistema operativo. El paquete se puede descargar del sitio web oficial de Oracle. En este ejemplo utilizamos la versión 12.1.0.2.0 para Windows de 32 bits y, por tanto, Oracle JDK de 32 bits.
- Disponer de los datos de conexión: host, puerto, nombre del servicio, nombre de usuario y contraseña.

Para conectarse a Oracle a través del paquete Instant Client Package:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones JDBC**.
3. En el campo *Variables classpath* debe introducir la ruta de acceso del archivo .jar que ofrece conectividad con la base de datos. Si fuera necesario, también puede introducir una lista de rutas de archivo .jar separadas por caracteres de punto y coma. En este ejemplo, el archivo .jar está ubicado en esta ruta de acceso: `C:\jdbc\instantclient_12_1\ojdbc7.jar`. Tenga en cuenta que este campo puede dejarse en blanco si añadió la ruta de acceso de los archivos .jar a la variable de entorno CLASSPATH del sistema operativo (*véase Configurar la variable CLASSPATH*).
4. En el campo *Controlador* seleccione **oracle.jdbc.OracleDriver** o **oracle.jdbc.driver.OracleDriver**. Recuerde que esta entrada solo estará disponible si se encuentra una ruta de archivo .jar válida en el campo Variables *classpath* o en la variable de entorno CLASSPATH del sistema operativo.
5. Introduzca el nombre de usuario y contraseña de la base de datos.



6. Introduzca la cadena de conexión para el servidor de BD en el cuadro de texto *URL de la base de datos* (reemplace lo valores resaltados con los de su servidor de base de datos).

```
jdbc:oracle:thin:@//host:puerto:servicio
```

7. Haga clic en **Conectarse**.

10.2.9.14 Oracle (ODBC)

Este ejemplo explica cómo conectarse a un servidor de base de datos Oracle desde UModel. En este ejemplo la base de datos Oracle está en un equipo de la red y la conexión se establece a través de un cliente de base de datos Oracle instalado en el equipo local.

En este ejemplo ofrecemos instrucciones para configurar un DSN ODBC con el asistente para la conexión de base de datos de UModel. Si ya tiene un DSN o prefiere crear uno desde el administrador de orígenes de datos ODBC de Windows, puede seleccionarlo desde el asistente. Para más información consulte el apartado Conexiones ODBC.

Requisitos:

- El cliente de BD Oracle (que incluye el controlador ODBC Oracle) está instalado y configurado en el sistema. Para más información consulte la documentación del software de Oracle.
- El archivo `tnsnames.ora` ubicado en el directorio de inicio de Oracle contiene una entrada que describe los parámetros de conexión de la base de datos:

```
ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = server01)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = orcl)
      (SERVER = DEDICATED)
    )
  )
```

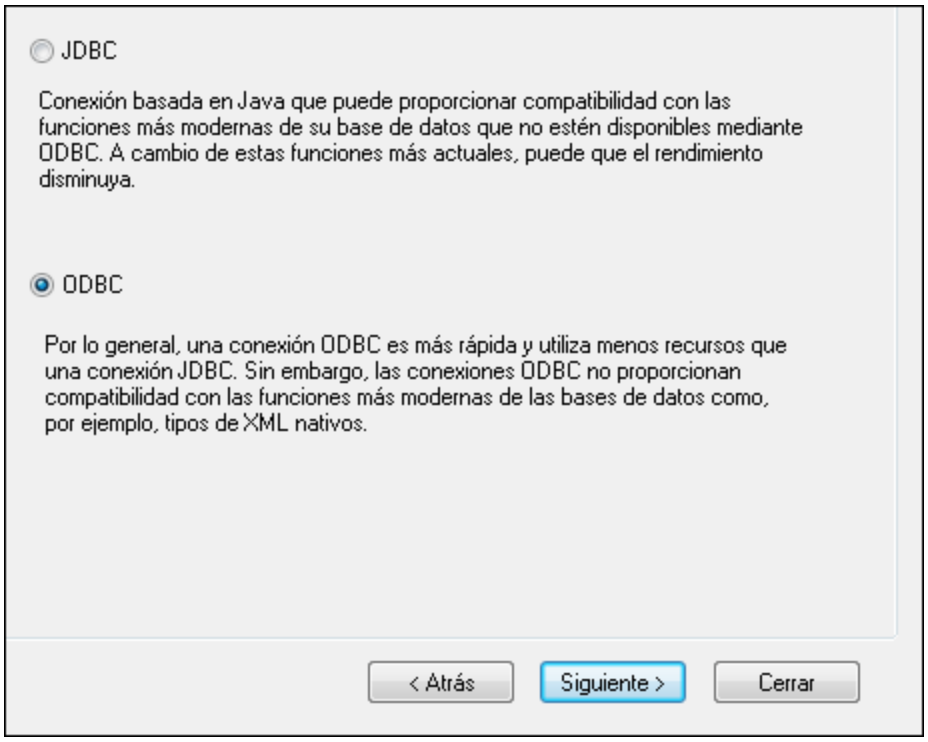
La ruta de acceso del archivo `tnsnames.ora` depende de la ubicación del directorio de inicio de Oracle. Por ejemplo, en el caso del cliente de base de datos Oracle 11.2.0, la ruta de acceso predeterminada del directorio de inicio podría ser:

```
C:\app\nombreUsuario\product\11.2.0\client_1\network\admin\tnsnames.ora
```

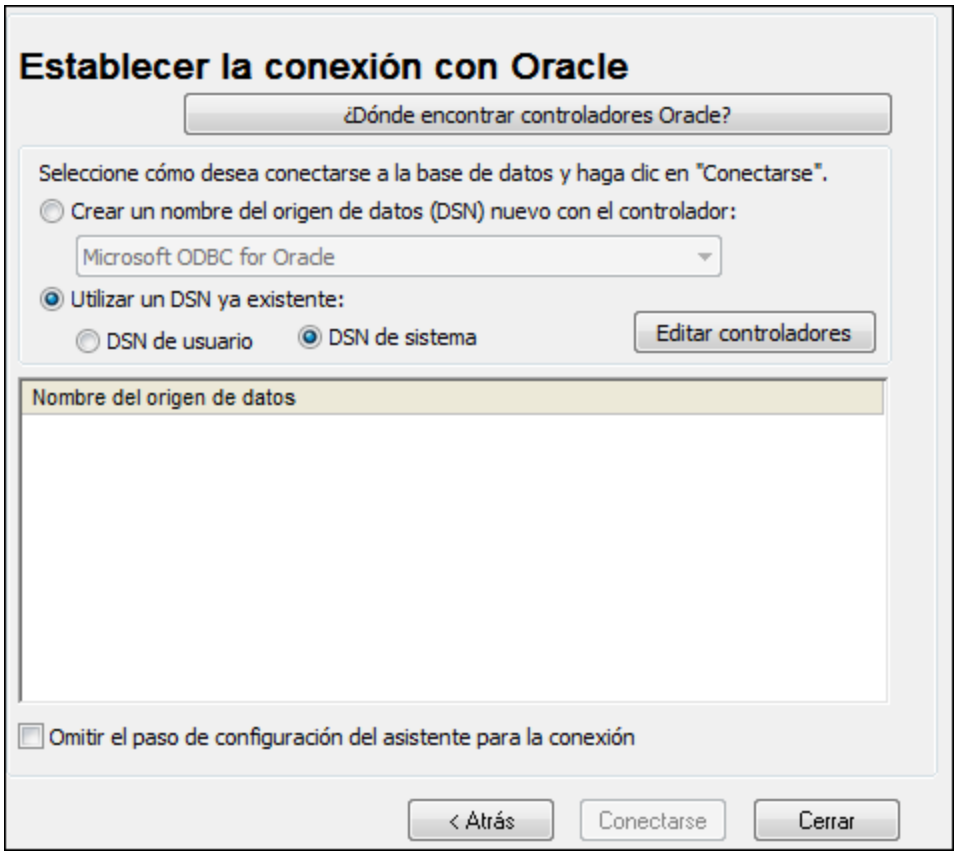
En el archivo `tnsnames.ora` puede introducir entradas nuevas, pegando los datos de conexión y guardando el archivo o ejecutando el asistente *Net Configuration Assistant* de Oracle (si está disponible). Si quiere que estos valores aparezcan en las listas desplegables durante el proceso de configuración, puede que necesite añadir la ruta de acceso a la carpeta de administrador como una variable de entorno **TNS_ADMIN**.

Para conectarse a Oracle por ODBC:

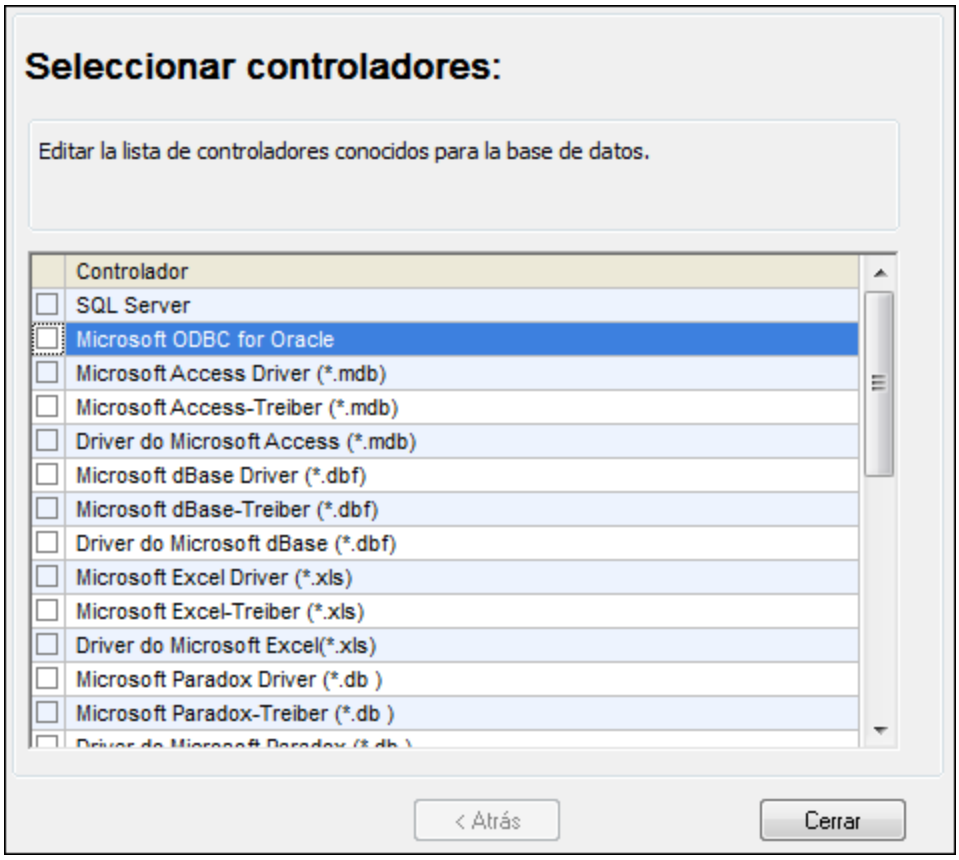
1. Inicie el asistente para la conexión de base de datos.
2. Seleccione la opción *Oracle (ODBC / JDBC)* y después haga clic en **Siguiente**.



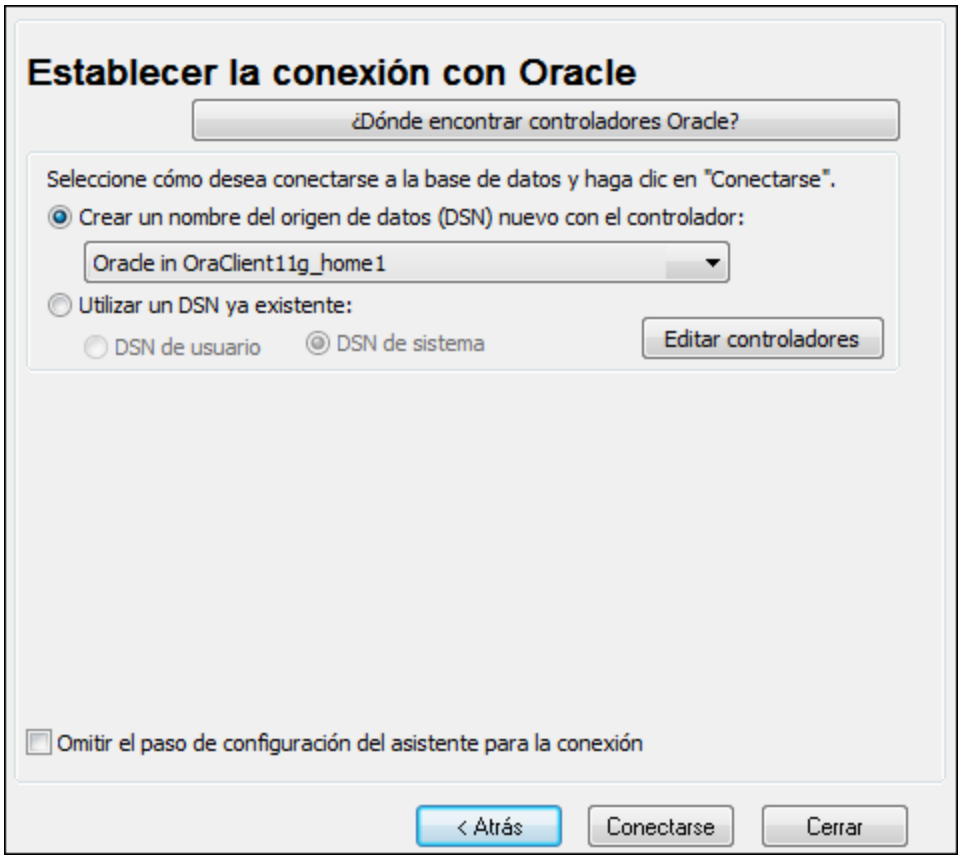
3. Seleccione el botón de opción *ODBC*.



4. Haga clic en **Editar controladores**.

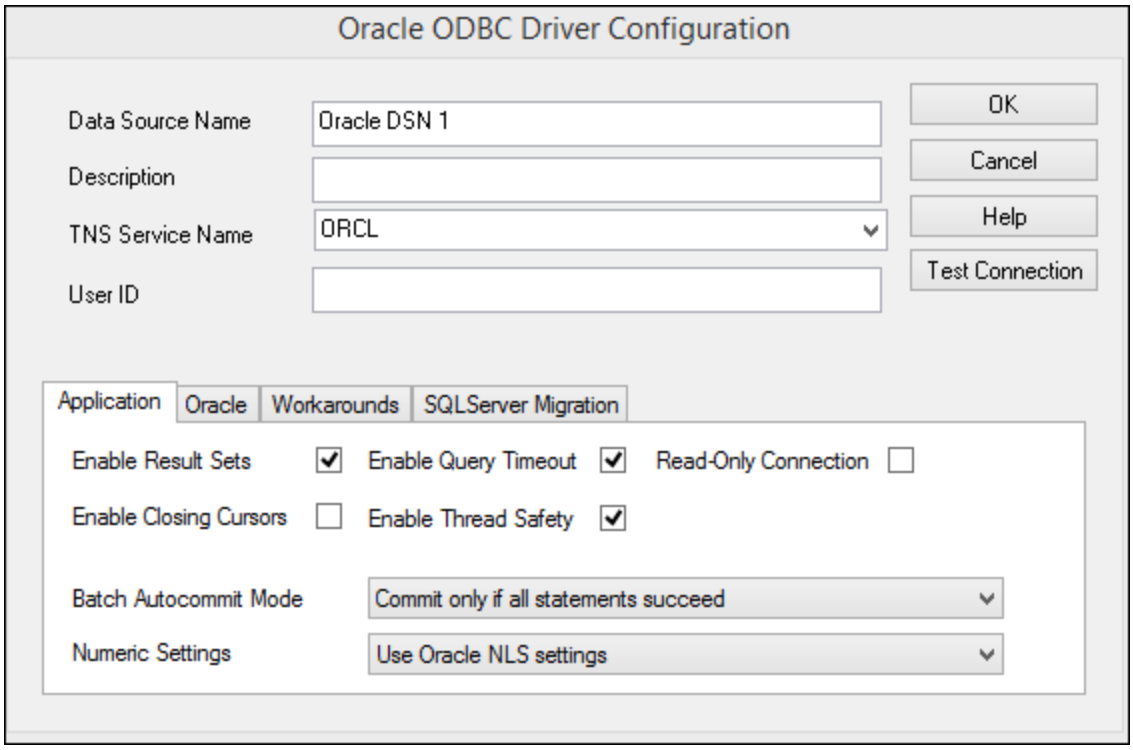


5. Seleccione los controladores Oracle que desea usar (en este ejemplo usamos **Oracle in OraClient11g_home1**). La lista incluye todos los controladores Oracle que están disponibles en el sistema después de instalar el cliente Oracle.
6. Haga clic en **Atrás**.
7. Seleccione la opción *Crear un DSN nuevo con el controlador* y después seleccione el controlador de Oracle elegido en el paso nº 4.

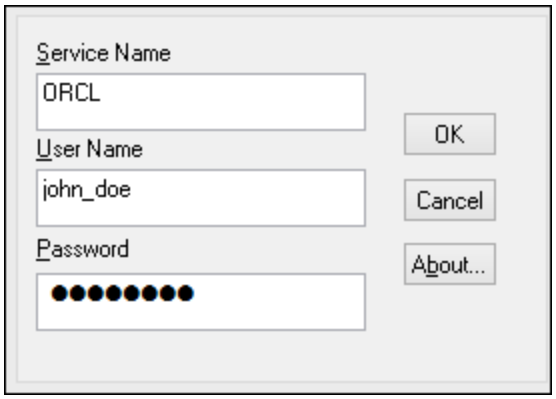


Recomendamos no utilizar el controlador **Microsoft ODBC for Oracle**. Microsoft recomienda utilizar el controlador ODBC que ofrece Oracle (consulte http://msdn.microsoft.com/en-us/library/ms714756%28v=vs.85%29.aspx)

8. Haga clic en **Conectarse**.



9. En el cuadro de texto *Nombre del origen de datos* introduzca un nombre que le ayude a identificar el origen de datos más adelante (en este ejemplo **Oracle DSN 1**).
10. En el cuadro de texto *Nombre del servicio TNS* introduzca el nombre de la conexión tal y como se define en el archivo `tnsnames.ora` (ver Requisitos). En este ejemplo el nombre de la conexión es **ORCL**.
11. Haga clic en **Aceptar**.



12. Escriba el nombre de usuario y la contraseña de la base de datos y haga clic en **Aceptar** para terminar.

10.2.9.15 PostgreSQL (ODBC)

Este ejemplo explica cómo conectarse a un servidor de base de datos PostgreSQL desde un equipo Windows con el controlador ODBC. El controlador ODBC PostgreSQL no está disponible en Windows así que deberá descargarlo e instalarlo por separado. En este ejemplo usamos la versión 11.0 del controlador psqlODBC, que se puede descargar del sitio web oficial (consulte el apartado Resumen de controladores de base de datos).

Nota: también se puede conectar a un servidor de base de datos PostgreSQL directamente (es decir, sin el controlador ODBC). Consulte el apartado Conexiones PostgreSQL para más información.

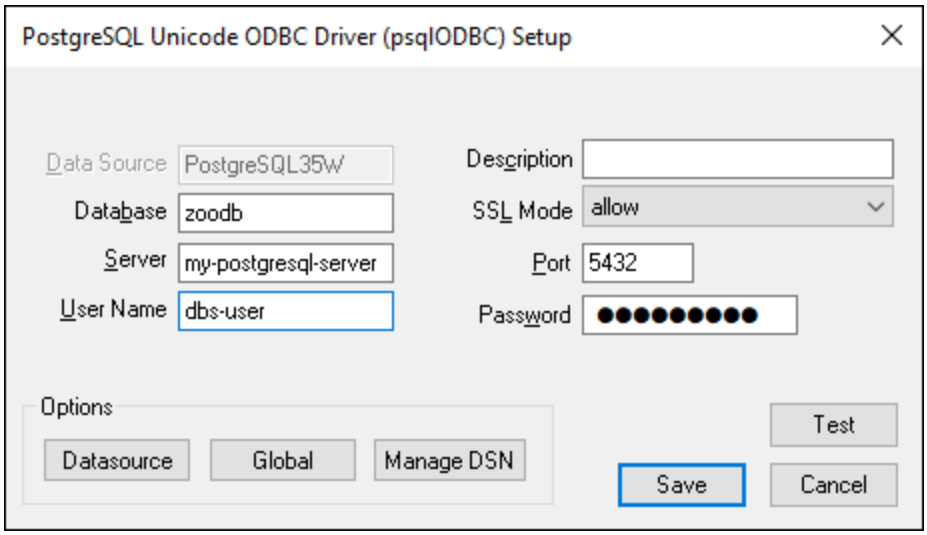
Requisitos:

- El controlador *psqlODBC* está instalado en el sistema.
- Disponer de los datos de conexión: servidor, puerto, base de datos, nombre de usuario y contraseña.

Para conectarse a PostgreSQL por ODBC:

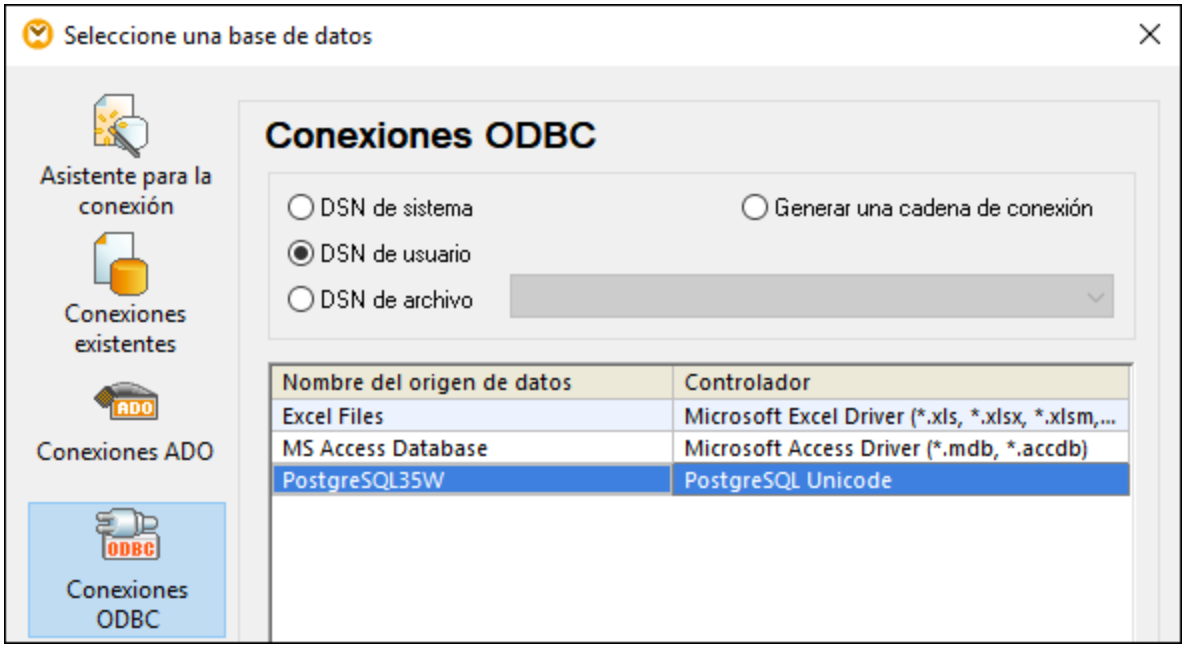
1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones ODBC**.
3. Seleccione la opción *DNS de usuario*.
4. Haga clic en **Crear DNS nuevo** y seleccione el controlador en la lista desplegable. Si no hay ningún controlador PostgreSQL en la lista, asegúrese de que el controlador PostgreSQL está instalado en su sistema operativo, tal y como se indica más arriba en los requisitos.

5. Haga clic en **DNS de usuario**.



6. Rellene las credenciales de la conexión de base de datos (las debe proporcionar el propietario de la base de datos) y haga clic en **Aceptar**.

Ahora la conexión está disponible en la lista de conexiones ODBC. Para conectarse a la base de datos puede hacer doble clic en la conexión o seleccionarla y hacer clic en **Conectarse**.



10.2.9.16 Progress OpenEdge (JDBC)

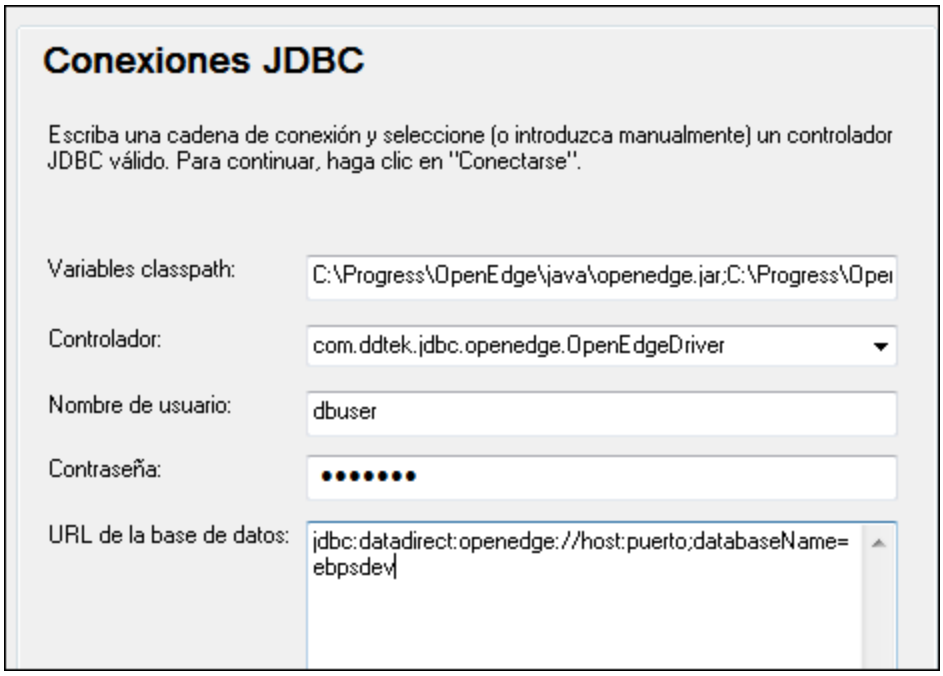
En este apartado encontrará instrucciones para conectarse a un servidor de base de datos Progress OpenEdge 11.6 por JDBC.

Requisitos:

- Debe tener instalado Java Runtime Environment (JRE) o Java Development Kit (JDK). Este último puede ser el JDK de Oracle o uno de código abierto, como Oracle OpenJDK. UModel identifica la ruta de acceso al equipo virtual Java (JVM) a partir de estas ubicaciones, en este orden: a) la ruta personal de acceso al JVM que puede indicar en en las **Opciones** (véase el apartado Java); b) la ruta de acceso al JVM que se encuentra en el registro; c) la variable de entorno `JAVA_HOME`.
- Asegúrese de que la plataforma de UModel (32 o 64 bits) es la misma que la de JRE/JDK.
- La variable de entorno `PATH` del sistema incluye la ruta al directorio `bin` del directorio de instalación de JRE o JDK (p. ej. `C:\Archivos de programa (x86)\Java\jre1.8.0_51\bin`).
- El controlador JDBC Progress OpenEdge está disponible en el sistema operativo. En este ejemplo la conexión por JDBC se consigue con los archivos de controlador **openedge.jar** y **pool.jar** disponibles en `C:\Progress\OpenEdge\java` y que se instalan con OpenEdge SDK.
- Disponer de los datos de conexión: host, puerto, nombre de la base de datos, nombre de usuario y contraseña.

Para conectarse a Progress OpenEdge por JDBC:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones JDBC**.
3. En el campo *Variables classpath* debe introducir la ruta de acceso del archivo .jar que ofrece conectividad con la base de datos. Si fuera necesario, también puede introducir una lista de rutas de archivo .jar separadas por caracteres de punto y coma. En este ejemplo, los archivos .jar están ubicados en esta ruta de acceso: **`C:\Progress\OpenEdge\java\openedge.jar;C:\Progress\OpenEdge\java\pool.jar;`**. Tenga en cuenta que este campo puede dejarse en blanco si añadió la ruta de acceso de los archivos .jar a la variable de entorno CLASSPATH del sistema operativo (*véase Configurar la variable CLASSPATH*).
4. En el campo *Controlador* seleccione **com.ddtek.jdbc.openedge.OpenEdgeDriver**. Recuerde que esta entrada solo estará disponible si se encuentra una ruta de archivo .jar válida en el campo Variables *classpath* o en la variable de entorno CLASSPATH del sistema operativo.



5. Introduzca el nombre de usuario y la contraseña de la base de datos.
6. Introduzca la cadena de conexión para el servidor de BD en el cuadro de texto *URL de la base de datos* (reemplace lo valores resaltados con los de su base de datos).

```
jdbc:datadirect:openedge://host:puerto;databaseName=nombre_BD
```

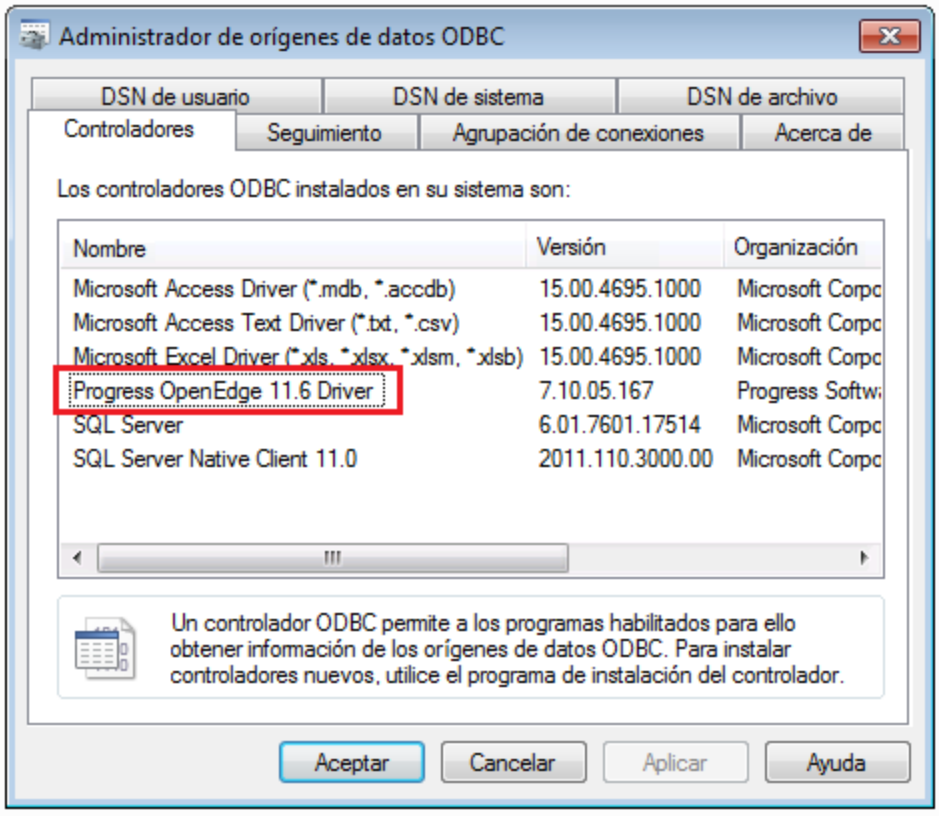
7. Haga clic en **Conectarse**.

10.2.9.17 Progress OpenEdge (ODBC)

En este apartado encontrará instrucciones para conectarse a un servidor de base de datos Progress OpenEdge por medio del controlador ODBC Progress OpenEdge 11.6.

Requisitos:

- El controlador *ODBC Connector for Progress OpenEdge* está instalado en el sistema operativo. Este controlador se puede descargar del sitio web del proveedor (consulte la lista del apartado Resumen de controladores de base de datos). Si trabaja con la versión de 32 bits de UModel, descargue el controlador de 32 bits. Por el contrario, si usa la versión de 64 bits, descargue el controlador de 64 bits. Una vez finalizada la instalación, compruebe que el controlador ODBC está disponible en el equipo (*véase Ver los controladores ODBC disponibles*).

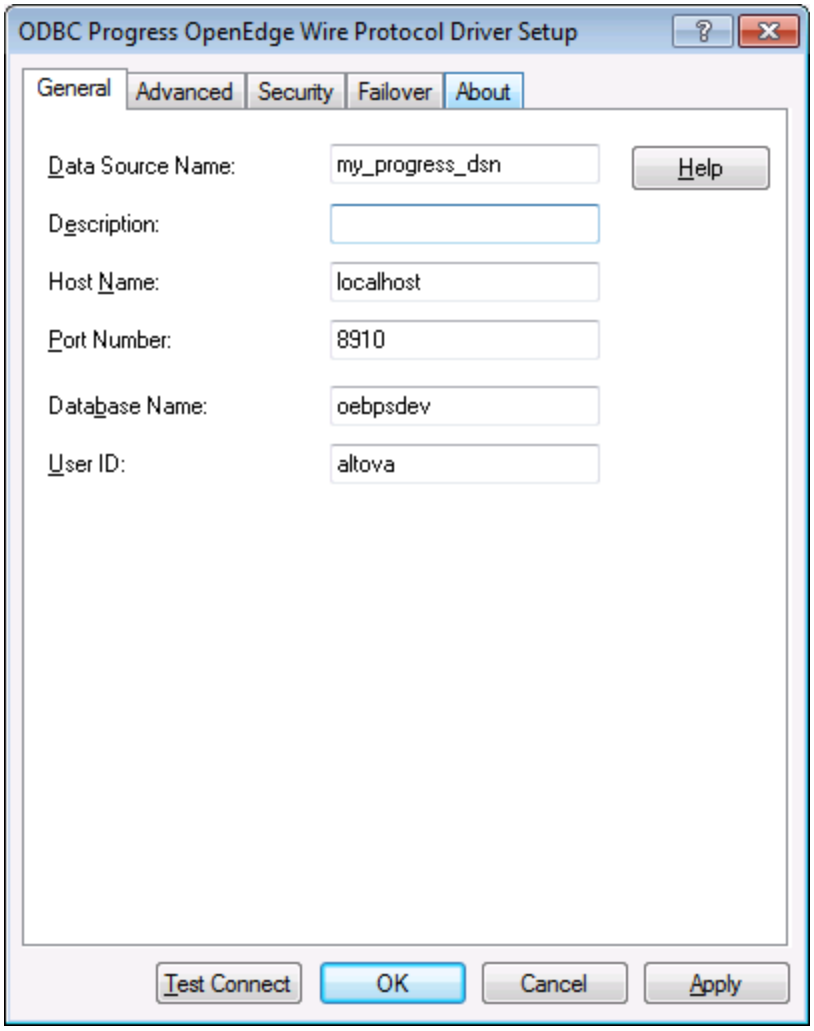


- Disponer de los datos de conexión: nombre de host, número de puerto, nombre de la base de datos, id. de usuario y contraseña.

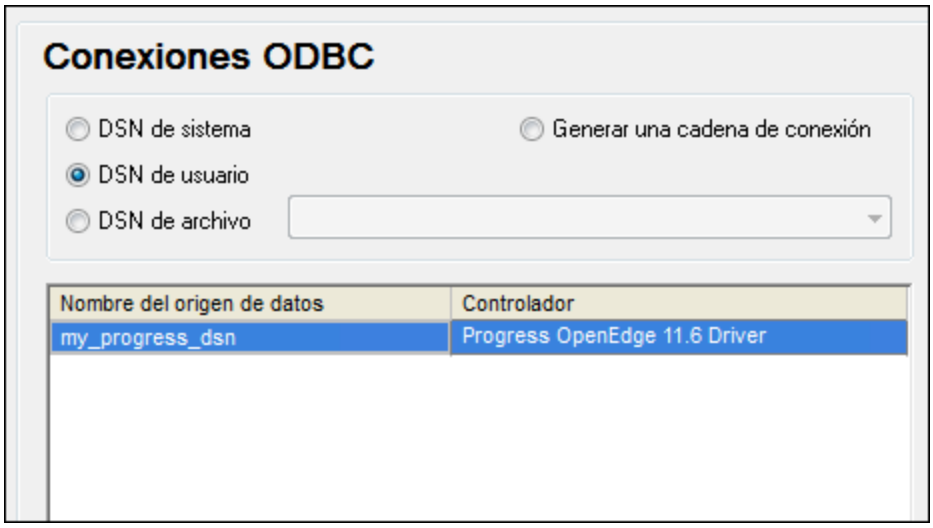
Para conectarse a Progress OpenEdge por ODBC:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en el botón Conexiones ODBC.
3. Haga clic en *DSN de usuario* (o en *DSN de sistema* o *DSN de archivo*).
4. Ahora haga clic en el icono **Agregar** .
5. Seleccione el controlador **Progress OpenEdge** de la lista y haga clic en el botón **DSN de usuario** (o **DSN de sistema** según corresponda).

6. Rellene las credenciales para la conexión con la base de datos (base de datos, servidor, puerto, nombre de usuario y contraseña) y haga clic en **Aceptar**. Para probar la conexión antes de guardar los datos introducidos haga clic en el botón **Probar conexión**.



7. Haga clic en **Aceptar**. El origen de datos nuevo aparece ahora en la lista de orígenes de datos ODBC.



8. Para terminar haga clic en **Conectarse**.

10.2.9.18 Sybase (JDBC)

Este ejemplo explica cómo conectarse a un servidor de base de datos Sybase por JDBC.

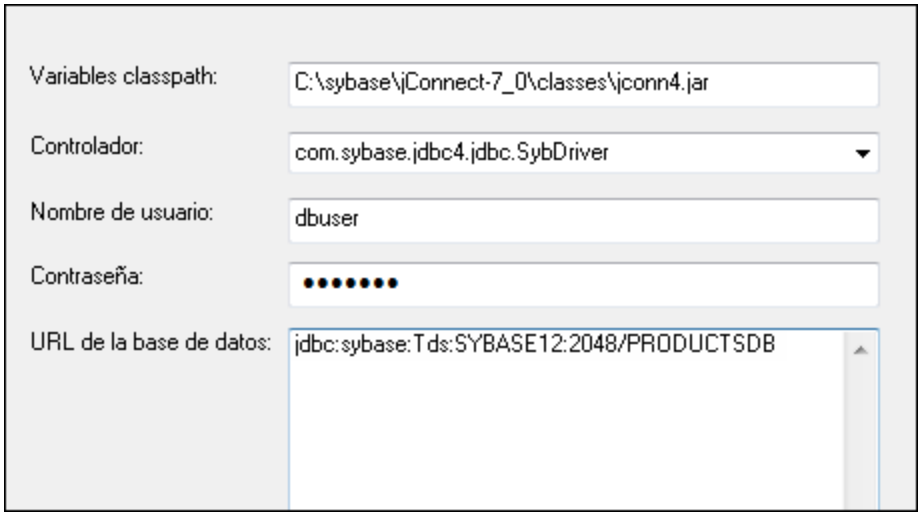
Requisitos:

- Debe tener instalado Java Runtime Environment (JRE) o Java Development Kit (JDK). Este último puede ser el JDK de Oracle o uno de código abierto, como Oracle OpenJDK. UModel identifica la ruta de acceso al equipo virtual Java (JVM) a partir de estas ubicaciones, en este orden: a) la ruta personal de acceso al JVM que puede indicar en en las **Opciones** (véase el apartado Java); b) la ruta de acceso al JVM que se encuentra en el registro; c) la variable de entorno `JAVA_HOME`.
- Asegúrese de que la plataforma de UModel (32 o 64 bits) es la misma que la de JRE/JDK.
- El componente Sybase *jConnect* está instalado en el sistema (en este ejemplo usamos *jConnect 7.0* que se instala con el cliente *Sybase Adaptive Server Enterprise PC Client*. Consulte la documentación de Sybase para obtener más información.
- Disponer de los datos de conexión: host, puerto, nombre de la base de datos, nombre de usuario y contraseña.

Para conectarse a Sybase por JDBC:

1. Inicie el asistente para la conexión de base de datos.
2. Haga clic en **Conexiones JDBC**.
3. En el campo *Variables classpath* debe introducir la ruta de acceso del archivo .jar que ofrece conectividad con la base de datos. Si fuera necesario, también puede introducir una lista de rutas de archivo .jar separadas por caracteres de punto y coma. En este ejemplo, el archivo .jar está ubicado en esta ruta de acceso: `C:\sybase\jConnect-7_0\classes\jconn4.jar`. Tenga en cuenta que este campo puede dejarse en blanco si añadió la ruta de acceso de los archivos .jar a la variable de entorno CLASSPATH del sistema operativo (*véase Configurar la variable CLASSPATH*).

4. En el campo *Controlador* seleccione **com.sybase.jdbc4.jdbc.SybDriver**. Recuerde que esta entrada solo estará disponible si se encuentra una ruta de archivo .jar válida en el campo Variables *classpath* o en la variable de entorno CLASSPATH del sistema operativo.



5. Introduzca el nombre de usuario y la contraseña de la base de datos.
6. Introduzca la cadena de conexión para el servidor de BD en el cuadro de texto *URL de la base de datos* (reemplace lo valores resaltados con los de su base de datos).

```
jdbc:sybase:Tds:nombreHost:puerto/nombreBD
```

7. Haga clic en **Conectarse**.

10.2.9.19 Teradata (JDBC)

Este ejemplo explica cómo conectarse a un servidor de base de datos Teradata por JDBC.

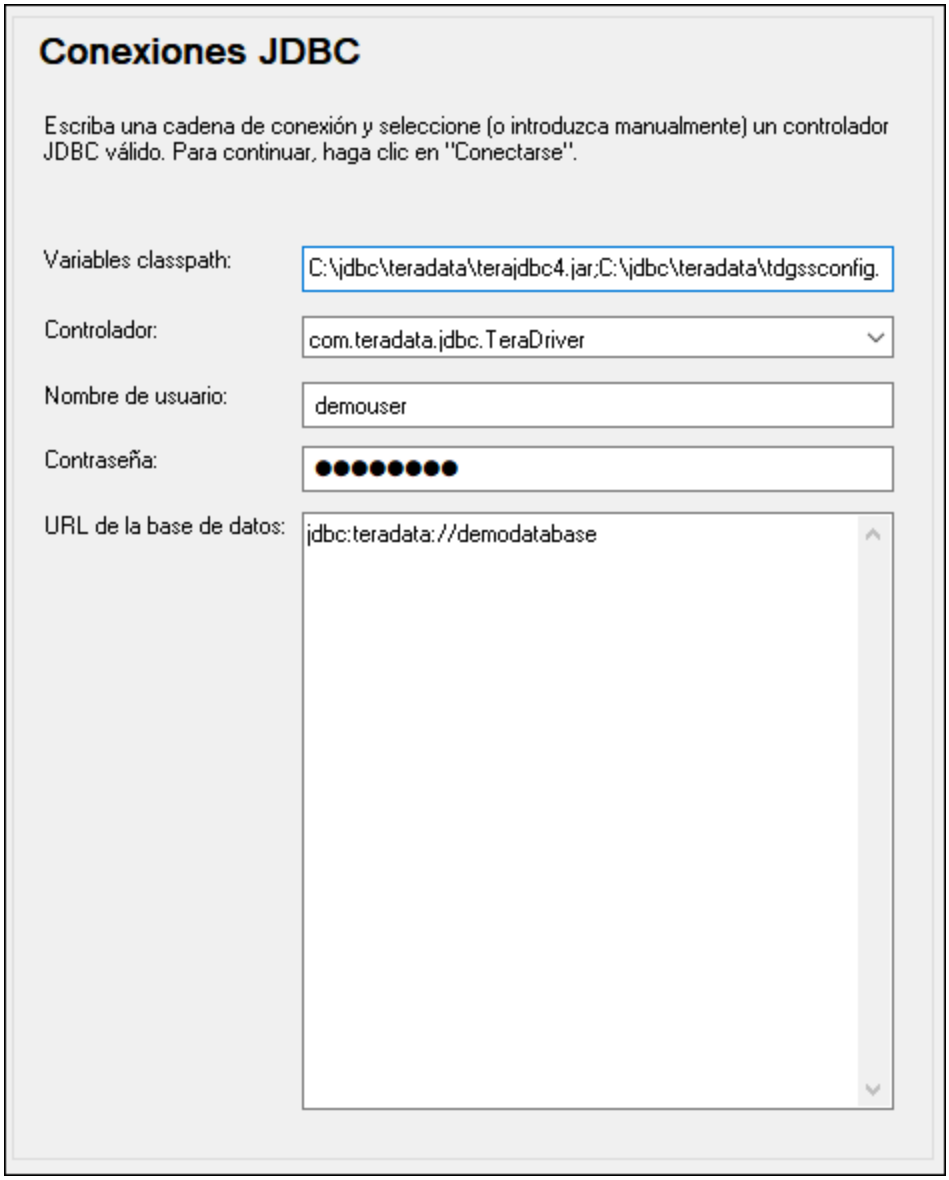
Requisitos:

- Debe tener instalado Java Runtime Environment (JRE) o Java Development Kit (JDK). Este último puede ser el JDK de Oracle o uno de código abierto, como Oracle OpenJDK. UModel identifica la ruta de acceso al equipo virtual Java (JVM) a partir de estas ubicaciones, en este orden: a) la ruta personal de acceso al JVM que puede indicar en en las **Opciones** (véase el apartado Java); b) la ruta de acceso al JVM que se encuentra en el registro; c) la variable de entorno `JAVA_HOME`.
- Asegúrese de que la plataforma de UModel (32 o 64 bits) es la misma que la de JRE/JDK.
- Debe contar con el controlador JDBC (archivos .jar que permiten conectarse a la base de datos) en el sistema operativo. Para este ejemplo usamos el controlador JDBC de Teradata 16.20.00.02 (https://downloads.teradata.com/download/connectivity/jdbc-driver).
- Disponer de los datos de conexión: host, base de datos, puerto, nombre de usuario y contraseña.

Para conectarse a Teradata por JDBC:

1. Inicie el asistente para la conexión de base de datos.

2. Haga clic en **Conexiones JDBC**.
3. Junto a *Variables Classpath* introduzca la ruta de acceso del archivo .jar que permite conectarse a la base de datos. Si es necesario, también puede introducir una lista de rutas de archivo .jar separadas por punto y comas. Para este ejemplo los archivos .jar están ubicados en esta ruta de acceso: **C:\jdbc\teradata**. Recuerde que puede dejar vacío el cuadro *Variables Classpath* si añadió las rutas de archivo .jar a la variable de entorno CLASSPATH del sistema operativo (véase Configurar la variable CLASSPATH).
4. En el cuadro *Controlador* seleccione **com.teradata.jdbc.TeraDriver**. Recuerde que esta entrada está disponible si en el cuadro *Variables Classpath* o en la variable de entorno CLASSPATH del sistema operativo se encuentra una ruta de archivo .jar válida (ver paso anterior).



5. Introduzca el nombre de usuario y la contraseña de la base de datos en las casillas correspondientes.
6. Introduzca la cadena de conexión en el cuadro de texto *URL de la base de datos* (reemplazando el valor resaltado con el valor correspondiente).

```
jdbc:teradata://nombreServidorBaseDatos
```

7. Por último, haga clic en **Conectarse**.

10.2.9.20 Teradata (ODBC)

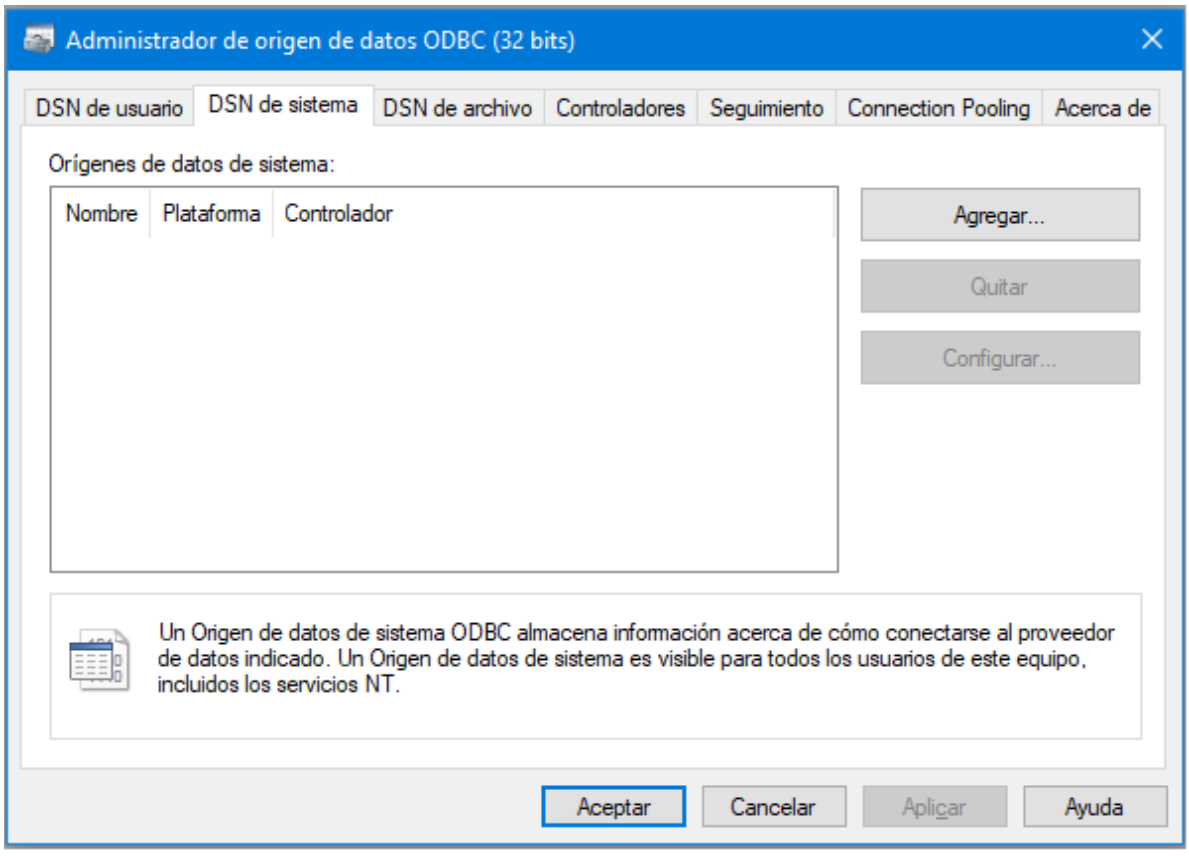
Este ejemplo explica cómo conectarse a un servidor de base de datos Teradata por ODBC.

Requisitos:

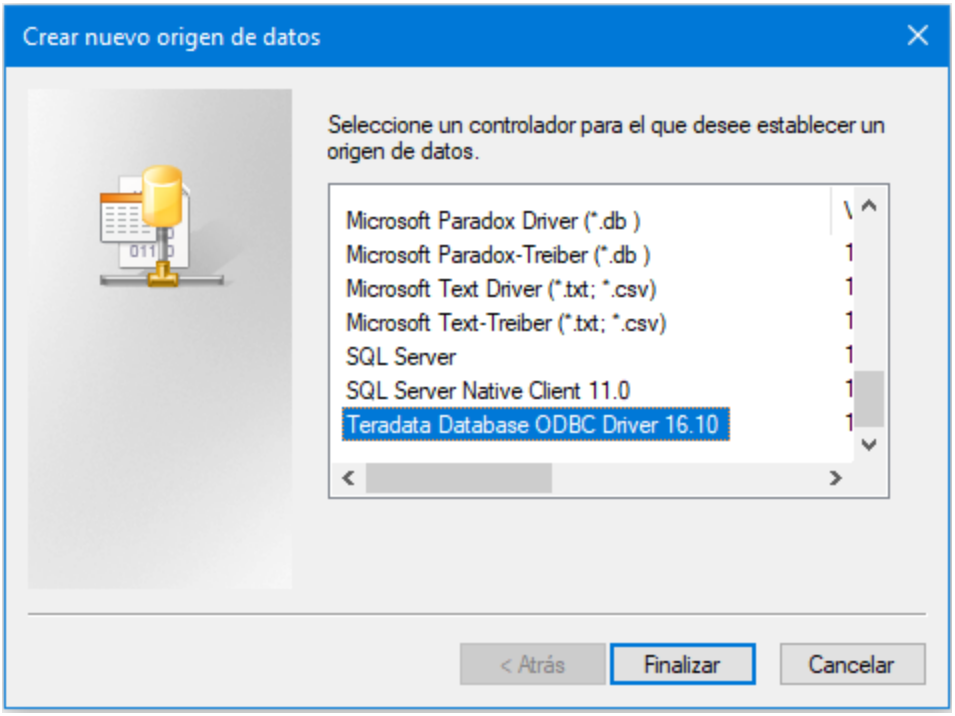
- Debe tener instalado el controlador ODBC de Teradata (https://downloads.teradata.com/download/connectivity/odbc-driver/windows). Para este ejemplo se utiliza el controlador ODBC de Teradata para Windows versión 16.20.00.
- Disponer de los datos de conexión: host, nombre de usuario y contraseña.

Para conectarse a Teradata por ODBC:

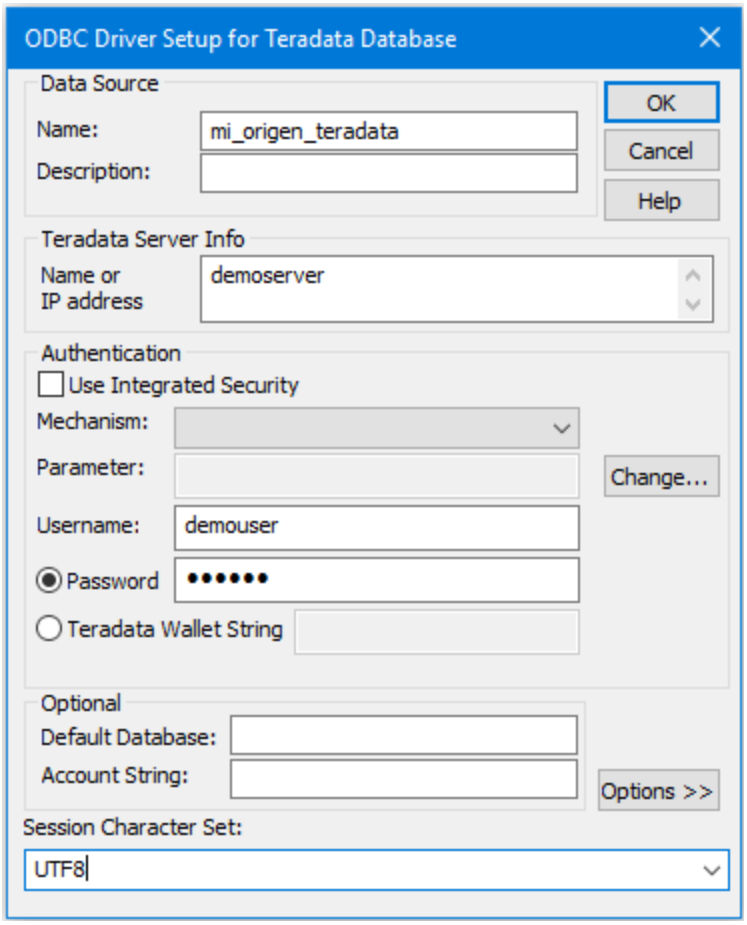
1. Pulse la tecla Windows, teclee "ODBC" y seleccione **Configurar orígenes de datos ODBC (32 bits)** en la lista de sugerencias. Si tiene un controlador ODBC de 64 bits, seleccione la opción **Configurar orígenes de datos ODBC (64 bits)** y utilice UModel de 64 bits a partir de ese momento.



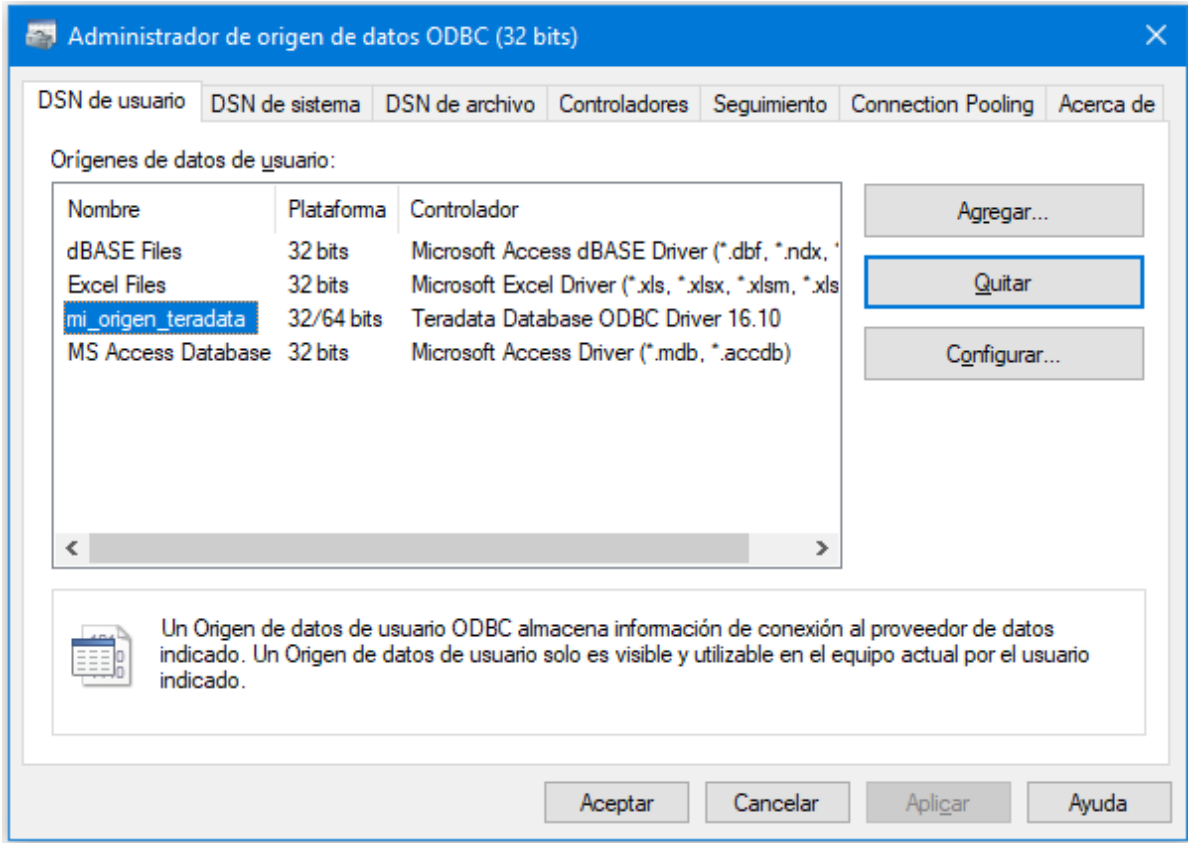
2. Haga clic en la pestaña *DSN de sistema* y después haga clic en **Agregar**.



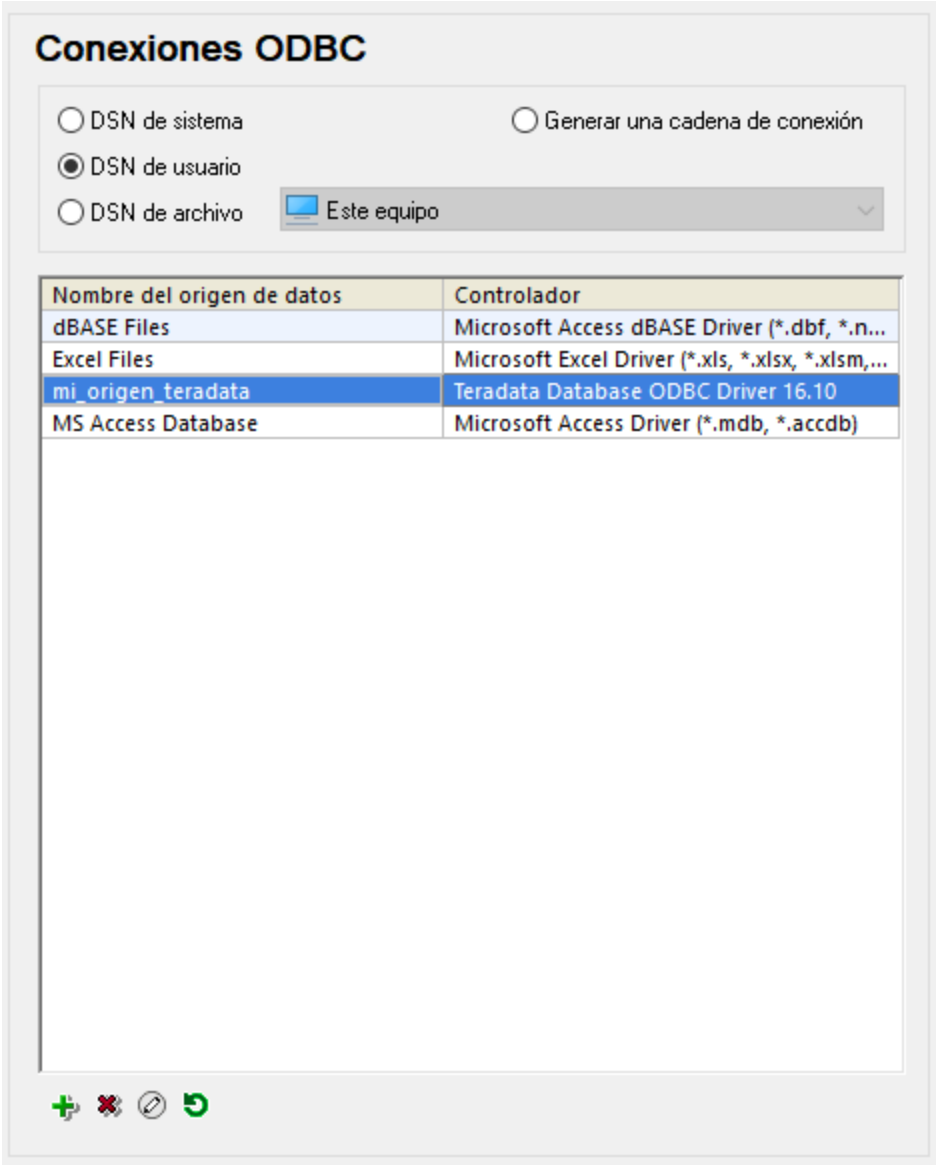
3. Seleccione **Teradata Database ODBC Driver** y después haga clic en **Finalizar**.



4. Introduzca el nombre y, si quiere, una descripción que le ayude a identificar este origen de datos ODBC más adelante. Además, deberá introducir las credenciales de la conexión de base de datos (servidor de BD, usuario y contraseña) y, si quiere, seleccione una base de datos.
5. Haga clic en **Aceptar**. El origen de datos aparecerá en la lista.



6. Ejecute UModel e inicie el asistente para la conexión de base de datos.
7. Haga clic en **Conexiones ODBC**.



8. Haga clic en *DSN de sistema,* seleccione el origen de datos creado en los pasos anteriores y después haga clic en el botón **Conectarse**.

Nota: si recibe el mensaje de error "Controlador devuelto no válido (o error en la devolución) SQL_DRIVER_ODBC_VER: 03.80", compruebe que la ruta de acceso del cliente ODBC (p. ej. **C:\Archivos de programa\Teradata\Client\16.10\bin**) existe en su variable de entorno PATH del sistema. Si falta esta ruta de acceso, entonces deberá añadirla a mano.

11 XMI: intercambio de metadatos XML

Sitio web de Altova: Intercambio de modelos UML mediante archivos XMI

Puede exportar proyectos de UModel a archivos de intercambio de metadatos (XML de Intercambio de Metadatos) e importar archivos XMI en proyectos de UModel. Esto garantiza la interoperabilidad con otras herramientas UML que admitan XMI. Estas son las versiones compatibles:

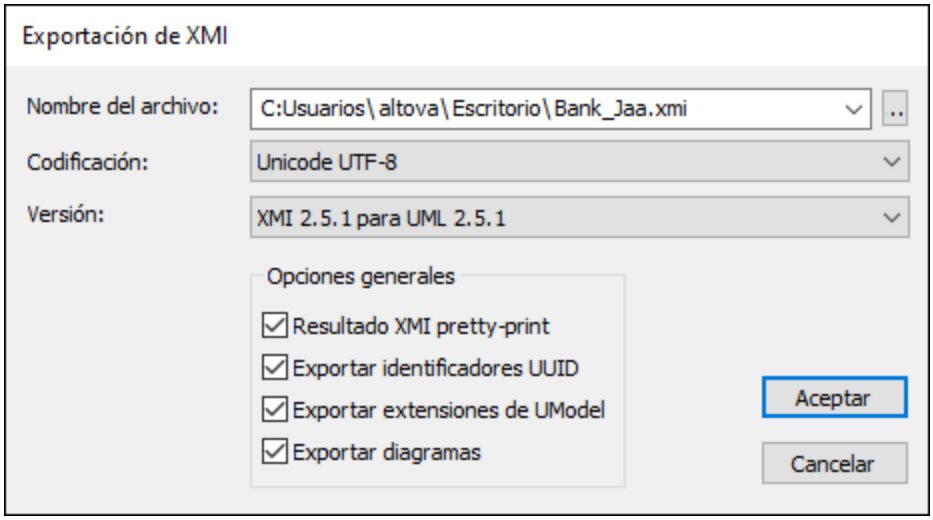
- XMI 2.1 para UML 2.0
- XMI 2.1 para UML 2.1.2
- XMI 2.1 para UML 2.2
- XMI 2.1 para UML 2.3
- XMI 2.4.1 para UML 2.4.1
- XMI 2.4.1 para UML 2.5
- XMI 2.5.1 para UML 2.5.1

Para importar un archivo XMI en UModel:

- En el menú **Archivo** haga clic en **Importar desde un archivo XMI**.

Para exportar un proyecto UModel a un archivo XMI:

- En el menú **Archivo** haga clic en **Exportar a un archivo XMI**



Notas:

- Durante el proceso de exportación se exportan todos los archivos incluidos, también los incluidos mediante referencia.
- Si tiene pensado reimportar el código XMI al proyecto de UModel, asegúrese de marcar la casilla *Exportar extensiones de UModel*.

A continuación explicamos las opciones disponibles al exportar proyectos a UMI.

Resultado XMI pretty-print

Marque esta casilla para aplicar sangría a las etiquetas XML del archivo XMI de destino e aplicarle los retornos de carro/saltos de línea correspondientes.

Exportar identificadores UUID

El estándar XMI define tres tipos de identificadores de elementos: ID, UUID y etiquetas.

- Los **ID** son identificadores únicos en el documento XMI y son compatibles con la mayoría de las herramientas UML. UModel exporta este tipo de identificadores por defecto (es decir, no hace falta marcar ninguna casilla).
- Los **UUID** son identificadores universales únicos que sirven para asignar a cada elemento una identificación global única. Estos identificadores son únicos a nivel global (es decir, no solo en el documento XMI). Marque la casilla *exportar identificadores UUID* para generar identificadores UUID.
- Los UUID se almacenan en el formato estándar UUID/GUID (p. ej. "6B29FC40-CA47-1067-B31D-00DD010662DA", "550e8400-e29b-41d4-a716-446655440000",...)
- UModel admite el uso de etiquetas para identificar elementos en XMI.

Nota: el proceso de importación XMI es compatible automáticamente con los identificadores ID y UUID.

Exportar extensiones de UModel

XMI define un mecanismo que permite a cada aplicación exportar sus propias extensiones a la especificación UML. Sin embargo, algunas herramientas UML solo son capaces de importar los datos UML estándar (pasando por alto las extensiones de UModel). Sin embargo, cuando importe datos a UModel estos datos de extensión estarán disponibles.

Algunos datos de UModel, como los nombres de archivo de clases o los colores de los elementos, no forman parte de la especificación UML y, por tanto, deben eliminarse en XMI o guardarse en *extensiones*. Si se exportan como extensiones y se vuelven a importar, estos nombres de archivo y colores se importan tal y como se definieron. Si no usa las extensiones para el proceso de exportación, estos datos de UModel se perderán.

Al importar un documento XMI, UModel detecta el formato y genera un modelo automáticamente.

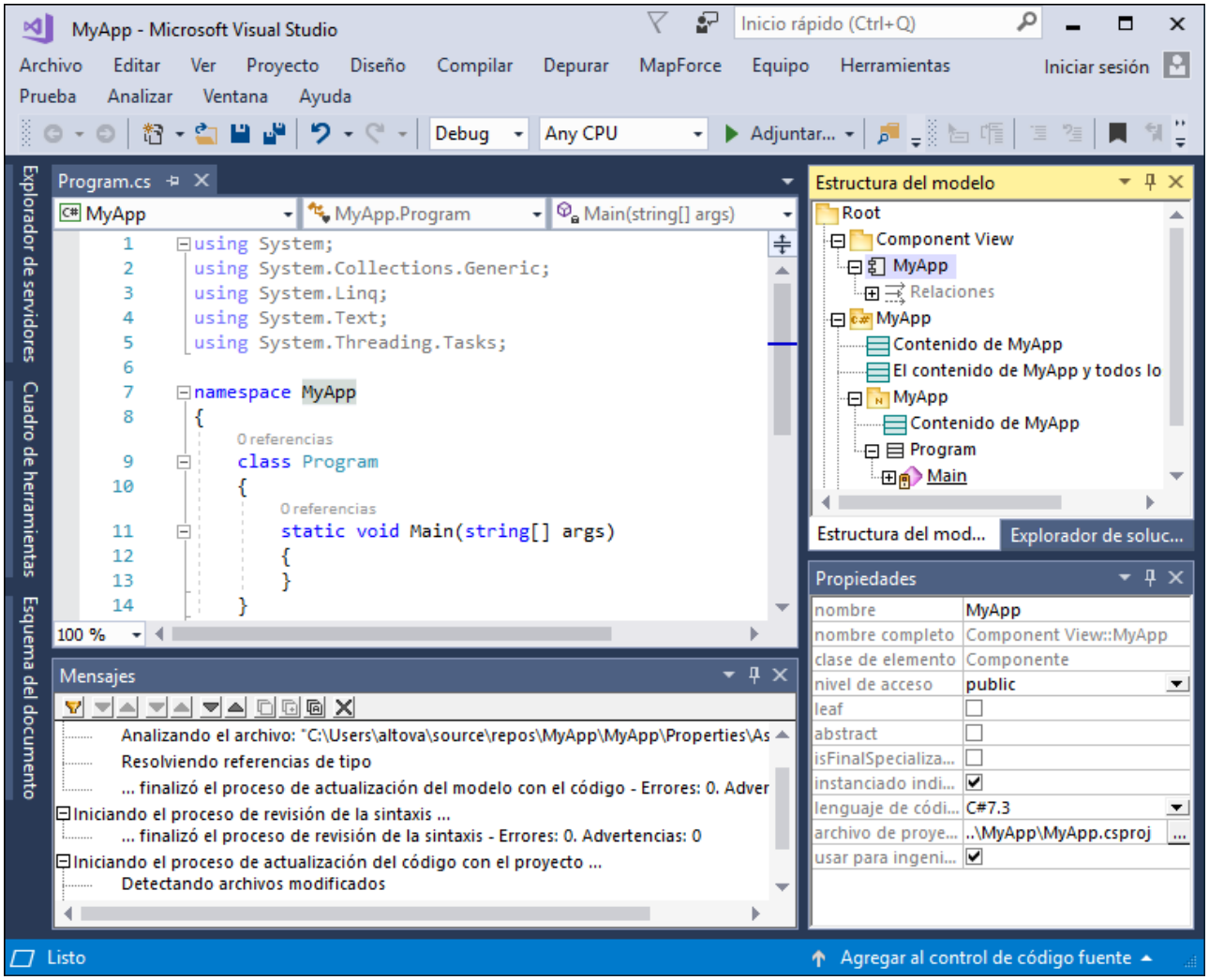
Exportar diagramas

Marque esta casilla para exportar los diagramas de UModel como *extensiones* en el archivo XMI. Para poder guardar los diagramas como extensiones debe activar también la opción *Exportar extensiones de UModel.*

12 Complemento de UModel para Visual Studio

UModel 2023 se puede integrar en Microsoft Visual Studio 2012/2013/2015/2017/2019/2022 y así combinar las funciones de modelado de UModel con este popular entorno de desarrollo.

Una de las ventajas de usar UModel como complemento de Visual Studio es que podrá sincronizar automáticamente su código C# o VB.NET y su modelo UML. Esto significa que, si realiza cambios en el código en Visual Studio, estos cambios se propagan automáticamente al modelo. Asimismo, si realiza cambios en el modelo (p. ej. si edita los diagramas de clases), estos cambios se propagan automáticamente al código. Además, puede deshabilitar la sincronización automática si prefiere sincronizar a mano el código con el modelo o viceversa.



Proyecto de ejemplo del complemento de UModel para Visual Studio 2017

El complemento de UModel para Visual Studio tiene algunas diferencias con la edición independiente de UModel:

- La sincronización automática de modelos de UModel y código de proyecto está disponible (consulte el apartado Sincronizar el modelo y el código).
- Puede acceder a las funciones de UModel desde estos menús de Visual Studio:

Archivo	Incluye comandos para trabajar con UModel y con Visual Studio.
Edición	Incluye comandos para trabajar con UModel y con Visual Studio.
Vista	Los comandos para trabajar con UModel están dentro del submenú **UModel**.
Proyecto	Los comandos para trabajar con UModel están dentro del submenú **UModel**.
Diseño	Incluye los mismos comandos que la edición independiente de UModel.
Herramientas	Incluye comandos para trabajar con UModel y con Visual Studio. Las opciones de UModel están en el submenú **Opciones de UModel**.
Ayuda	Los archivos de ayuda de UModel están en el submenú **Ayuda de UModel**.

- Si el cursor está en el editor de código de Visual Studio y hace clic con el botón derecho, estos son los comandos de UModel que incluye el menú contextual (si procede):
  - **Ir al modelo UML**
  - **Aplicar ingeniería inversa al archivo actual**
  - **Generar diagrama de secuencias...**

  Además, si el cursor está dentro de un elemento en la ventana *Estructura del modelo*, también estará disponible el comando de menú contextual **Ir al código** (si procede).

- Cuando utilice UModel como complemento de Visual Studio, puede usar las funciones de control de código fuente de Visual Studio. Los comandos de control de código fuente de la edición independiente de UModel no funcionarán.
- Los cuadros de diálogo que se abren al seleccionar los comandos de menú **UModel | Importar directorio de código fuente** y **UModel | Importar proyecto de código fuente** no incluyen la opción para seleccionar el lenguaje C# ni Visual Basic. La importación de otros proyectos se hace con los comandos de Visual Studio (p. ej. **Archivo | Agregar | Proyecto existente**).
- El editor de scripts (**Herramientas | Editor de scripts**) y el comando de menú **Herramientas | Restaurar barras de herramientas y ventanas** no se incluye en el complemento para Visual Studio.

12.1 Instalar el complemento de UModel para Visual Studio

Siga estas instrucciones para instalar el complemento de UModel para Visual Studio:

1. Instale Microsoft Visual Studio 2012/2013/2015/2017/2019/2022. Tenga en cuenta que a partir de su versión 2022, Visual Studio solo está disponible como aplicación de 64 bits.
2. Instale UModel (ediciones Enterprise o Professional Edition). Si tiene instalada la versión 2022 o una más avanzada, debe instalar la versión de 64 bits de UModel.
3. Descargue y ejecute el paquete de integración de UModel. Este paquete está disponible en la página de descargas de UModel (ediciones Enterprise y Professional) de www.altova.com/es.

Tras instalar el paquete de integración podrá empezar a usar UModel en el entorno de Visual Studio.

Nota importante

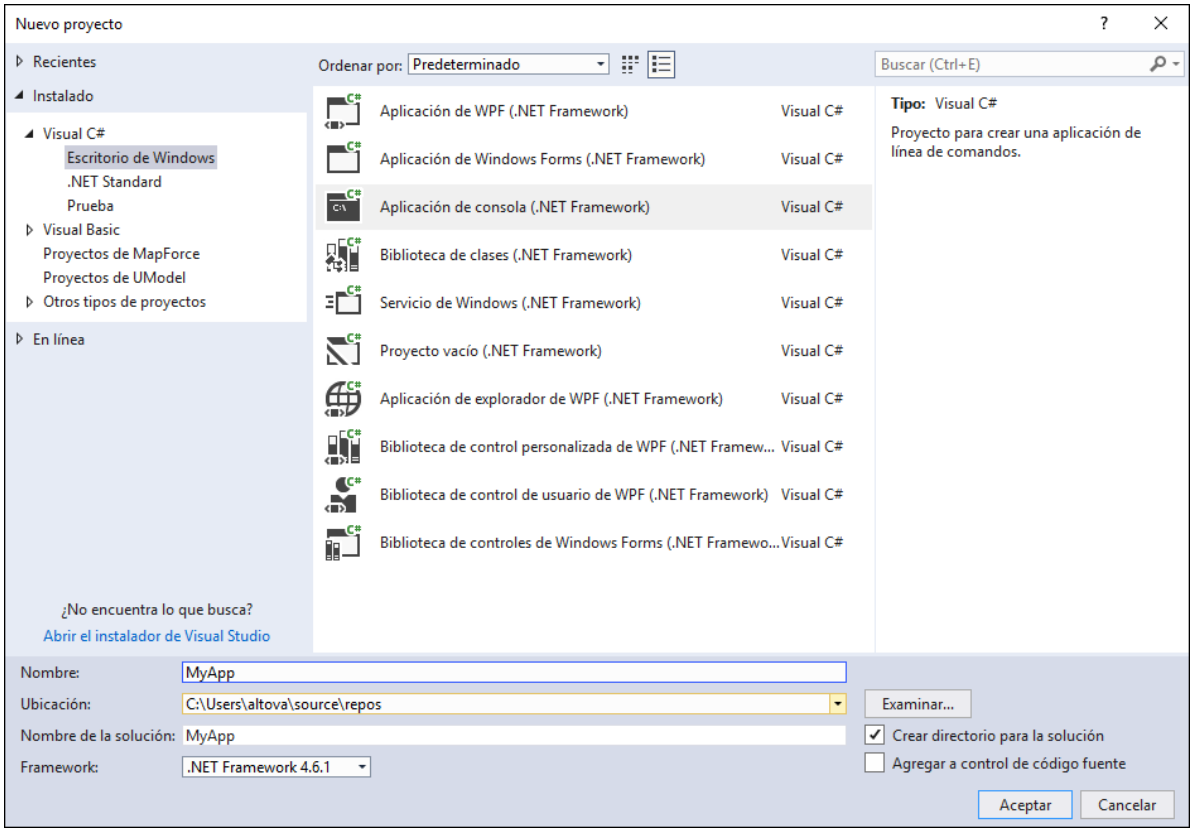
Debe usar el paquete de integración correspondiente a su versión de UModel (la versión actual es 2023). El paquete de integración no es específico de ninguna versión, por lo que se puede usar tanto para la edición Enterprise como Professional.

12.2 Agregar funciones de UModel a proyectos de Visual Studio

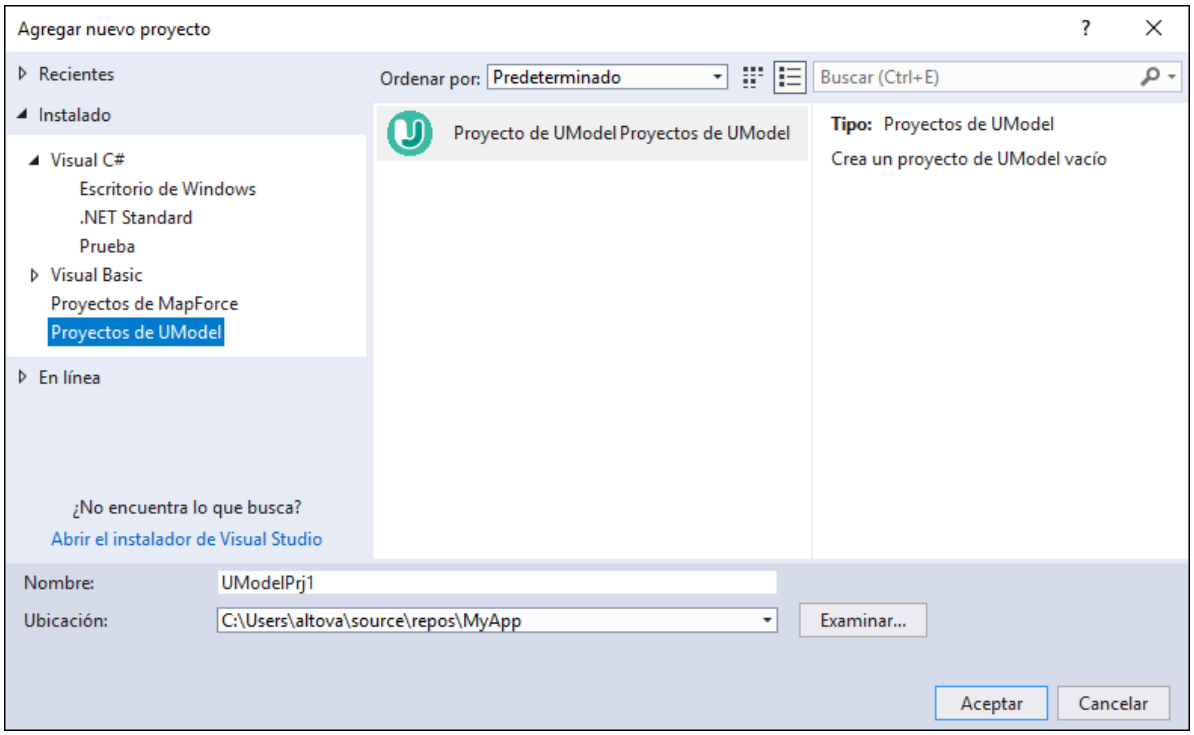
Puede agregar funciones de UModel a proyectos nuevos o actuales de Visual Studio para sincronizarlos automáticamente con modelos de UModel. Una solución de Visual Studio puede incluir un máximo de un proyecto de UModel.

Para agregar funciones de UModel a un proyecto de Visual Studio:

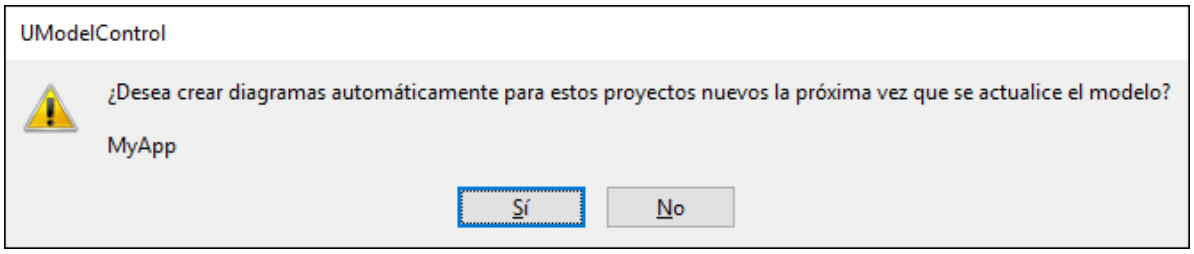
1. Cree un proyecto de Visual Studio nuevo o abra uno actual. Para este ejemplo creamos un proyecto C# llamado "MyApp" con Visual Studio 2017.



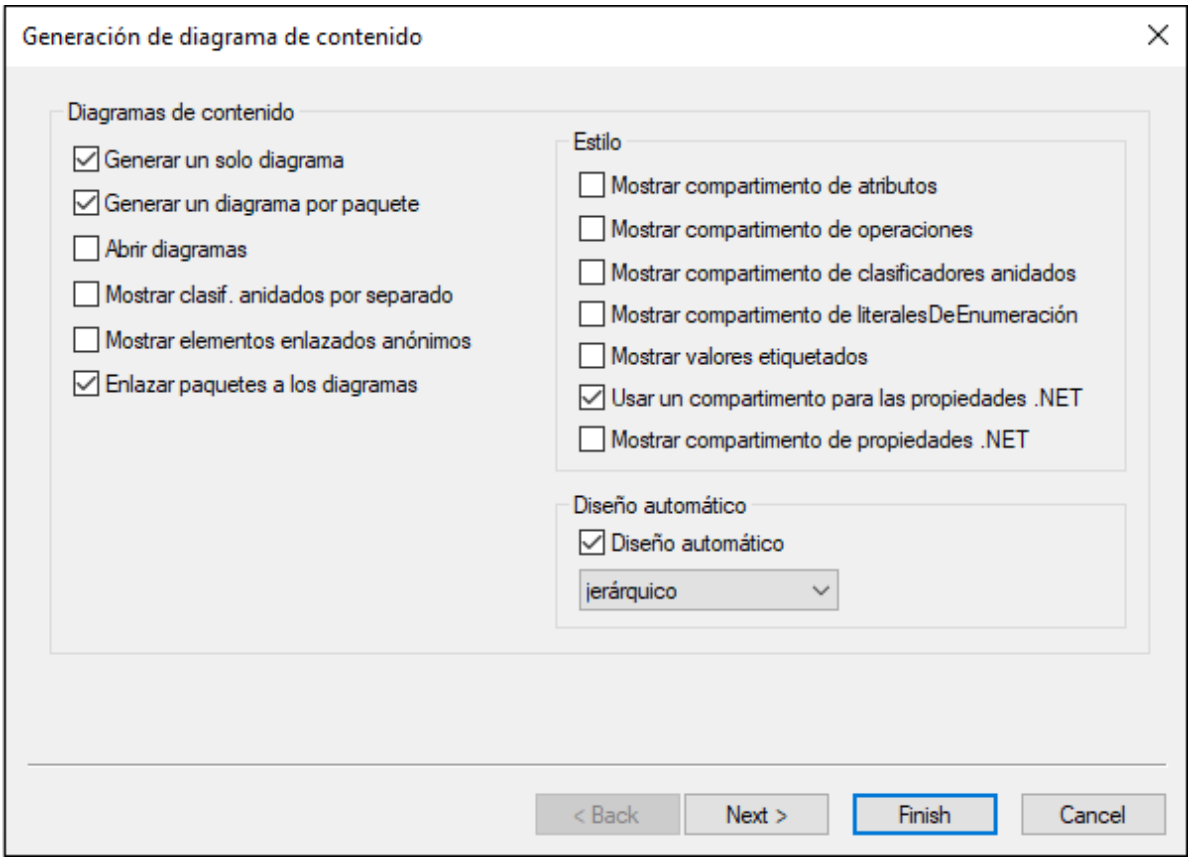
2. En el menú **Archivo** haga clic en **Agregar** y después en **Proyecto**.
3. Seleccione **Proyectos de UModel** y haga clic en **Aceptar**.



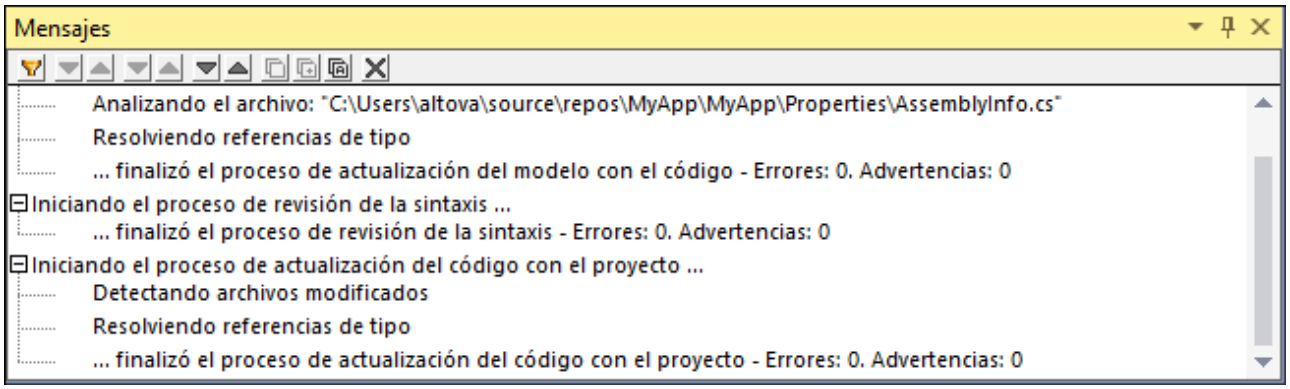
4. Si desea crear diagramas automáticamente en el modelo a partir del código, haga clic en **Sí** cuando aparezca un mensaje a tal efecto (opción recomendada).



5. Llegado el momento de configurar la generación de diagramas, elija las opciones que prefiera en el asistente y después haga clic en **Finalizar**. Los pasos del asistente son idénticos a los de la edición independiente de UModel.



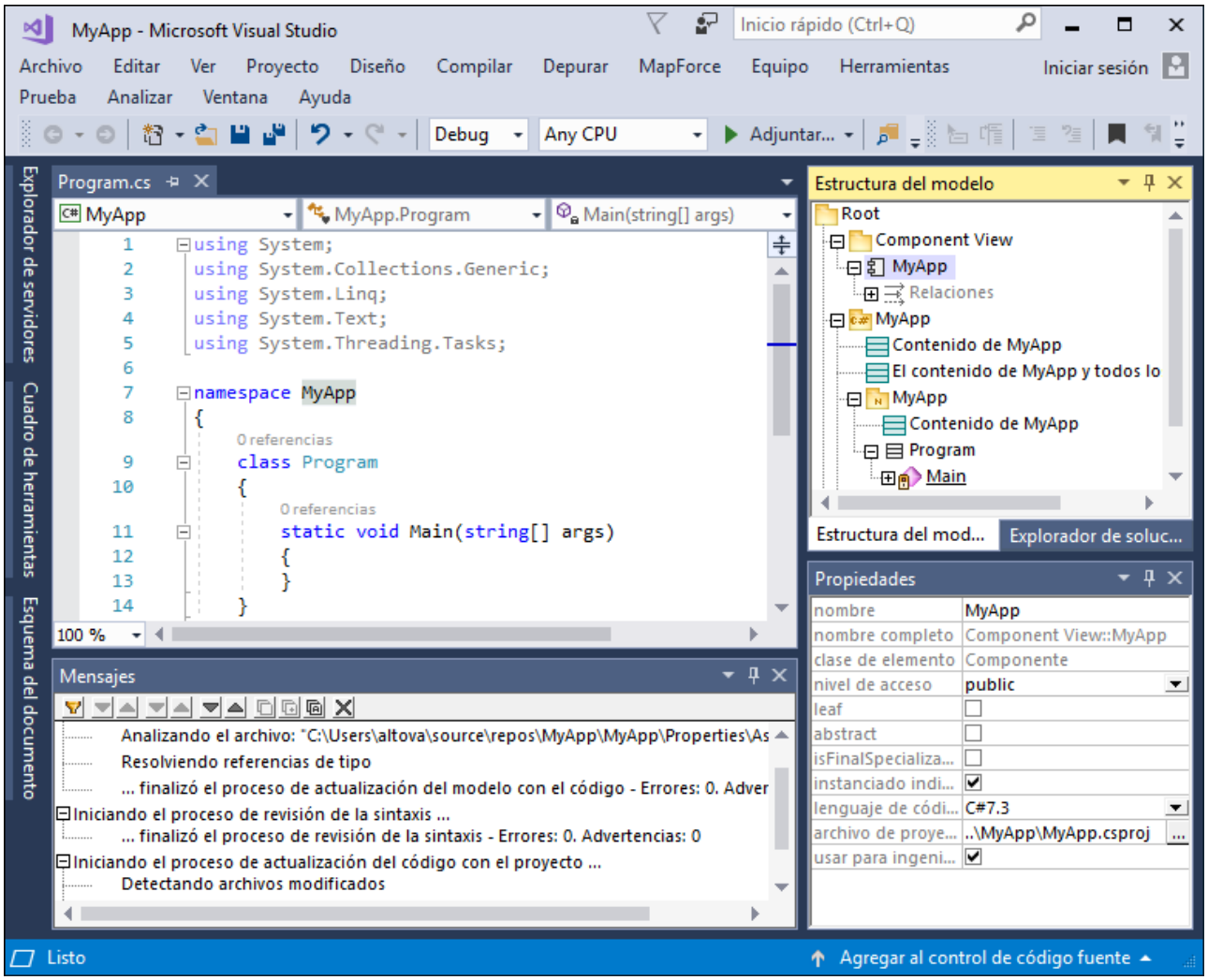
Al hacer clic en **Finalizar** UModel inicia el proceso de sincronización y aparece un cuadro de diálogo. Haga clic en **Aceptar** para cerrar este cuadro de diálogo. Los detalles de la sincronización aparecen en la ventana Mensajes.



Tenga en cuenta que la ventana Mensajes puede estar oculta en Visual Studio. Para ver esta ventana (o cualquier otra ventana de UModel) seleccione el comando de menú **Vista | UModel | [*Nombre de la ventana*]**.

Cuando añada un proyecto de UModel nuevo a una solución de Visual Studio, la configuración necesaria para la ingeniería de código (p. ej. la realización de componentes y el perfil C# o VB.NET) se define automáticamente. Para ver esta configuración basta con abrir los paneles *Estructura del modelo* y *Propiedades*

(en el menú **Vista** haga clic en **UModel | Estructura del modelo** y en **UModel | Propiedades** respectivamente). Asegúrese de hacer clic en el componente de ingeniería de código en la ventana *Estructura del modelo* (en este caso "MyApp") para que la ventana *Propiedades* incluya información.



12.3 Cargar y descargar proyectos de UModel

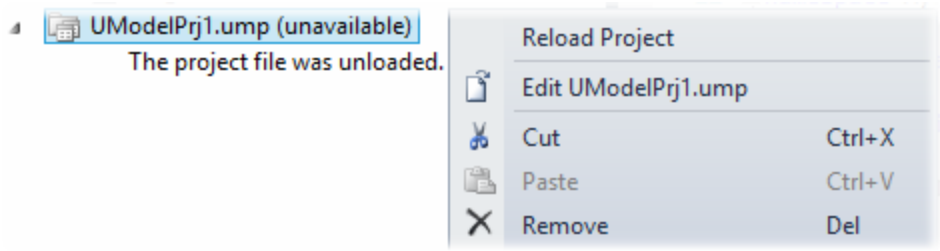
Una vez añadido a la solución de Visual Studio, el proyecto de UModel aparece en el explorador de soluciones de Visual Studio junto a los demás proyectos que formen parte de la solución. Si fuera necesario, puede descargar el proyecto de UModel temporalmente de la solución. Cuando esto ocurre, los archivos del proyecto de UModel se conservan en el disco y en el explorador de soluciones. Así podrá volver a cargar el proyecto en la solución más adelante.

Para descargar un proyecto de UModel de una solución de Visual Studio:

1. Haga clic en el proyecto de UModel en el explorador de soluciones de Visual Studio.
2. Haga clic en el comando de menú **Proyecto | Descargar el proyecto**.

Para volver a cargar el proyecto de UModel en la solución:

- Haga clic con el botón derecho en el proyecto en el explorador de soluciones y seleccione **Volver a cargar el proyecto**.



Para quitar un proyecto de UModel de la solución de Visual Studio:

- Descargue el proyecto (ver más arriba).
- Haga clic con el botón derecho en el proyecto en el explorador de soluciones y seleccione **Quitar**.

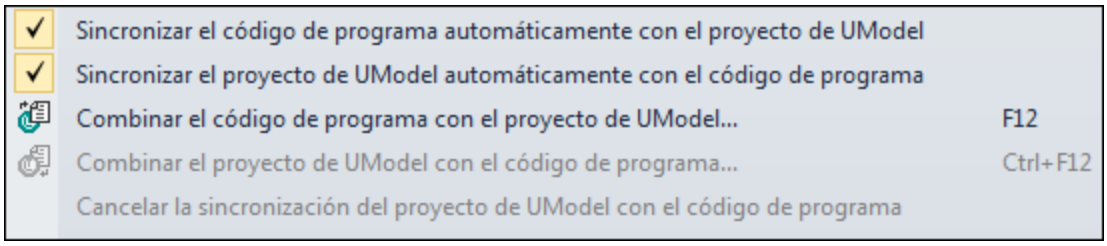
12.4 Sincronización de modelo y código

El proceso de sincronización entre el archivo de UModel (.ump), es decir, el modelo y el código C# o VB.NET puede ser un proceso manual o automático.

La sincronización automática tiene lugar después de añadir las funciones de UModel al proyecto de Visual Studio. Sincronización automática quiere decir que, cada vez que se edite el código, el complemento de UModel para Visual Studio analiza este código y actualiza el modelo. Asimismo, si se realizan cambios en el modelo (p. ej. si se edita un diagrama), el código se actualiza automáticamente para reflejar estos cambios. La sincronización manual, por otro lado, se inicia cuando el usuario lo solicita (*ver más abajo*).

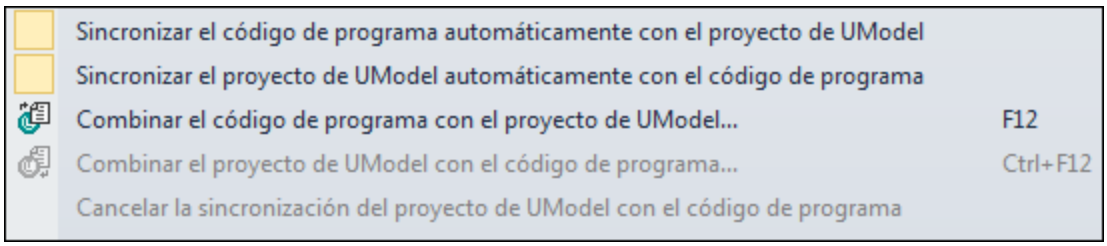
Tanto el proceso automático como el manual actualiza los cambios de forma masiva, es decir, para todo el proyecto. Cuando UModel se ejecuta como complemento de Visual Studio, en la *Estructura del modelo* no está disponible la opción para combinar o actualizar una sola clase.

Los comandos que controlan la sincronización automática o manual están disponibles en el menú **Proyecto | UModel**:



*Comandos del menú **Sincronización de código** (Visual Studio 2010)*

En las versiones más recientes de Visual Studio los elementos de menú seleccionados tienen un aspecto algo distinto:



*Comandos del menú **Sincronización de código** (Visual Studio 2017)*

A continuación describimos todos los comandos de sincronización de código.

Sincronizar el código de programa automáticamente con el proyecto de UModel	Esta opción está activada por defecto. Es decir, la sincronización del código con el modelo tiene lugar automáticamente. Para deshabilitar esta opción haga clic en ella.
Sincronizar el proyecto de UModel automáticamente con el código de	Igual que la opción anterior pero en el sentido contrario (se sincroniza el modelo con el código).

programa	
Combinar el código de programa con el proyecto de UModel	Actualiza el código de programa con los cambios realizados en el proyecto de UModel. Este mismo comando también se llama **Sobrescribir código de programa con el proyecto de UModel** si la opción está marcada en **Proyecto | Proyecto de UModel | Configurar sincronización**.
Combinar el proyecto de UModel con el código de programa	Actualiza el proyecto de UModel con los cambios realizados en el código de programa. Este mismo comando también se llama **Sobrescribir proyecto de UModel con el código de programa** si la opción está marcada en **Proyecto | Proyecto de UModel | Configurar sincronización**.
Cancelar la sincronización del proyecto de UModel con el código de programa	Permite cancelar la operación de sincronización que está teniendo lugar. Si no se está realizando ninguna operación de sincronización, entonces este comando está deshabilitado.

El progreso de la operación de sincronización aparece en la barra de estado de Visual Studio, por ejemplo:

En algunos casos no será posible sincronizar el código y el modelo:

- Si el código no se puede analizar.
- Si se produjo un error en el último proceso de ingeniería inversa o directa.
- Si UModel detecta un error de sintaxis.

Cuando esto ocurra, los detalles del error aparecerán en la ventana Mensajes. Para abrir el archivo de código fuente que contiene el error basta con hacer clic en la línea correspondiente en la ventana Mensajes. El cursor se coloca automáticamente en la línea que contiene el error.

Limitaciones de la sincronización automática

En Visual Studio algunas modificaciones de código C# y VB.NET no desencadenan un evento interno de Visual Studio y, por tanto, los cambios no se reflejan automáticamente en UModel. Cuando esto ocurra, tiene dos opciones: (i) forzar una sincronización manual o (ii) realizar una modificación distinta que sí desencadene la actualización del archivo de código fuente. La sincronización manual será necesaria cuando se añadan o modifiquen estas entidades:

- Valores predeterminados de atributos
- Valores predeterminados de parámetros de operaciones
- ParámetrosDePlantilla
- EnlacesDePlantilla
- Sección de resumen (para todos los elementos)

- Sección de comentarios (para todos los elementos)
- Todos los cambios en el cuerpo de los métodos

Recuerde que si realiza cambios en cualquiera de estos elementos de modelado en el modelo, la sincronización automática de código tendrá lugar. Es decir, no hay limitaciones a la hora de sincronizar el código con el modelo.

Para forzar la sincronización manual del modelo con el código, haga clic con el botón derecho en el archivo de código fuente en el editor de código y seleccione el comando **Aplicar ingeniería inversa al archivo actual** del menú contextual.

Si el proyecto de UModel contiene el perfil de lenguaje para Java, entonces la sincronización automática se deshabilita para dicho proyecto en Visual Studio y aparece un mensaje a tal efecto. Estos proyectos se deben sincronizar a mano (con los comandos de menú **UModel | Combinar el código de programa con el proyecto de UModel** y **UModel | Combinar el proyecto de UModel con el código de programa**). Si lo prefiere, también puede utilizar el complemento de UModel para Eclipse.

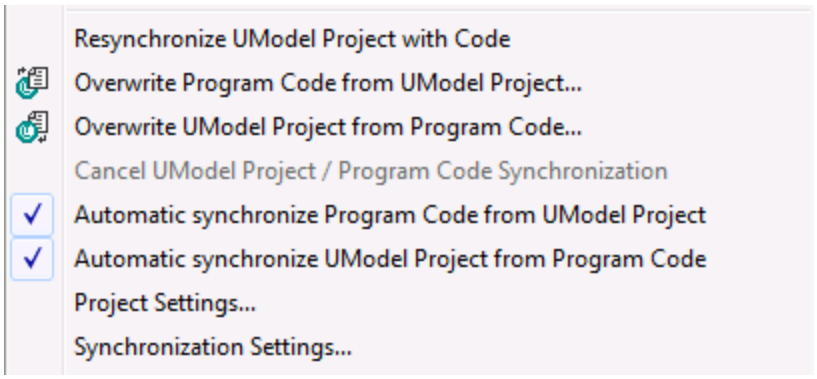
13 Complemento de UModel para Eclipse

Eclipse es un marco de trabajo de código abierto donde se integran diferentes tipos de aplicaciones en forma de complementos. El complemento de UModel para la plataforma Eclipse permite acceder a las funciones de UModel desde Eclipse directamente (versiones 2022-09, 2022-06, 2022-03, 2021-12) y expone comportamientos de Eclipse tal y como se explica en esta sección.

La principal ventaja de usar UModel como complemento de Eclipse es la función de sincronización automática entre código Java y los modelos de UModel. Es decir, si realiza cambios en el código Java en Eclipse, estos cambios se propagan automáticamente en el modelo. Asimismo, si realiza cambios en el modelo (p. ej. si edita diagramas de clases), estos cambios se propagan automáticamente en el código. Si lo prefiere, también puede deshabilitar la función de sincronización automática y sincronizar a mano el código con el modelo y viceversa.

El complemento de UModel para Eclipse tiene algunas diferencias con la edición independiente de UModel:

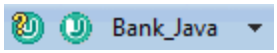
- En Eclipse algunos elementos de la interfaz gráfica se ajustan al entorno de desarrollo Eclipse (consulte el apartado La perspectiva UModel). Al igual que en la versión independiente, en Eclipse algunos elementos de la interfaz se deshabilitan cuando no son relevantes. Por ejemplo, los botones de la barra de herramientas de UModel se habilitan o deshabilitan en función de qué tipo de diagrama esté activo en el editor principal.
- En Eclipse hay un menú **UModel** que equivale al menú **Proyecto** de la versión independiente de UModel. La mayoría de los comandos son iguales que los de la versión independiente, pero el menú **UModel** de Eclipse incluye algunos comandos nuevos que sirven para controlar la sincronización automática:



Volver a sincronizar el proyecto de UModel con el código	Sirve para iniciar la sincronización entre el proyecto de UModel y el código de programa (cuando la última sincronización automática dio lugar a error, por ejemplo).
Sobrescribir el código de programa con el proyecto de UModel	Actualiza el código de programa con los cambios realizados en el proyecto de UModel.
Sobrescribir el proyecto de UModel con el código de programa	Actualiza el proyecto de UModel con los cambios realizados en el código de programa.

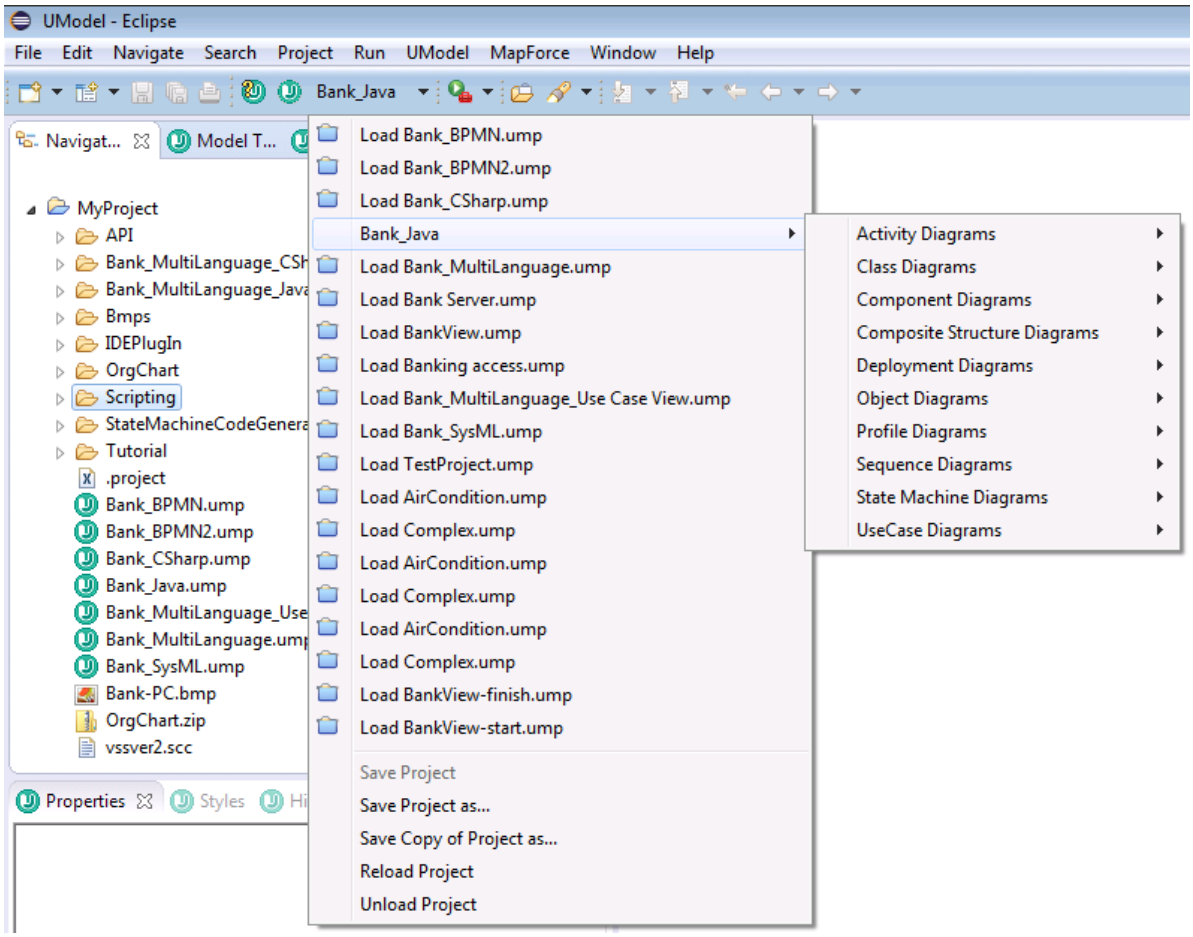
Cancelar sincronización del proyecto de UModel y el código de programa	Sirve para cancelar la operación de sincronización que está en curso. Si no hay ninguna operación de sincronización en curso, este comando se deshabilita.
Sincronizar automáticamente código de programa con proyecto de UModel	Este comando está activado por defecto, es decir, la sincronización tiene lugar automáticamente. Haga clic en el comando para desactivar la sincronización automática.
Sincronizar automáticamente proyecto de UModel con código de programa	Igual que el comando anterior pero en el sentido contrario.

- Los comandos de control de versiones de la edición independiente de UModel, que funcionan con la API del complemento Microsoft Source Control, no funcionan en Eclipse. En su lugar puede utilizar sistemas de control de código fuente de otros autores que sean compatibles con Eclipse.
- Los cuadros de diálogo que se abren al hacer clic en los comandos **UModel | Importar un directorio de código fuente y UModel | Importar proyecto de código fuente** no ofrecen ninguna opción para seleccionar el lenguaje Java. Para importar código Java en un proyecto de Eclipse, utilice el comando de importación estándar de Eclipse (p. ej. **File | Import**).
- Eclipse incluye una barra de herramientas con comandos de Eclipse.



El botón  abre el archivo de ayuda, mientras que el botón  muestra el estado actual del proceso de ingeniería de código (si está en rojo, significa que se produjo un error y debe consultar la ventana Mensajes para obtener más información). Por último, la lista desplegable de la barra de herramienta ofrece varias funciones:

  - Puede cargar o descargar en Eclipse un proyecto de UModel concreto. El proyecto de Eclipse debe incluir un proyecto de UModel como mínimo. De lo contrario, la lista desplegable estará deshabilitada.
  - Cuando hay un proyecto de UModel cargado en Eclipse, esta lista desplegable incluye varios comandos contextuales, incluidos algunos comandos para acceder rápidamente a los diagramas del proyecto que está cargado:



- El editor de scripts (**Herramientas | Editor de scripts**) y el comando **Herramientas | Restaurar barras de herramientas y ventanas** de UModel no funcionan en Eclipse.
- Los comandos de ayuda de UModel están en el menú **Help | Ayuda de UModel** de Eclipse. Para ver la versión del complemento de UModel para Eclipse haga clic en **Help | About Eclipse** y después en el icono de UModel.

13.1 Instalar el complemento de UModel para Eclipse

Requisitos

- Eclipse 2022-09, 2022-06, 2022-03, 2021-12 (http://www.eclipse.org).
- Un Java JRE/JDK para la plataforma de 64 bits
- UModel Enterprise o Professional Edition de 64 bits

Nota: todos los requisitos anteriores deben corresponder a la plataforma de 64 bits. El complemento para Eclipse de versiones anteriores de 32 bits ya no es compatible, aunque es posible que aún funcione.

Una vez haya comprobado que cumple con los requisitos de la lista anterior, puede instalar el paquete de integración de UModel de 64 bits para integrar la aplicación en Eclipse. La integración se puede llevar a cabo al instalar el paquete de integración o de forma manual desde Eclipe una vez haya instalado el paquete de integración. El paquete de integración de UModel se puede descargar en https://www.altova.com/es/components/download.

Nota: debe cerrar Eclipse para poder instalar o desinstalar el paquete de integración de UModel.

Instalar el complemento de UModel para Eclipse

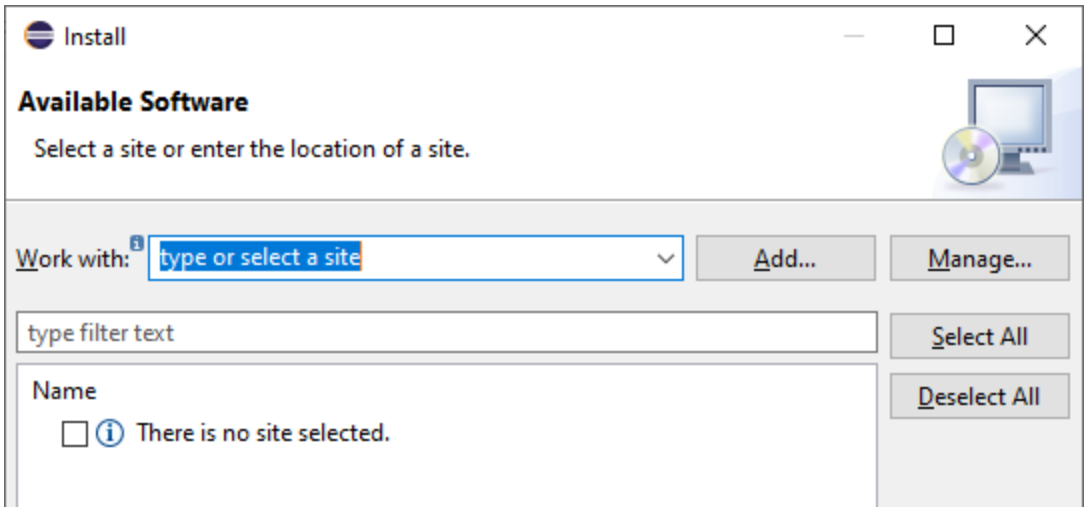
Puede integrar el complemento de UModel en Eclipse ante la instalación del paquete de integración de UModel. Para ello:

1. Ejecute el paquete de integración de UModel para iniciar el asistente de instalación.
2. Cuando el instalador se lo pida seleccione *Instalar el complemento para Eclipse* y haga clic en **Aceptar**.
3. Cuando le pida que escoja cómo quiere integrar el complemento de UModel en Eclipse seleccione *Permitir integración de UModel en Eclipse* y navegue hasta el directorio en el que está el ejecutable de Eclipse (`eclipse.exe`).
4. Haga clic en **Siguiente** y complete la instalación.

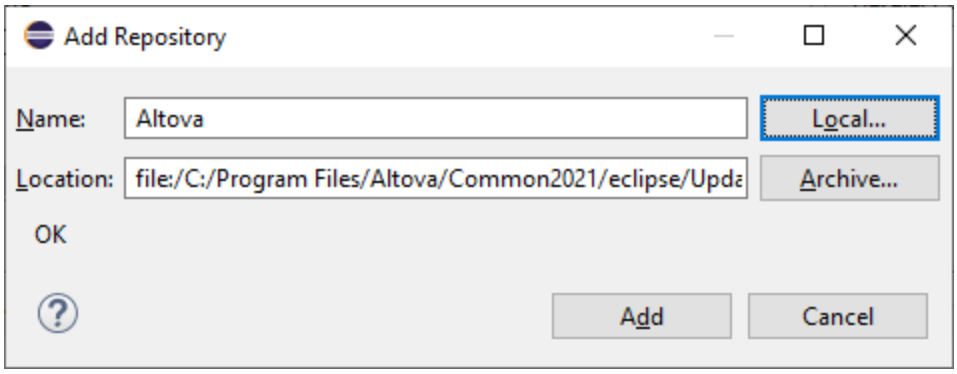
Si elige la integración automática los menús y la perspectiva de UModel se habilitarán en Eclipse la siguiente vez que lo inicie.

Integrar el complemento de UModel para Eclipse manualmente

1. En Eclipse seleccione el comando de menú **Ayuda | Instalar software nuevo**.
2. En el cuadro de diálogo Instalar haga clic en **Agregar**.



3. En el cuadro de diálogo "Agregar repositorio" haga clic en **Local**. Navegue hasta la carpeta `C:\Program Files\Altova\Common2023\eclipse\UpdateSite` y selecciónsela. Elija un nombre para el sitio (por ejemplo, "Altova").

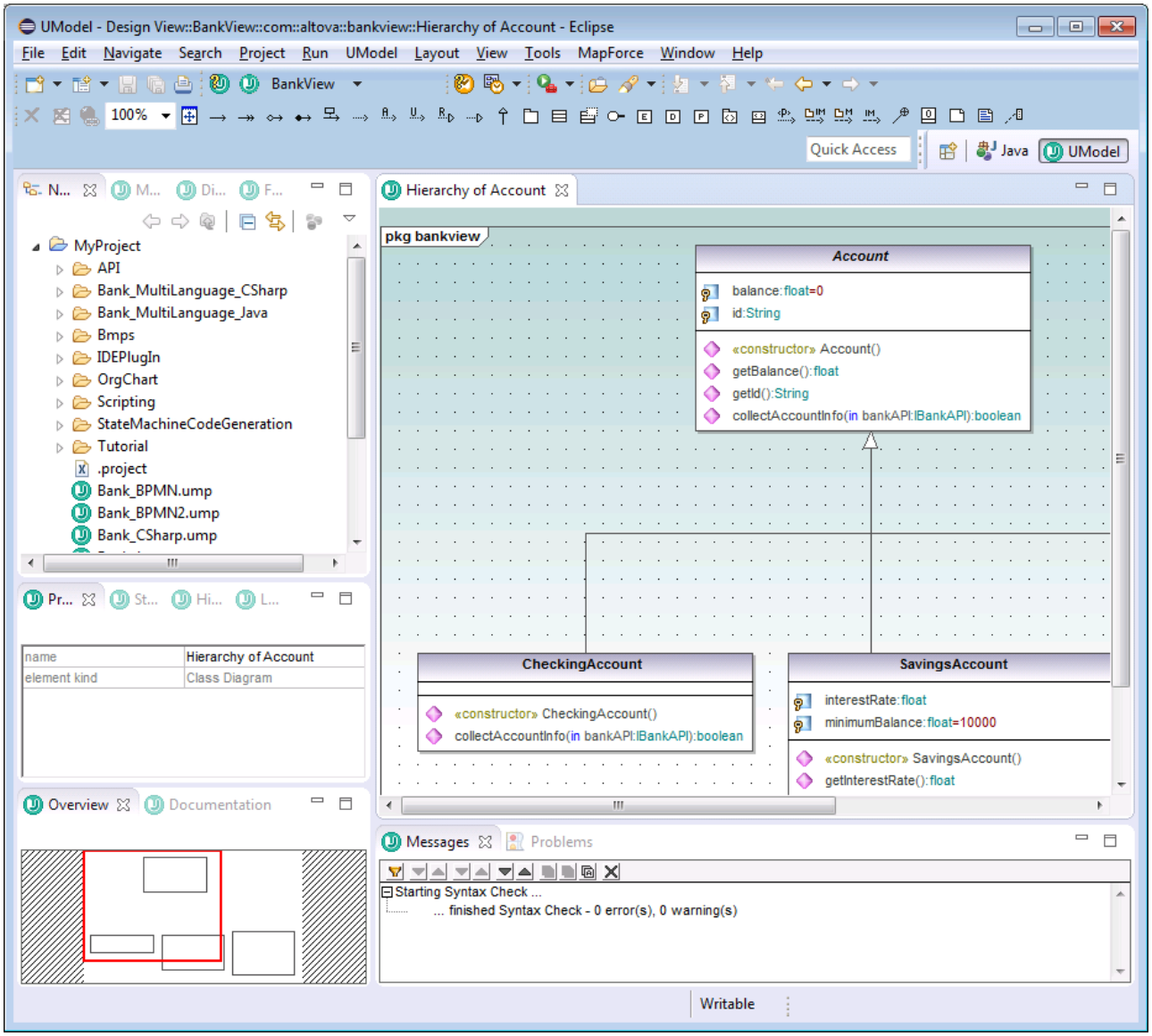


4. Repita los pasos 2-3 del punto anterior pero esta vez elija la carpeta `C:\Program Files\Altova\UModel2023\eclipse\UpdateSite` y un nombre como "Altova UModel".
5. En el cuadro de diálogo seleccione *Solo sitios locales*. A continuación seleccione la carpeta "Altova category" y haga clic en **Siguiente**.
6. Revise los elementos que va a instalar y haga clic en **Siguiente** para continuar.
7. Marque la casilla correspondiente para aceptar el acuerdo de licencia.
8. Haga clic en Finalizar para terminar la instalación.

Nota: si hay algún problema con el complemento (iconos que faltan, por ejemplo), pruebe a iniciar Eclipse desde la línea de comandos con el elemento flag `-clean`.

13.2 La perspectiva UModel

Tras instalar el complemento de UModel para Eclipse se activa la perspectiva UModel. Esta perspectiva se parece por defecto a la interfaz gráfica de la edición independiente de UModel. Para cambiar a la perspectiva UModel haga clic en **Window | Open Perspective | Other** y elija UModel en la lista desplegable. A continuación puede ver un proyecto de UModel en Eclipse, con la perspectiva UModel activada. Se trata del proyecto BankView.ump.



La perspectiva UModel de Eclipse está formada por estos componentes:

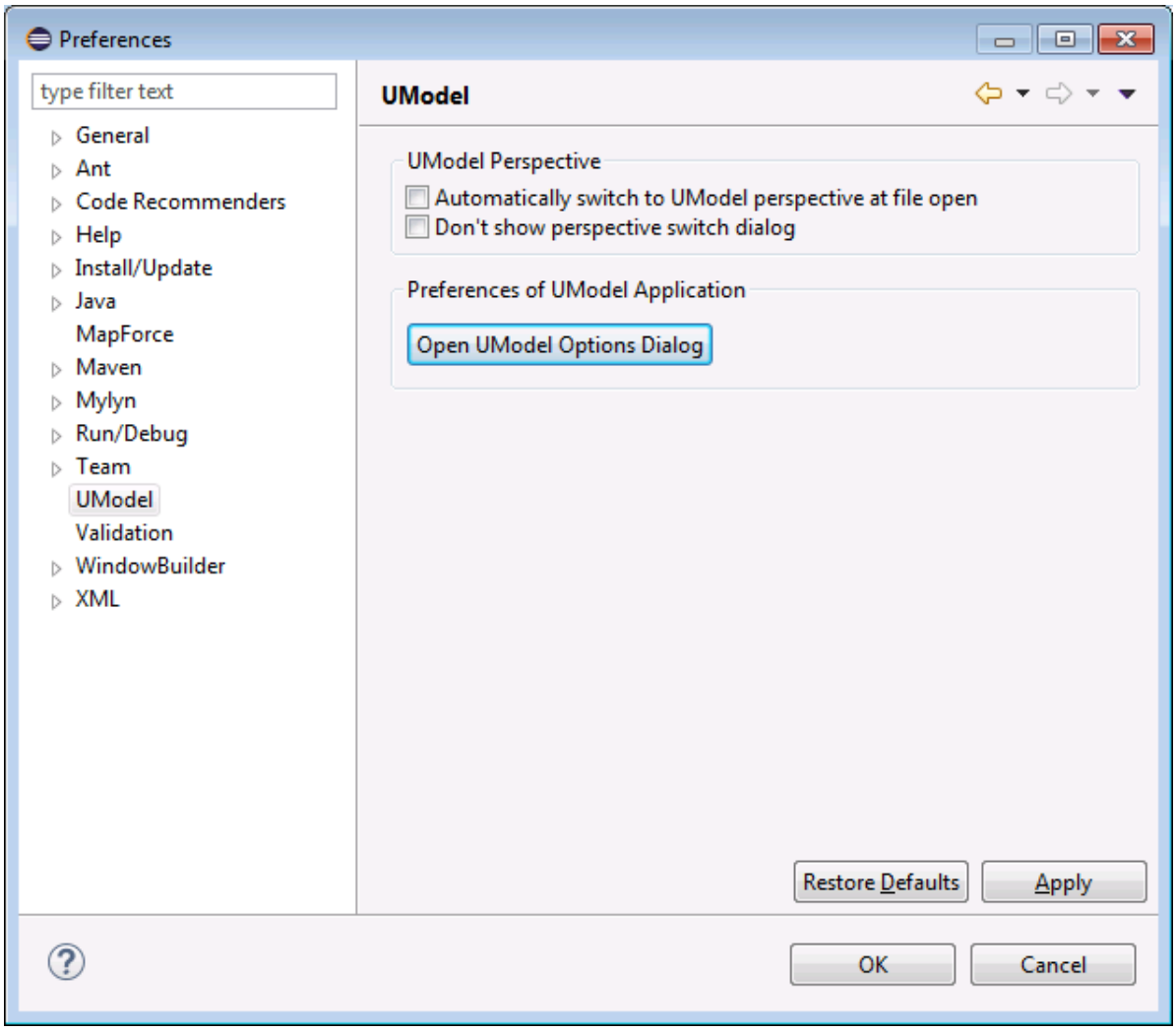
- La ventana de diagramas es un editor de Eclipse. Al igual que en la edición independiente, si hay varios diagramas abiertos, la interfaz muestra varios editores.
- Estos paneles de UModel están disponibles como vistas de Eclipse (situadas por defecto a la izquierda del editor principal):

  - Árbol de diagramas
  - Favoritos
  - Propiedades
  - Estilos
  - Jerarquía
  - Vista general
  - Documentación
  - Capas
- Por último, la ventana Mensajes de UModel también está disponible (y está situada por defecto debajo del editor principal).

El comportamiento de la perspectiva UModel es idéntico al de cualquier otra perspectiva de Eclipse. Es decir, puede cambiar a la perspectiva con el comando **Window | Navigation | Next Perspective**.

Para cambiar las opciones de configuración de la perspectiva UModel:

1. En el menú **Window** haga clic en **Preferences**.
2. En el cuadro de diálogo "Preferences" seleccione **UModel**.



Clic para ampliar

Para personalizar el aspecto de la perspectiva UModel (visibilidad de la barra de herramientas y de los menús, etc.) cambie a la perspectiva UModel y seleccione el comando de menú **Window | Perspective | Customize Perspective**. Para restaurar la configuración predeterminada seleccione **Window | Perspective| Reset Perspective**.

Para abrir una vista concreta de la perspectiva UModel cambie primero a la perspectiva y después seleccione la vista correspondiente en el menú **Window | Show View**.

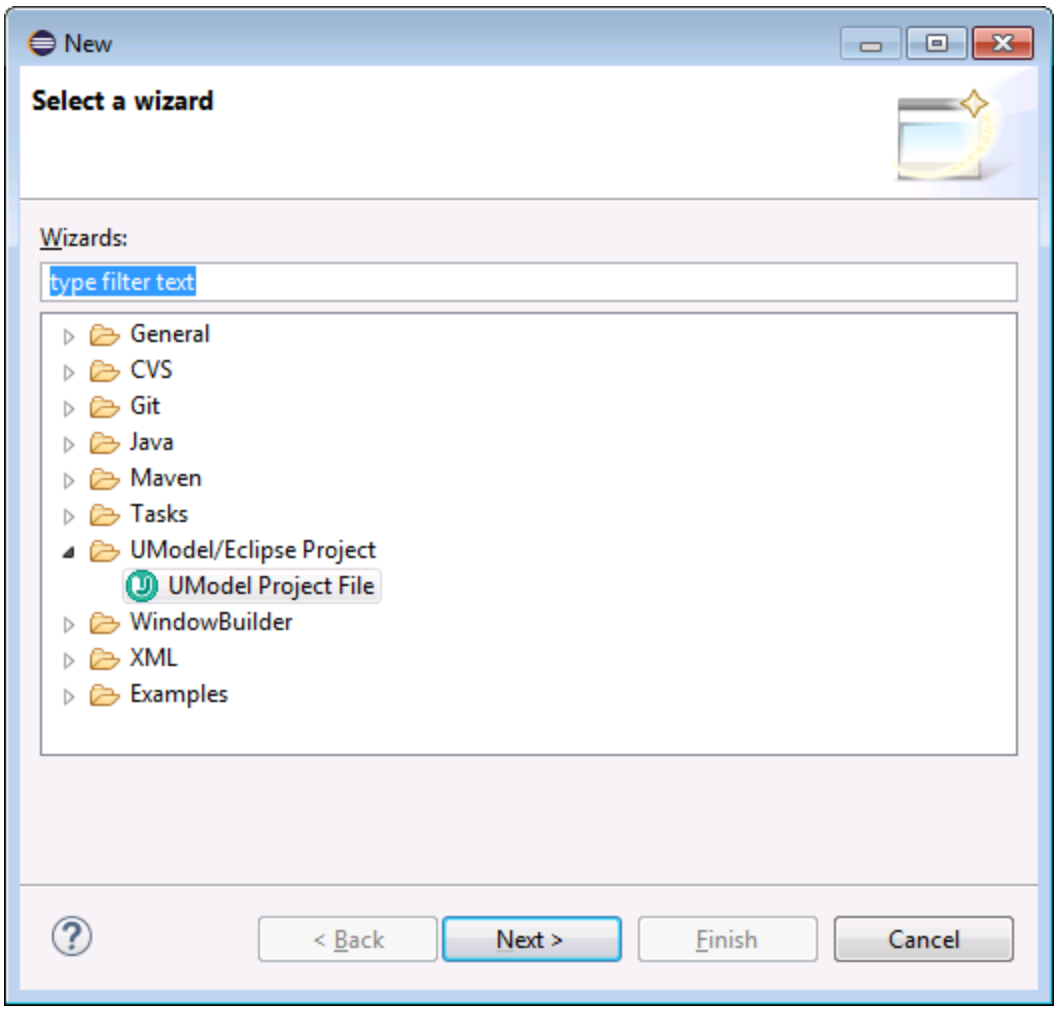
Para obtener más información sobre el comportamiento de las perspectivas consulte la documentación de Eclipse.

13.3 Agregar funciones de UModel a proyectos de Eclipse

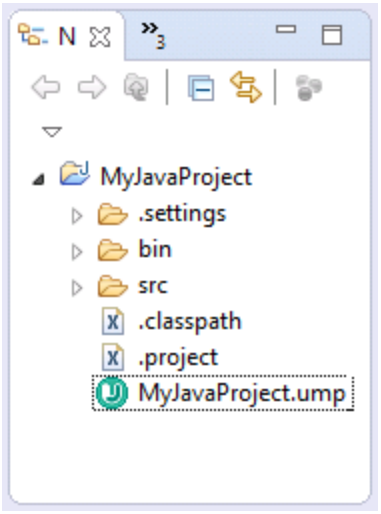
Para poder trabajar con proyectos de UModel (archivos .ump) en el entorno Eclipse, primero es necesario crear o abrir un proyecto de Eclipse (por ejemplo, un proyecto de Java al que desee añadir compatibilidad con UML). En este apartado explicamos cómo crear un proyecto de UModel nuevo dentro de un proyecto de Eclipse. Si desea aprender a importar un proyecto de UModel en un proyecto de Eclipse consulte el siguiente apartado Importar proyectos de UModel.

Siga estas instrucciones para agregar un proyecto de UModel a un proyecto de Eclipse:

1. Cree un proyecto de Eclipse nuevo o abra un proyecto actual con los comandos **File | New | Project** o **File | Open File** respectivamente.
2. En el menú **File** haga clic en **New | Other** y seleccione el tipo de archivo **Archivo de proyecto de UModel** en el cuadro de diálogo.



3. Haga clic en **Next**.
4. Después seleccione una carpeta principal para el proyecto de UModel nuevo y haga clic en **Finish**. El proyecto de UModel nuevo puede verse en la vista Navigator, bajo la carpeta principal seleccionada.

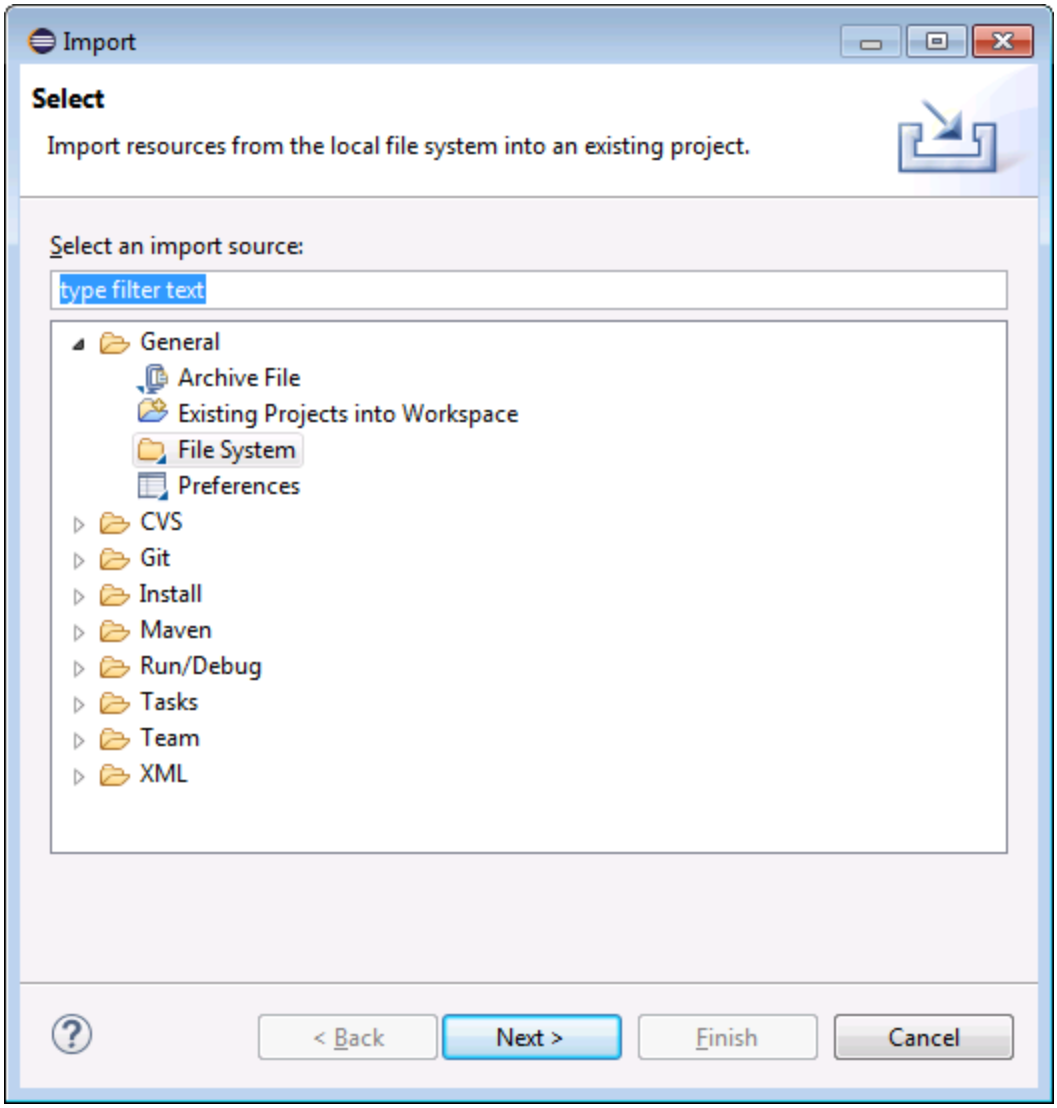


Los proyectos de UModel no se pueden abrir en un editor. Para realizar cambios en el proyecto (p. ej. guardar o cargar su contenido en Eclipse) haga clic con el botón derecho en el archivo .ump y seleccione el comando correspondiente.

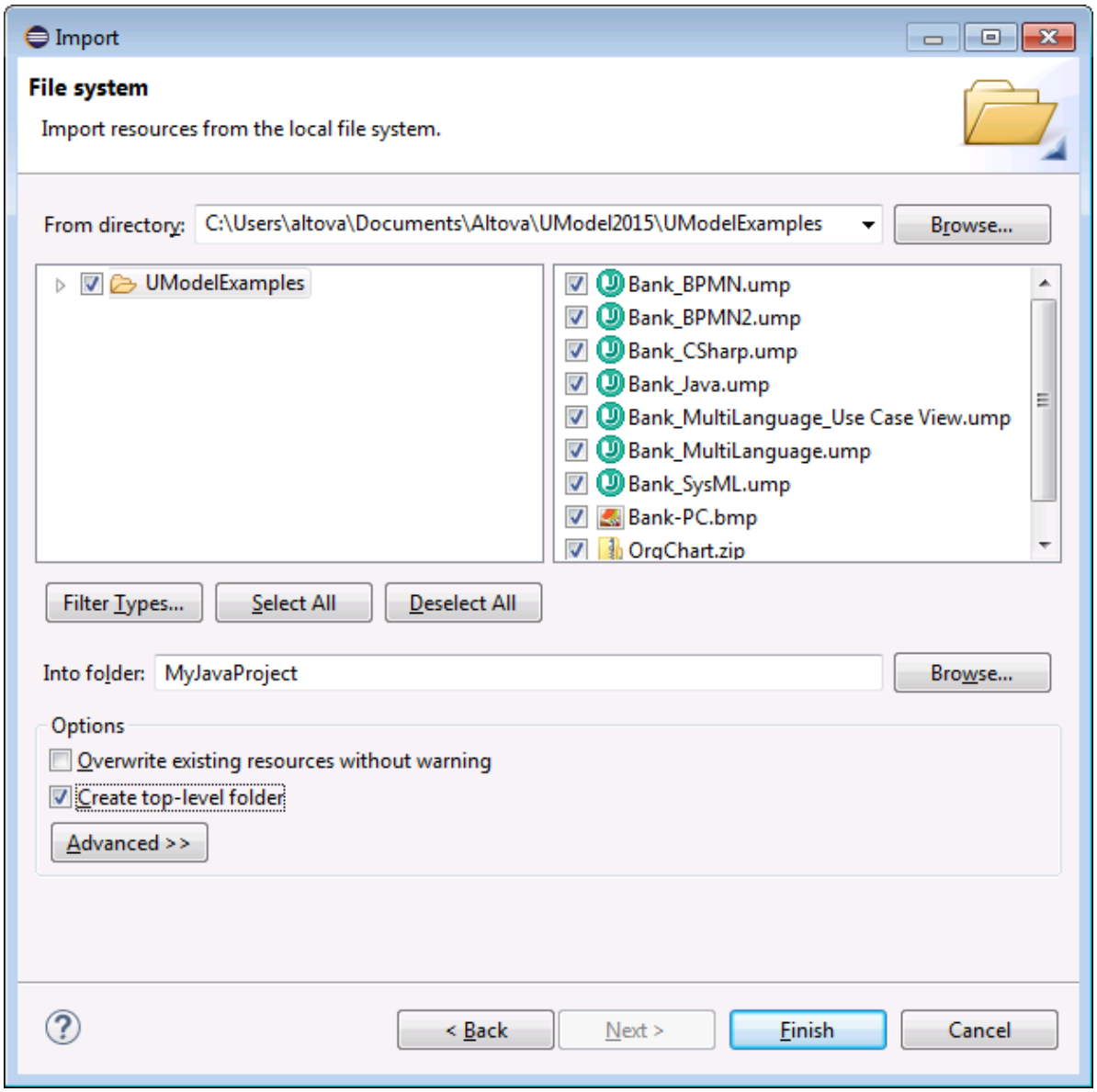
13.4 Importar proyectos de UModel

Para importar proyectos de UModel en Eclipse:

1. Cree un proyecto de Eclipse nuevo o abra un proyecto actual.
2. En el menú **File** haga clic en **Import**.
3. Seleccione **General | File System**.



4. Haga clic en **Next**.
5. Ahora haga clic en **Browse** y seleccione las carpetas del proyecto de UModel que desea importar (p. ej. la carpeta `UModelExamples` de UModel).



6. Haga clic en **Finish**.

13.5 Cargar y descargar proyectos de UModel

Una vez creados o importados, los archivos de proyecto de UModel aparecen en la vista Navigator de Eclipse. Aunque los proyectos de Eclipse pueden incluir varios archivos de proyectos de UModel, en Eclipse puede haber un proyecto de UModel activo (es decir, cargado) como máximo. Hay dos maneras de cargar un proyecto:

- Haciendo clic con el botón derecho en el archivo de proyecto en la vista Navigator y seleccionando **UModel | Cargar**.
- Seleccionando **Cargar Proyecto.ump** en la barra de herramientas de UModel.

Para descargar un proyecto:

- Haciendo clic con el botón derecho en el archivo de proyecto en la vista Navigator y seleccionando **UModel | Descargar**.
- Seleccionando **Descargar proyecto** en la barra de herramientas de UModel.

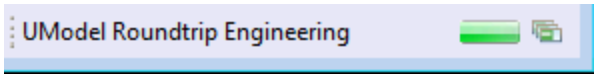
13.6 Funcionamiento de la sincronización automática

La sincronización automática tiene lugar después de añadir las funciones de UModel al proyecto Java. Sincronización automática quiere decir que, cada vez que se edite el código en el entorno Eclipse, el complemento de UModel analiza este código y actualiza el modelo. Asimismo, si se realizan cambios en el modelo (p. ej. si se edita un diagrama), el código se actualiza automáticamente para reflejar estos cambios.

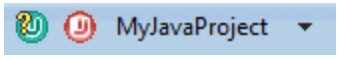
Si el proyecto de UModel contiene el perfil para C# o Visual Basic, la sincronización automática se deshabilita automáticamente para el proyecto y aparece un mensaje a tal efecto. En estos proyectos deberá realizar la sincronización a mano (con los comandos de menú **UModel | Combinar el código de programa con el proyecto de UModel** y **UModel | Combinar el proyecto de UModel con el código de programa**).

Tanto el proceso automático como el manual actualiza los cambios de forma masiva, es decir, para todo el proyecto. En la *Estructura del modelo* de Eclipse no está disponible la opción para combinar o actualizar una sola clase.

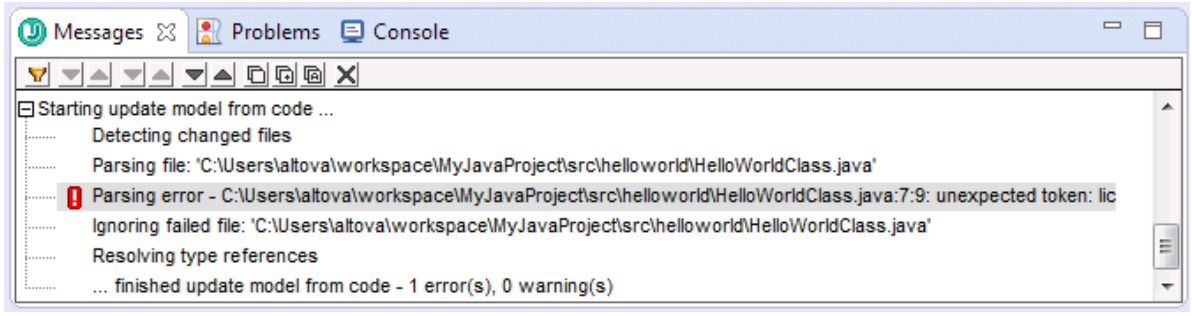
El progreso de la operación de sincronización aparece en la barra de estado de Eclipse:



En caso de que el código no se pueda analizar, el botón de la barra de herramientas Estado de la ingeniería de código estará en rojo. Esto también ocurre si el último proceso de ingeniería inversa o directa detectó un error y cuando la revisión de sintaxis emite un mensaje de error en UModel.



En la ventana Mensajes encontrará todos los detalles sobre el error.



Para abrir el archivo de código fuente que contiene el error basta con hacer clic en la línea correspondiente en la ventana Mensajes. El cursor se coloca automáticamente en la línea que contiene el error.

13.7 Ejemplo: configurar la sincronización automática

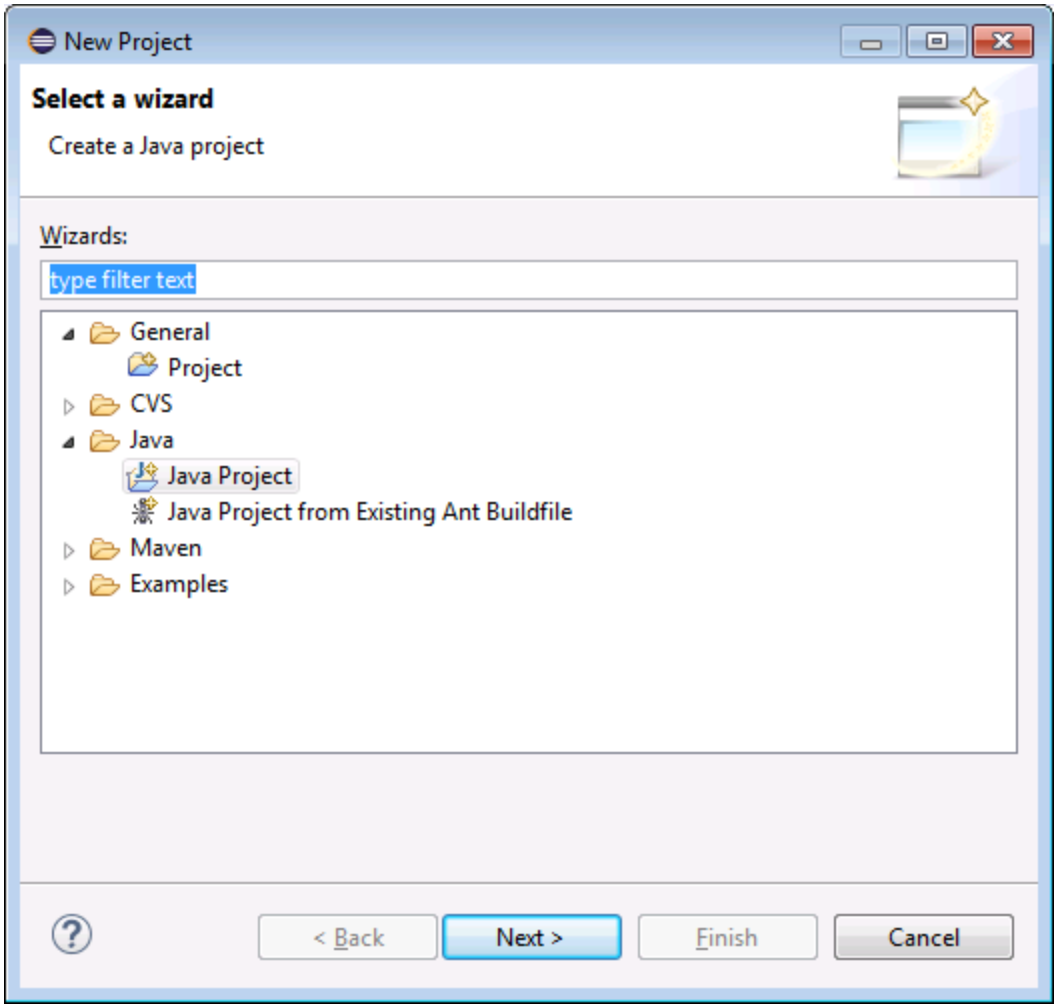
En este ejemplo puede ver cómo se configura la sincronización automática de un proyecto de código y su correspondiente modelo UML. Antes de empezar con el ejemplo, asegúrese de que ya tiene instalado el complemento de UModel para Eclipse y el kit de desarrollo Java (no solo JRE) que necesita Eclipse.

Paso nº1: crear un proyecto de Java nuevo

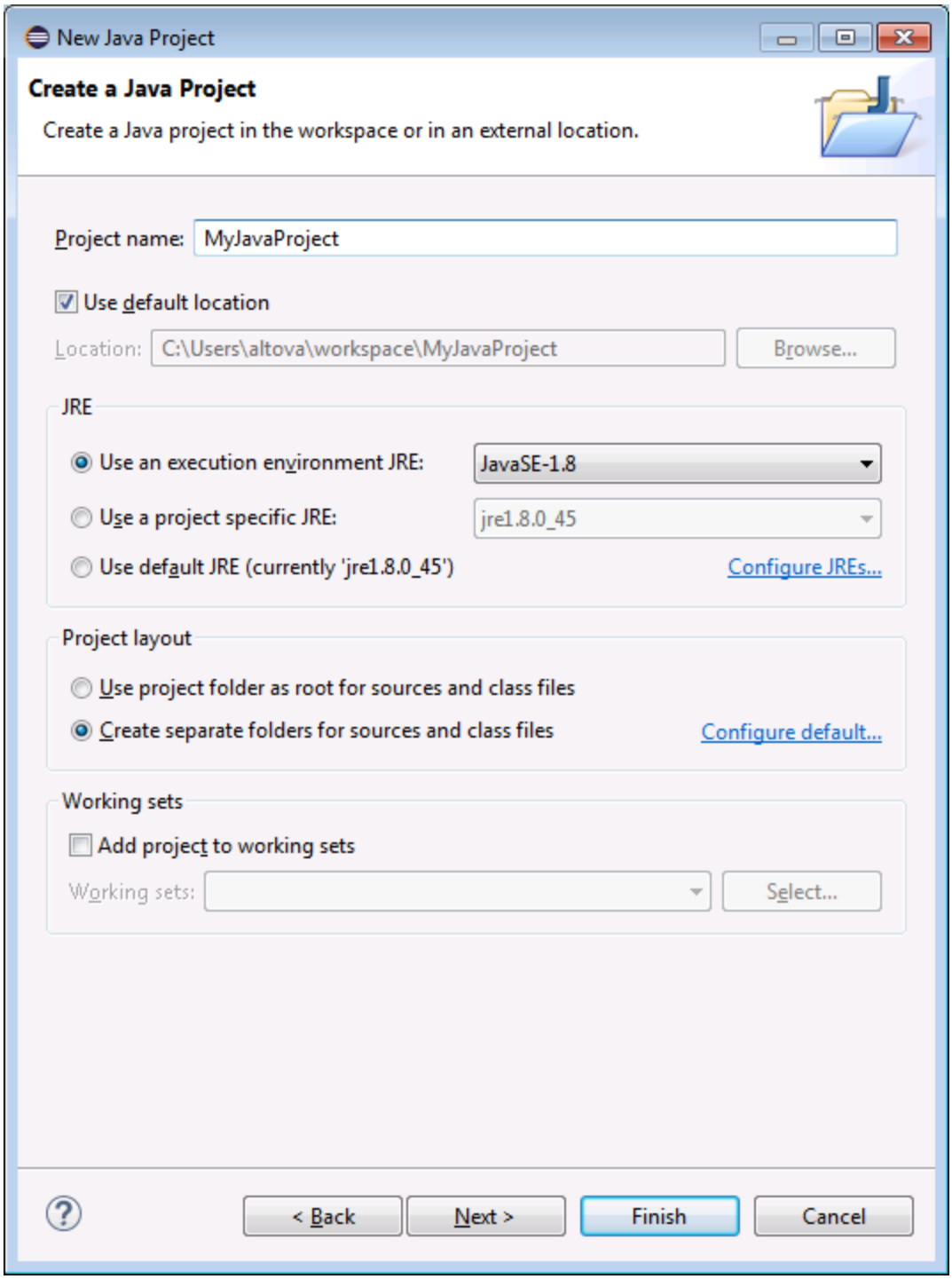
Empezaremos por crear un proyecto de Java nuevo en eclipse. Para este ejemplo utilizamos una aplicación simple que presenta el texto "Hello, World" cuando se ejecuta.

Siga estas instrucciones para crear la aplicación "Hello, World":

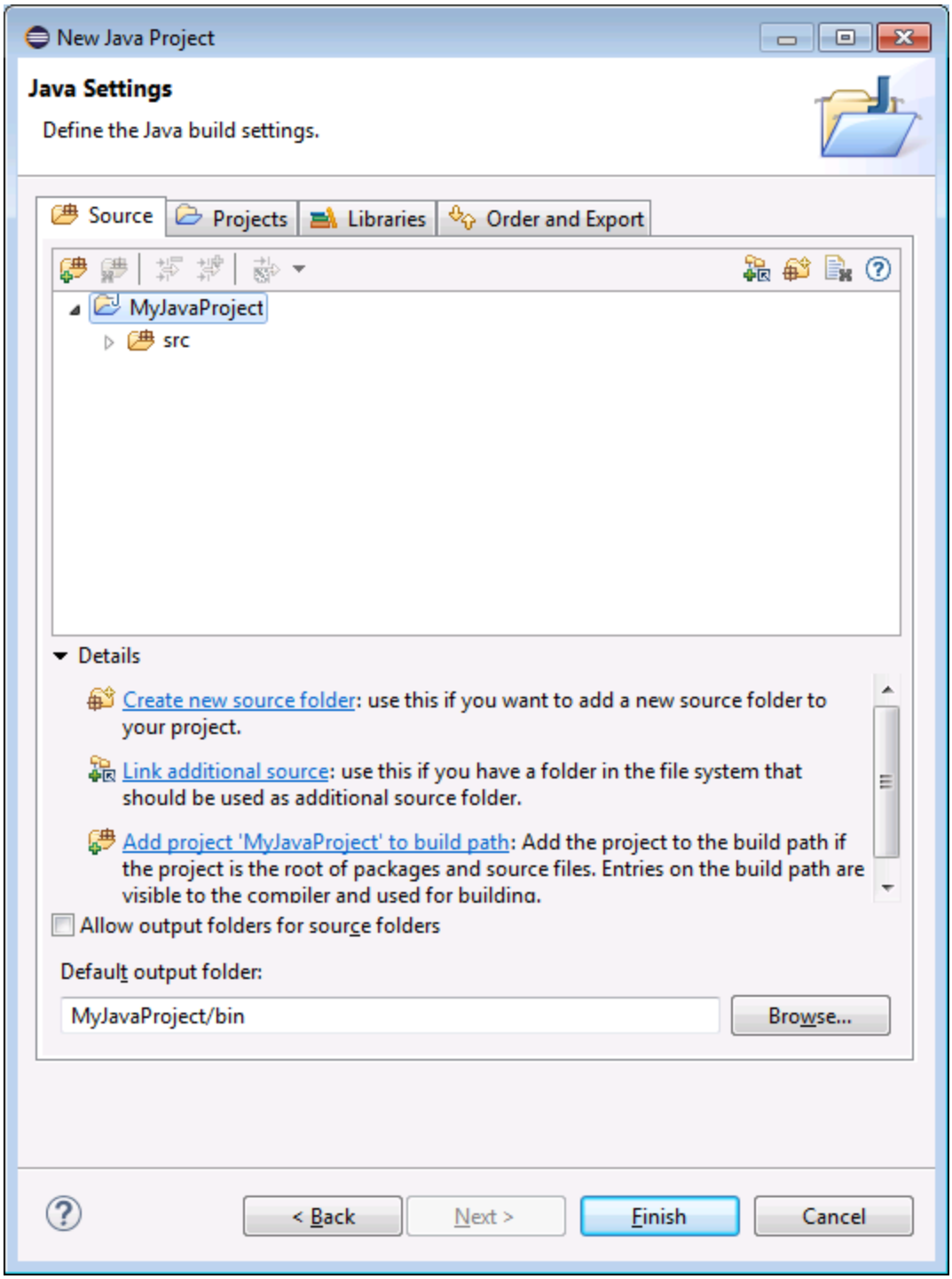
1. Inicie Eclipse y cambie a la perspectiva Java.
2. En el menú **File** haga clic en **New | Project**.



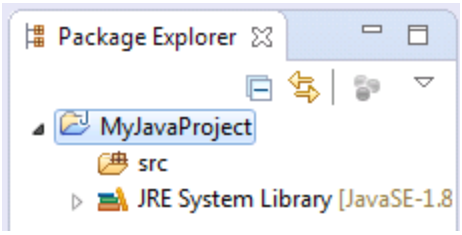
3. Seleccione **Java | Java Project** y después haga clic en **Next**.



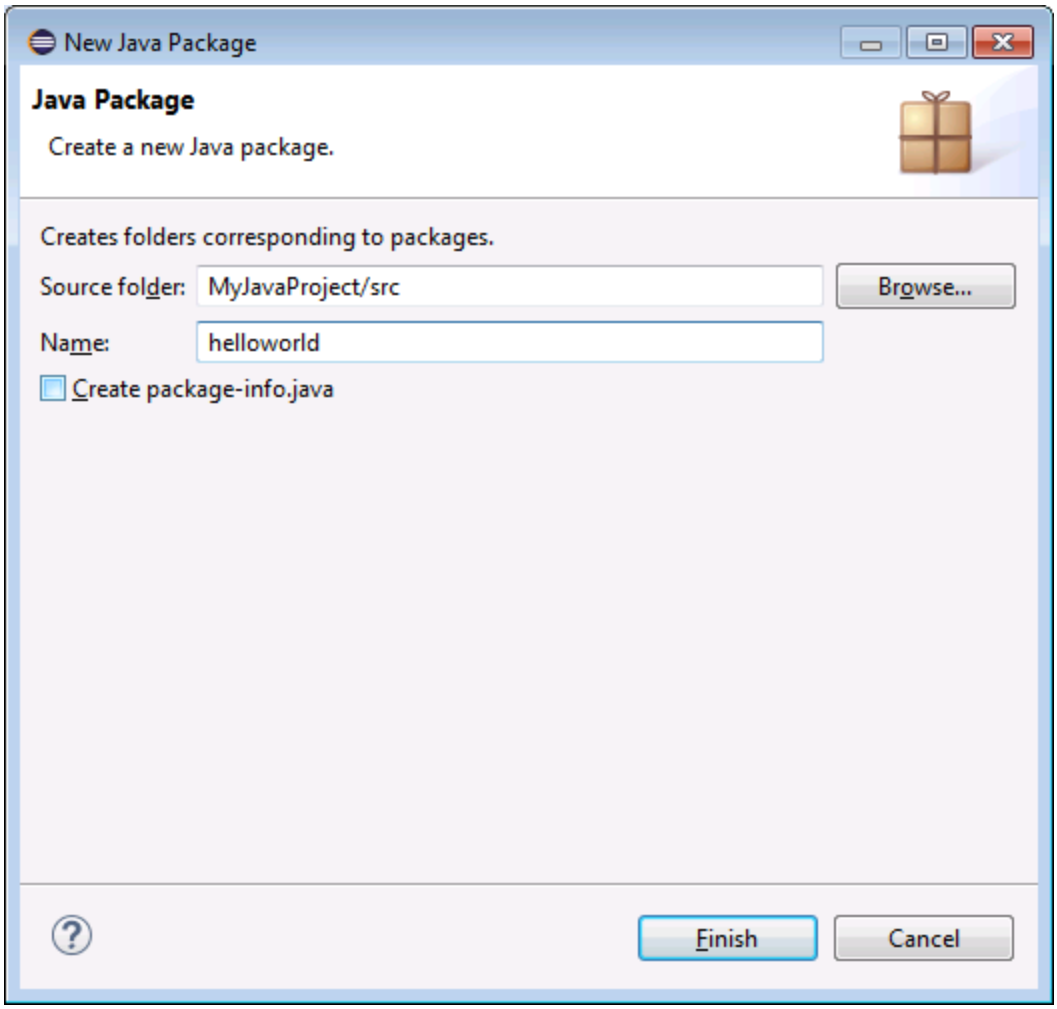
4. Escriba el nombre de proyecto "MyJavaProject" y después haga clic en **Next**.



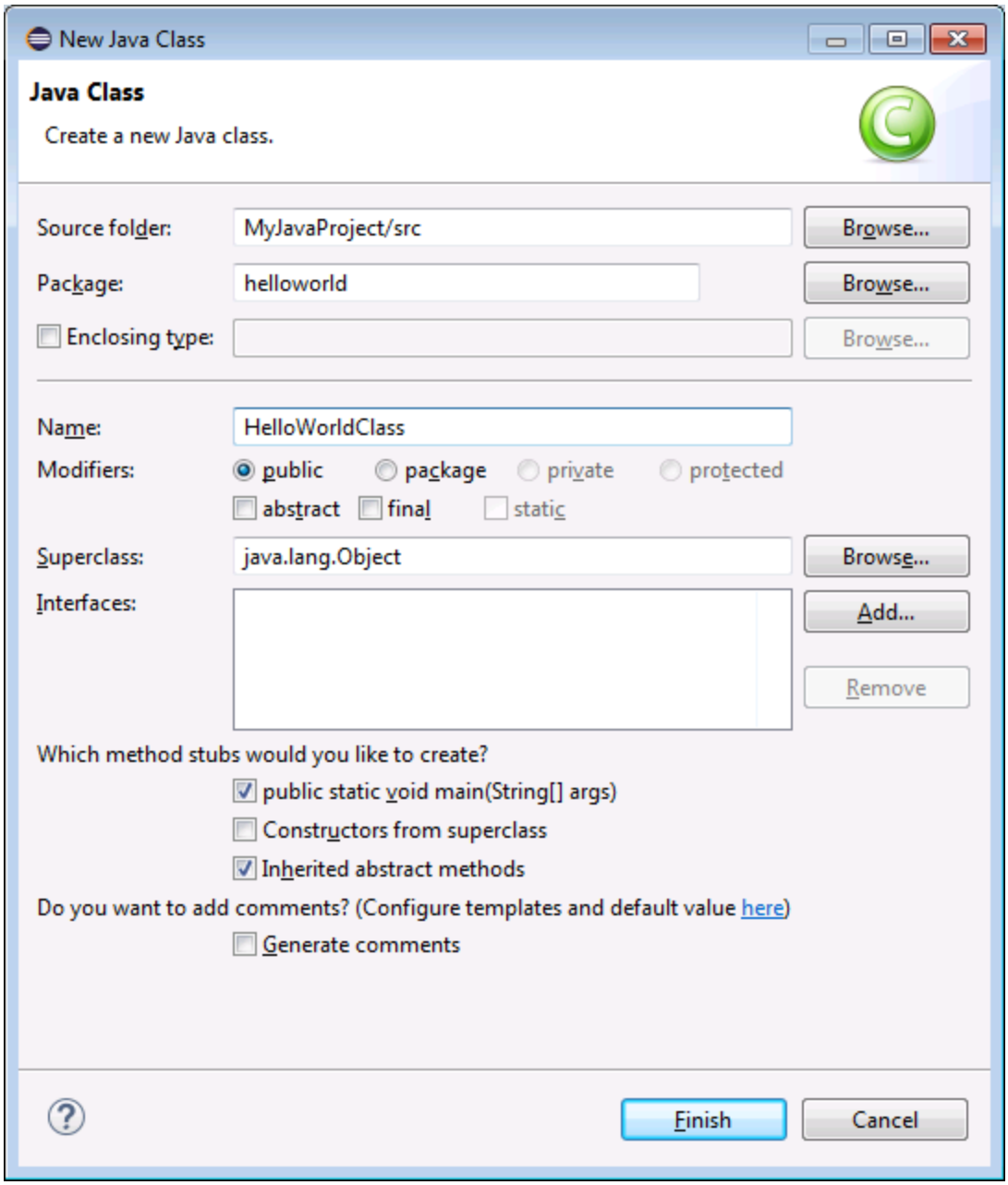
5. Deje las opciones de configuración como están y haga clic en **Finish**. El proyecto nuevo aparece en la ventana Package Explorer.



6. En el menú **File** haga clic en **New | Package**.



7. Escriba el nombre de paquete "helloworld" y haga clic en **Finish**.
8. En el menú **File** haga clic en **New | Class**. Escriba el nombre de clase "HelloWorldClass" y asegúrese de marcar la opción **public static_void main(String[] args)**.

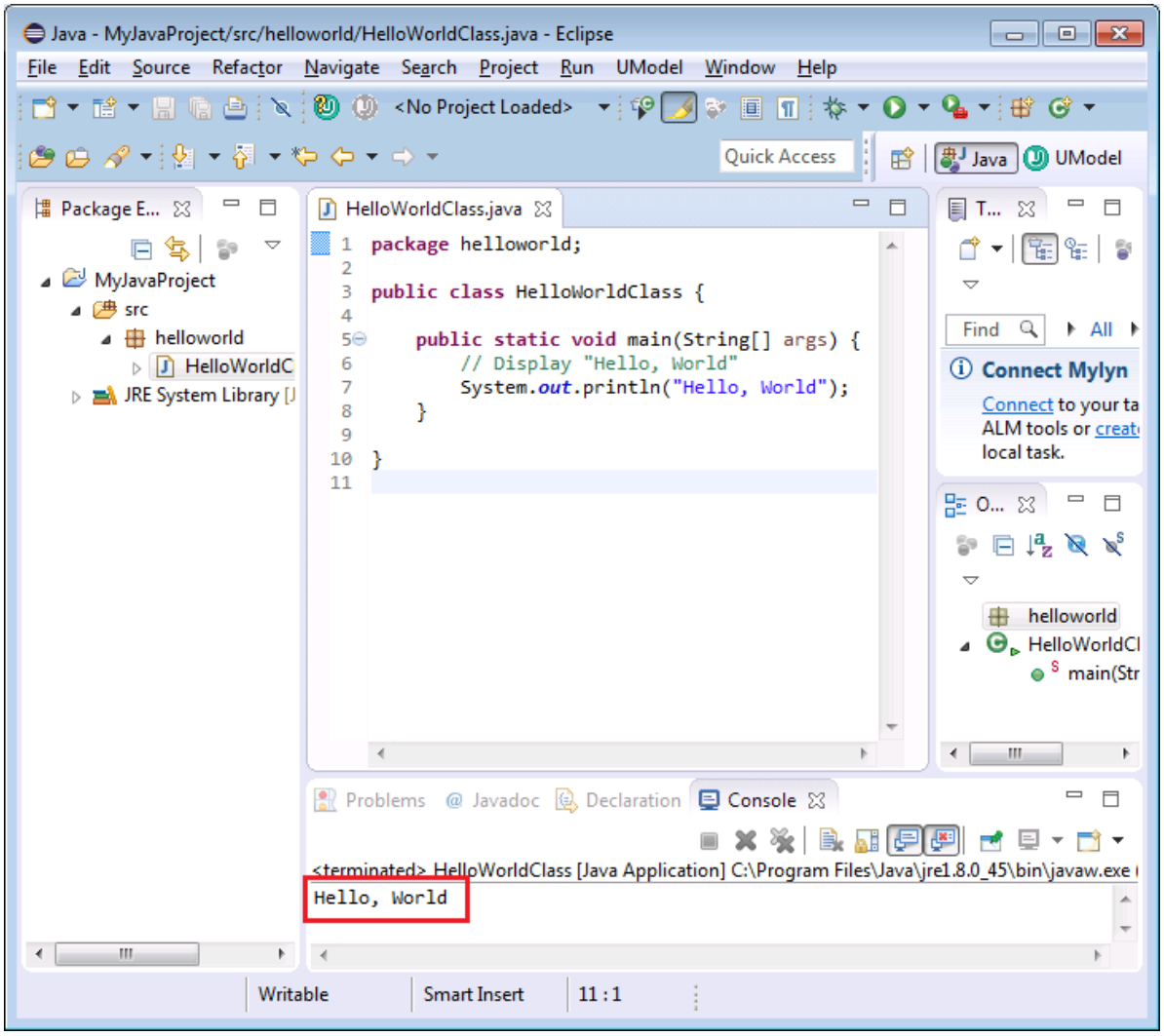


9. Abra el archivo de clases y añada este texto al cuerpo de la clase:

```
package helloworld;

public class HelloWorldClass {

   public static void main(String[] args) {
      // Display "Hello, World"
      System.out.println("Hello, World");
   }
```

```
}
```

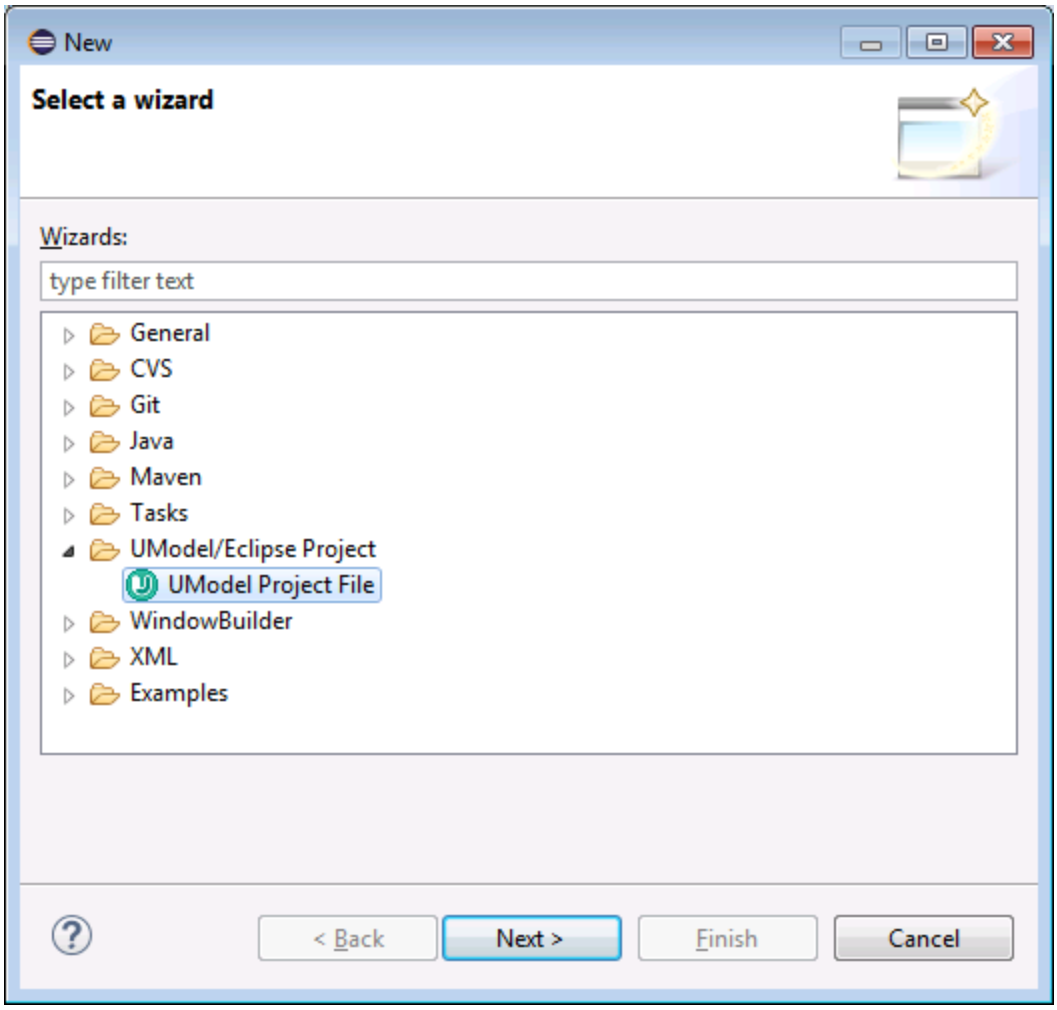
10. Ejecute la aplicación. La vista Console presenta el texto "Hello, World" (*imagen siguiente*).



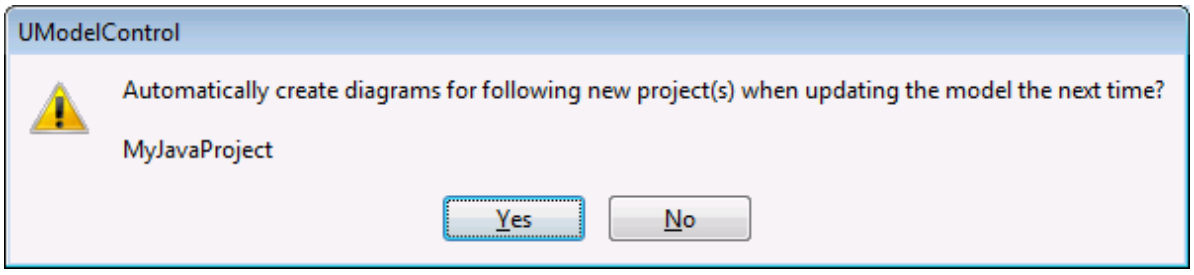
Paso nº2: agregar el proyecto de UModel al proyecto de Java

El siguiente paso consiste en agregar el archivo de proyecto de UModel al proyecto de Eclipse. Esto creará una relación de sincronización entre el modelo y el código.

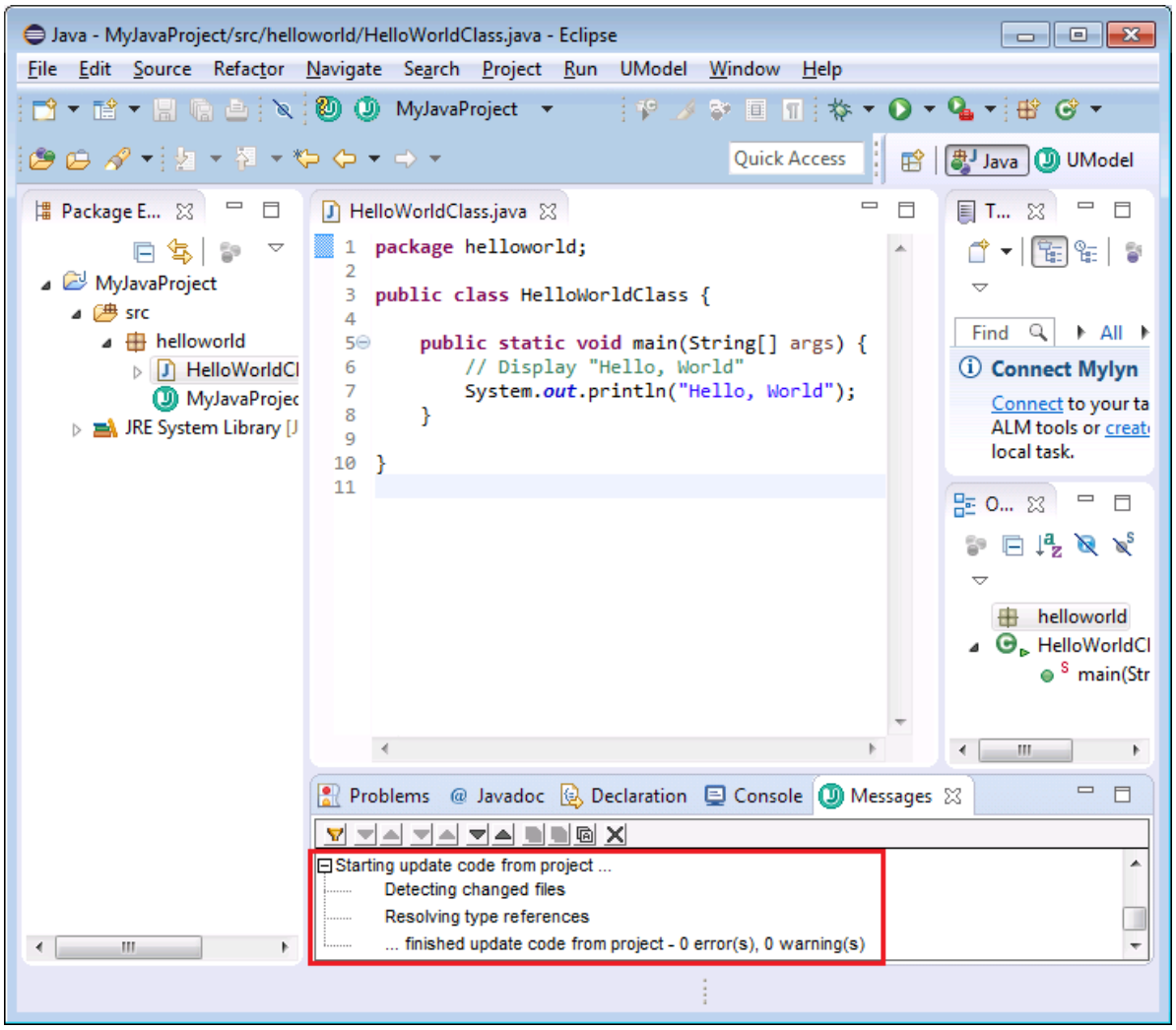
1. En el menú **File** haga clic en **New | Other** y seleccione **Archivo de proyecto de UModel**.



2. Haga clic en **Next**. Cuando deba indicar la ubicación del proyecto de UModel nuevo deje las opciones de configuración predeterminadas como están y haga clic en **Finish**.
3. UModel le preguntará si desea crear diagramas para el proyecto. Haga clic en **Sí**.



4. Siga los pasos del asistente sin cambiar las opciones de configuración predeterminada. Al hacer clic en **Finalizar** el proyecto de UModel nuevo se añade al proyecto de Eclipse y tiene lugar un proceso de sincronización automática entre el código y el modelo. Observe que la ventana Mensajes incluirá información sobre el proceso.

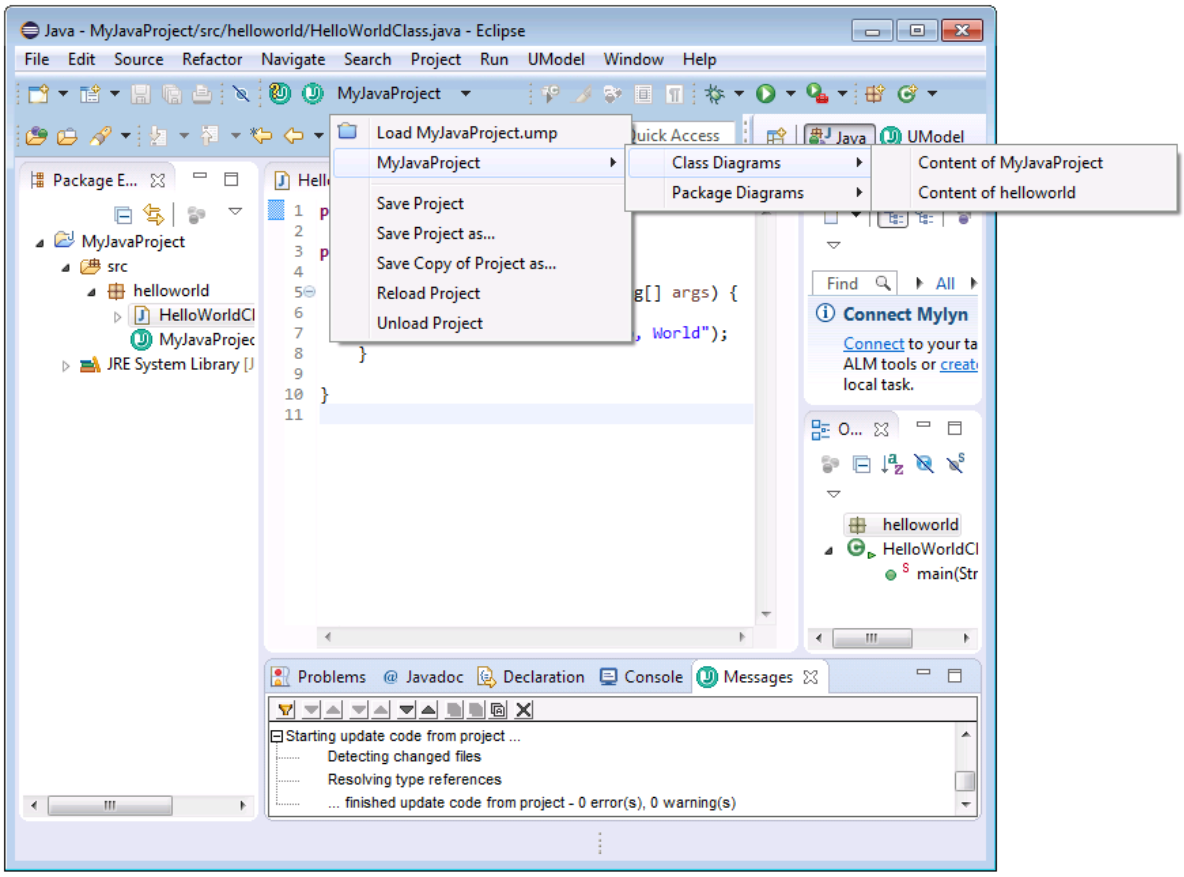


Paso nº3: desencadenar la sincronización automática del código con el modelo

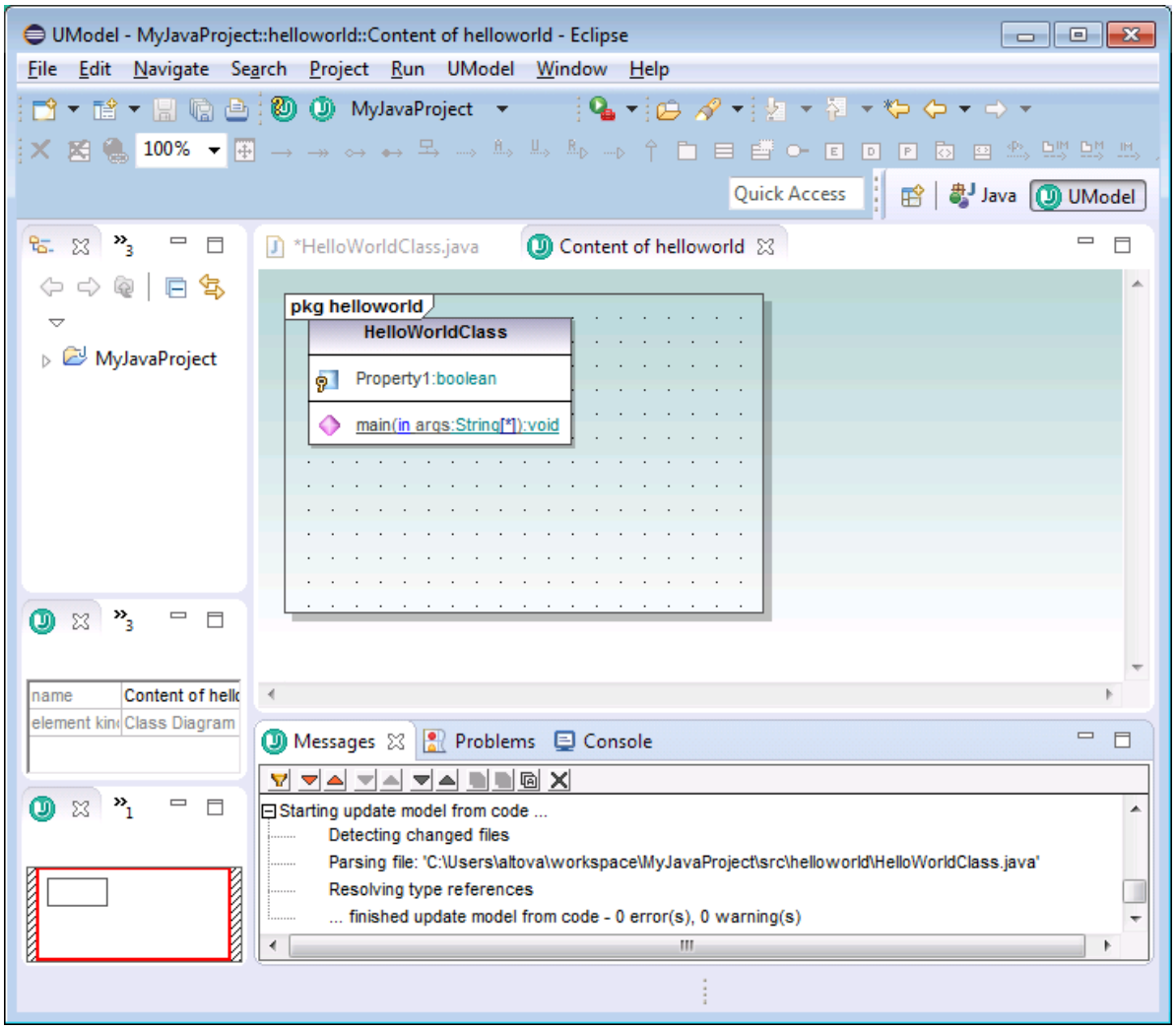
Para desencadenar la sincronización automática entre modelo y código, realizaremos algunos cambios en el diagrama de clases del modelo: vamos a añadir a la clase una propiedad nueva llamada "Property 1" de tipo "Boolean".

Siga estas instrucciones para añadir la propiedad a la clase:

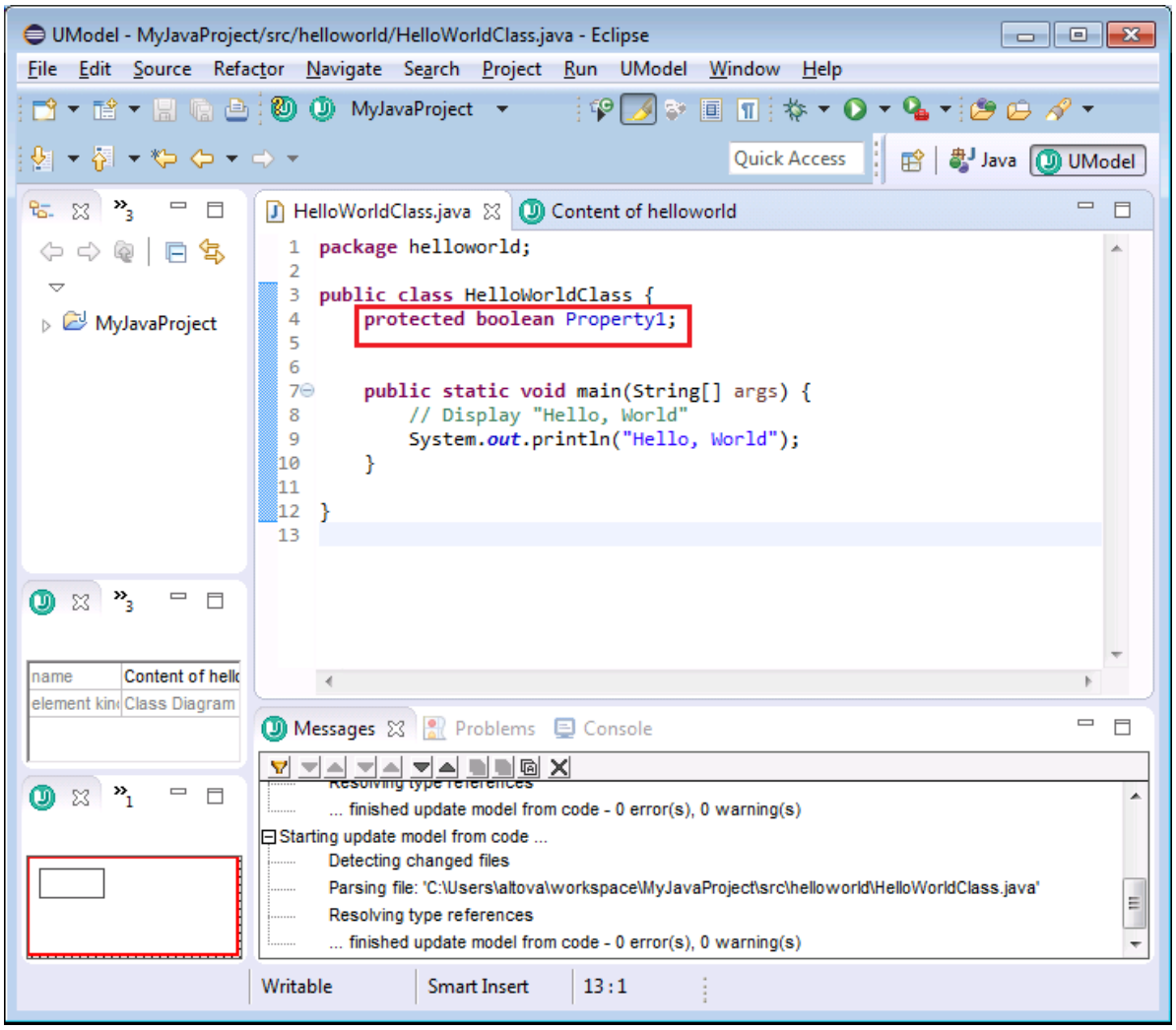
1. En la barra de herramientas de UModel expanda la lista desplegable de proyectos y abra el diagrama de clases "Contenido de helloworld".



2. Haga clic con el botón derecho en la clase y seleccione **Nuevo/a | Propiedad** en el menú contextual.
3. Escriba el nombre de la propiedad ("Property1"), seguido de dos puntos (:) y del tipo ("boolean").

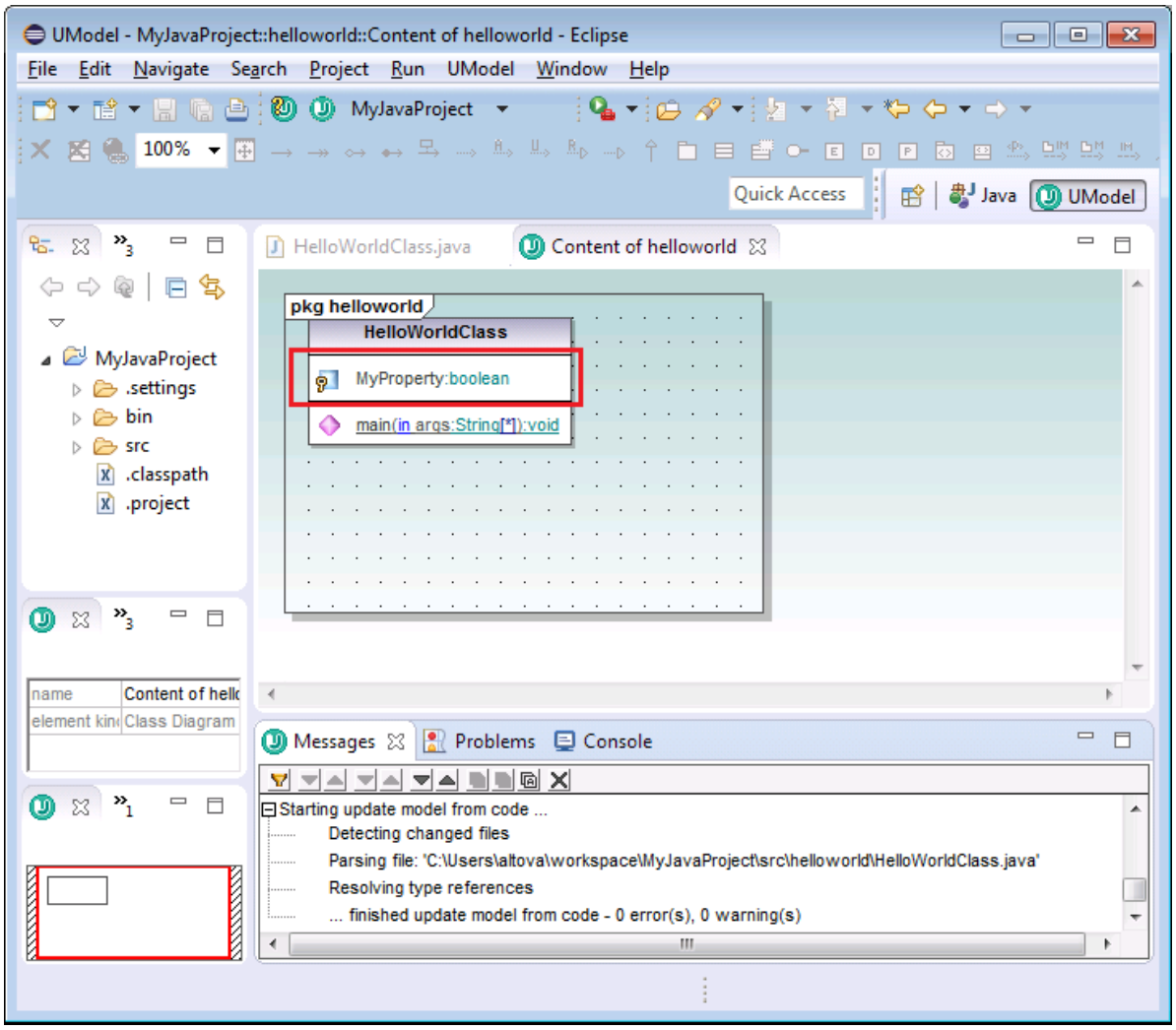


4. Vuelva al editor de código y observe que la propiedad que acaba de añadir aparece ya en el código.



Paso nº4: desencadenar la sincronización automática del modelo con el código

En este paso vamos a desencadenar la sincronización automática en el sentido opuesto (es decir, los cambios realizados en el código se reflejarán en el modelo). Para ello vamos a cambiar el nombre de la propiedad "Property1" en el código por "MyProperty" y después guardaremos el proyecto. Observe que los cambios se propagan al diagrama automáticamente.


UModel - MyJavaProject::helloworld::Content of helloworld - Eclipse
File Edit Navigate Search Project Run UModel Window Help
MyJavaProject
100%
Quick Access
Java
UModel
HelloWorldClass.java
Content of helloworld
pkg helloworld
HelloWorldClass
MyProperty:boolean
main(in args:String[*]):void
MyJavaProject
.settings
bin
src
.classpath
.project
name
Content of hello
element kin
Class Diagram
Messages
Problems
Console
Starting update model from code ...
Detecting changed files
Parsing file: 'C:\Users\altova\workspace\MyJavaProject\src\helloworld\HelloWorldClass.java'
Resolving type references
... finished update model from code - 0 error(s), 0 warning(s)

14 Control de código fuente

La función de control de código fuente de UModel funciona con la API del complemento Microsoft Source Control (antes conocido como MSSCCI), versiones 1.1, 1.2 y 1.3. Gracias a esta API podrá ejecutar comandos de control de código fuente como **Proteger** y **Desproteger** desde UModel directamente con prácticamente cualquier control de código fuente que permita la conexión a clientes nativos o de terceros a través de la API del complemento Microsoft Source Control.

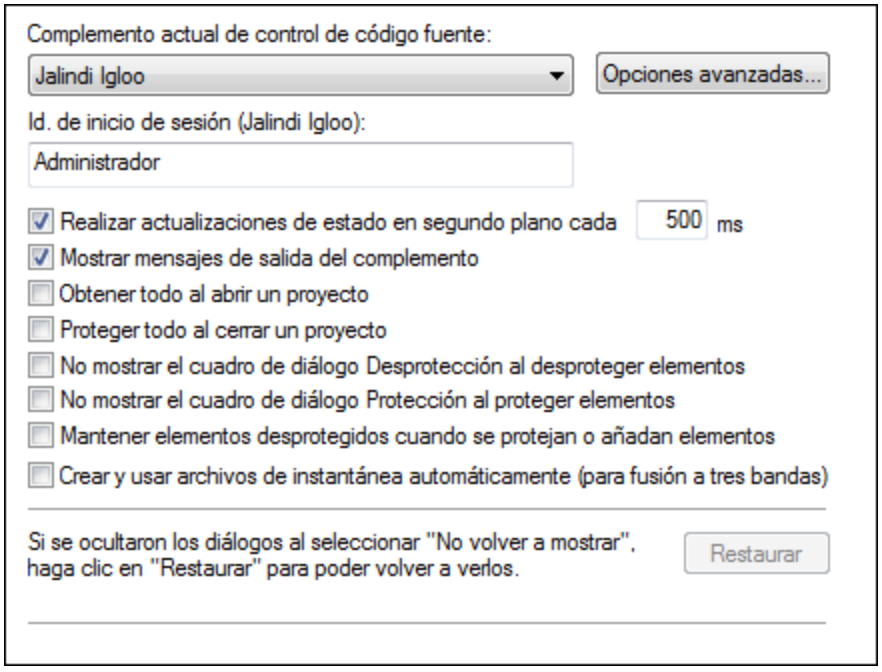
Puede usar cualquier complemento comercial o libre que sea compatible con la API del complemento Microsoft Source Control y puede conectarse a todos los sistemas de control de versiones compatibles (*ver lista de sistemas de control de código fuente compatibles*).

Instalar y configurar el proveedor de control de código fuente

Para ver los proveedores de control de código fuente que están disponibles en el sistema:

1. Haga clic en **Opciones** en el menú **Herramientas**.
2. Haga clic en la sección *Control de código fuente*.

Los complementos de control de código fuente que sean compatibles con la API del complemento Microsoft Source Control aparecerán en la lista desplegable *Complemento actual de control de código fuente*.



Si no se encuentra ningún complemento compatible en el sistema, aparece este mensaje:

```
"No se Registration of installed source control providers could not be found or is
incomplete."
```

Algunos sistemas de control de código fuente no instalan el complemento de control de código fuente automáticamente. En este caso deberá instalar el complemento por separado. UModel espera que los

complementos compatibles con la API del complemento Microsoft Source Code Control estén instalados bajo esta entrada del registro del sistema operativo:

```
HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\InstalledSCCProviders
```

Tras la instalación, el complemento aparecerá automáticamente en la lista de complementos disponibles de UModel.

Acceso a los comandos de control de código fuente

Los comandos para trabajar con el control de código fuente están en el menú **Proyecto | Control de código fuente**.

Problemas de rendimiento y recursos

Algunas bases de datos de control de código fuente de gran tamaño pueden crear problemas con recursos y de rendimiento cuando realicen actualizaciones automáticas de estado en segundo plano.

Para aumentar la velocidad del sistema puede deshabilitar (o aumentar el intervalo de) la opción *Realizar actualizaciones de estado en segundo plano cada....segundos* de la sección *Control de código fuente* del cuadro de diálogo "Opciones" (**Herramientas | Opciones**).

Nota: la versión de 64 bits de UModel es compatible automáticamente con todos los programas de control de código fuente de 32 bits que se enumeran en esta documentación. Cuando usa una versión de 64 bits de UModel con un programa de control de código fuente de 32 bits, la opción *Realizar actualizaciones de estado en segundo plano cada....segundos* se deshabilita automáticamente y no se puede seleccionar.

Comparación de datos con Altova DiffDog

Muchos sistemas de control de código fuente (como Git y TortoiseSVN) se pueden configurar para usar la herramienta de comparación Altova DiffDog. Para más información consulte la documentación de Altova DiffDog y el sitio web de Altova.

14.1 Sistemas de control de código fuente compatibles

A continuación puede ver una lista con todos los servidores de control de código fuente compatibles con UModel, junto con sus correspondientes clientes de control de código fuente. La lista está ordenada alfabéticamente.

Notas:

- Altova ha implementado la API del complemento Microsoft Source Control (versiones 1.1, 1.2 y 1.3) en UModel y ha probado la compatibilidad con los controladores y sistemas de control de versiones de la lista que aparece a continuación. Altova seguirá ofreciendo compatibilidad con estos productos cuando se actualicen.
- Los clientes de control de código fuente que no aparecen en la lista pero que implementan la API del complemento Microsoft Source Control también deberían funcionar con UModel.

Sistema de control de código fuente	Clientes de control de código fuente
AccuRev 4.7.0 Windows	AccuBridge for Microsoft SCC 2008.2
Bazaar 1.9 Windows	Aigenta Unified SCC 1.0.6
Borland StarTeam 2008	Borland StarTeam Cross-Platform Client 2008 R2
Codice Software Plastic SCM Professional 2.7.127.10 (Server)	Codice Software Plastic SCM Professional 2.7.127.10 (SCC Plugin)
Collabnet Subversion 1.5.4	• Aigenta Unified SCC 1.0.6 • PushOK SVN SCC 1.5.1.1 • PushOK SVN SCC x64 versión 1.6.3.1 • TamTam SVN SCC 1.2.24
ComponentSoftware CS-RCS (PRO) 5.1	ComponentSoftware CS-RCS (PRO) 5.1
Dynamsoft SourceAnywhere for VSS 5.3.2 Standard/Professional Server	Dynamsoft SourceAnywhere for VSS 5.3.2 Client
Dynamsoft SourceAnywhere Hosted	Dynamsoft SourceAnywhere Hosted Client (22252)
Dynamsoft SourceAnywhere Standalone 2.2 Server	Dynamsoft SourceAnywhere Standalone 2.2 Client
Git	PushOK GIT SCC plug-in (consulte Control de código fuente con Git)
IBM Rational ClearCase 7.0.1 (LT)	IBM Rational ClearCase 7.0.1 (LT)
March-Hare CVSNT 2.5 (2.5.03.2382)	Aigenta Unified SCC 1.0.6
March-Hare CVS Suite 2008	• Jalindi Igloo 1.0.3 • March-Hare CVS Suite Client 2008 (3321) • PushOK CVS SCC NT 2.1.2.5 • PushOK CVS SCC x64 versión 2.2.0.4 • TamTam CVS SCC 1.2.40

Sistema de control de código fuente	Clientes de control de código fuente
Mercurial 1.0.2 for Windows	Sergey Antonov HgSCC 1.0.1
Microsoft SourceSafe 2005 with CTP	Microsoft SourceSafe 2005 with CTP
Microsoft Visual Studio Team System 2008/2010 Team Foundation Server	Microsoft Team Foundation Server 2008/2010 MSSCCI Provider
Perforce 2008 P4S 2008.1	Perforce P4V 2008.1
PureCM Server 2008/3a	PureCM Client 2008/3a
QSC Team Coherence Server 7.2.1.35	QSC Team Coherence Client 7.2.1.35
Reliable Software Code Co-Op 5.1a	Reliable Software Code Co-Op 5.1a
Seapine Surround SCM Client/Server for Windows 2009.0.0	Seapine Surround SCM Client 2009.0.0
Serena Dimensions Express/CM 10.1.3 for Win32 Server	Serena Dimensions 10.1.3 for Win32 Client
Softimage Alienbrain Server 8.1.0.7300	Softimage Alienbrain Essentials/Advanced Client 8.1.0.7300
SourceGear Fortress 1.1.4 Server	SourceGear Fortress 1.1.4 Client
SourceGear SourceOffsite Server 4.2.0	SourceGear SourceOffsite Client 4.2.0 (Windows)
SourceGear Vault 4.1.4 Server	SourceGear Vault 4.1.4 Client
VisualSVN Server 1.6	• Aigenta Unified SCC 1.0.6 • PushOK SVN SCC 1.5.1.1 • PushOK SVN SCC x64 versión 1.6.3.1 • TamTam SVN SCC 1.2.24

14.2 Comandos de control de código fuente

En los apartados siguientes utilizamos Visual SourceSafe para explicar las características de control de código fuente de UModel. En todos los ejemplos utilizamos el proyecto de UModel `Bank_CSharp.ump` y sus archivos de código asociados, que están disponibles en la carpeta C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples. No se debe confundir el proyecto de control de código fuente con el proyecto de UModel. Los proyectos de control de código fuente dependen de directorios, mientras que los proyectos de UModel son construcciones lógicas sin dependencias de directorio directas.

Hay varias maneras de acceder a los comandos de control de código fuente:

- Desde el comando de menú **Proyecto | Control de código fuente**.
- Desde el **menú contextual** que aparece al hacer clic con el botón derecho en elementos del panel Estructura del modelo.
- Con los botones de la barra de herramientas Control de código fuente. Para activar esta barra de herramientas haga clic en **Herramientas | Personalizar | Barras de herramientas**.

Estos comandos son los de la edición independiente de UModel. En el complemento de UModel para Visual Studio y Eclipse están disponibles las funciones y comandos de control de código fuente de dichos entornos de desarrollo.

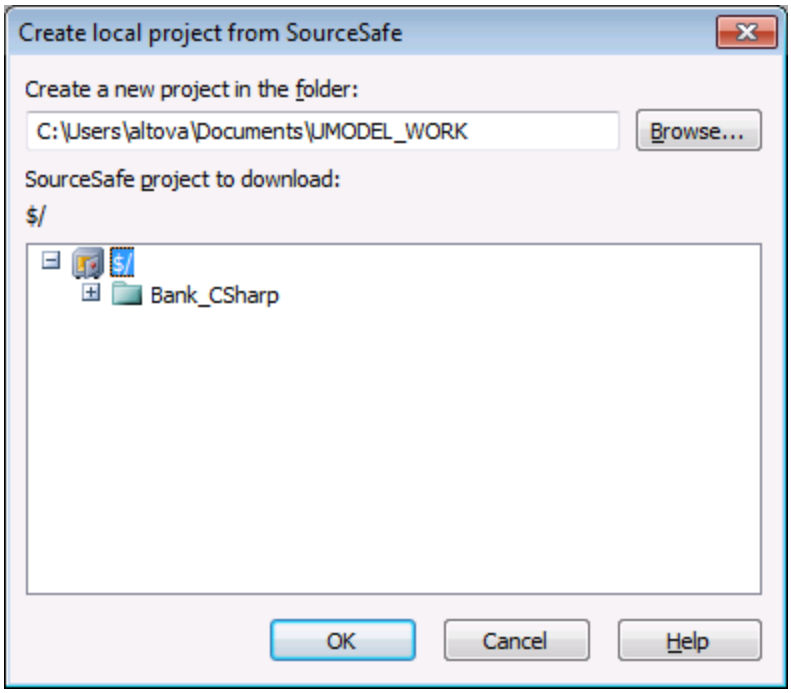
Abrir desde el control de código fuente
Habilitar control de código fuente
Obtener la versión más reciente
Obtener
Obtener carpetas
Desproteger
Proteger
Anular desprotección
Agregar al control de código fuente
Quitar del control de código fuente
Compartir desde el control de código fuente
Mostrar historial
Mostrar diferencias
Mostrar propiedades
Actualizar estado
Administrador del control de código fuente
Cambiar control de código fuente

14.2.1 Abrir desde el control de código fuente

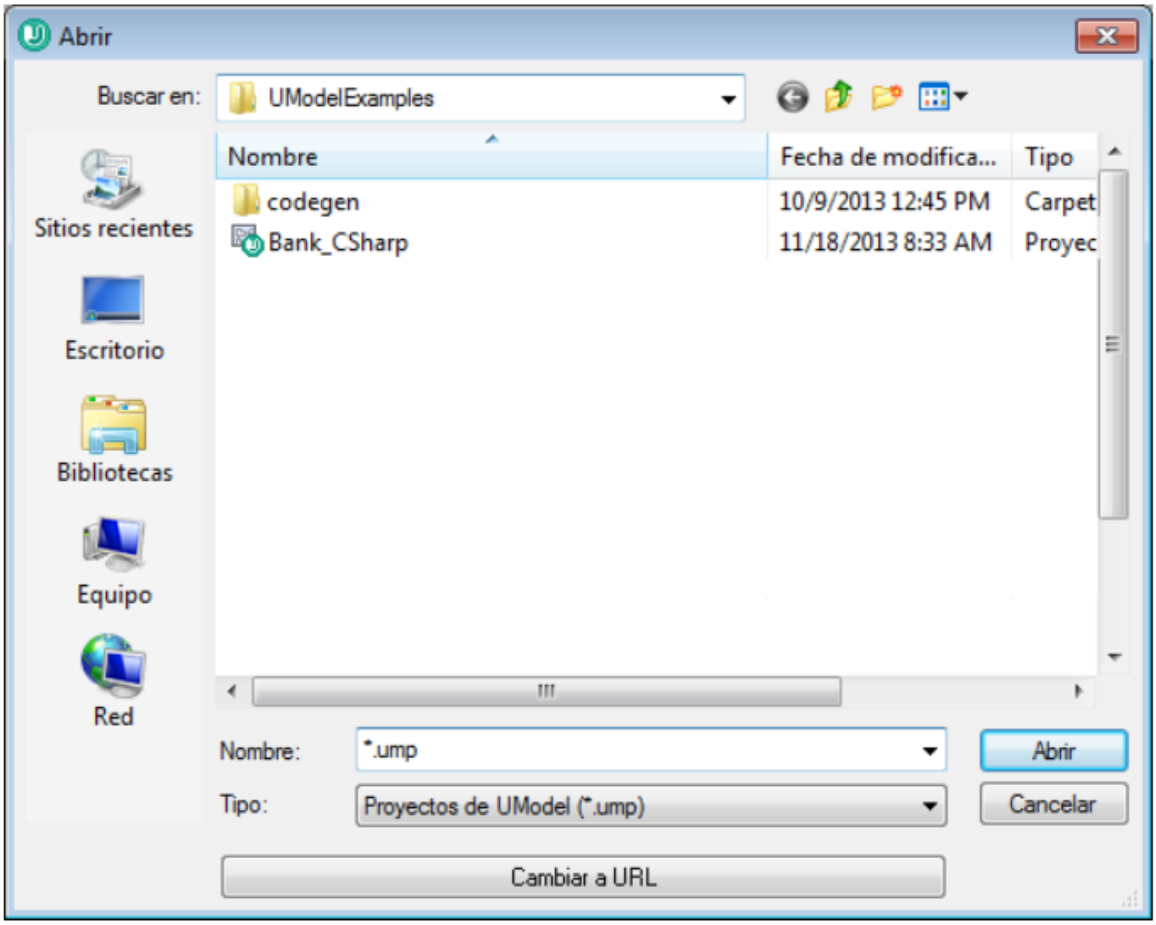
Este comando crea un proyecto local a partir de una BD de control de código fuente ya disponible y lo pone bajo control de código fuente. Para los ejemplos siguientes usamos SourceSafe.

1. Haga clic en **Proyecto | Control de código fuente | Abrir desde el control de código fuente**.
   Se abre el cuadro de diálogo de inicio de sesión. Inserte sus datos para continuar.
   Aparece el cuadro de diálogo "Create local project from SourceSafe".

2. Defina el directorio que debe contener el proyecto local nuevo (p. ej. `c:\temp\ssc`), que en adelante será el directorio de trabajo (o carpeta de desprotección).

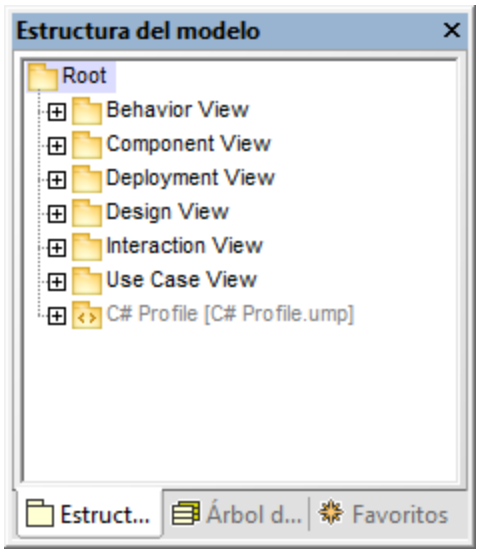


3. Seleccione el proyecto de SourceSafe que desea descargar (p. ej. `Bank_CSharp`).
   Si la carpeta que define aquí no existe en la ubicación, deberá crearla.
4. Haga clic en **Sí** para crear el directorio nuevo. Ahora aparece el cuadro de diálogo "Abrir".



5. Seleccione el archivo de proyecto de UModel `Bank_CSharp.ump` y haga clic en **Abrir**.

   `Bank_CSharp.ump` se abre en UModel y el archivo se pone bajo control de código fuente. Observe que junto a la carpeta **`Root`** (en la Estructura del modelo) aparece un icono en forma de candado. La carpeta **`Root`** representa tanto el archivo de proyecto como el directorio de trabajo para las operaciones de control de código fuente.



El directorio `BankCSharp` se creó localmente y ahora puede trabajar con estos archivos de forma totalmente normal.

Nota: para poner bajo control de código fuente los archivos de código generados cuando se sincronizó el código, consulte el apartado Agregar al control de código fuente.

Iconos del control de código fuente


o

El icono en forma de candado indica que el archivo / la carpeta está bajo control de código fuente pero no está desprotegido.


o

La marca de verificación roja indica que el archivo / la carpeta se desprotegió para poder editarlo. Si además en la barra de título de la aplicación aparece un asterisco, significa que se realizaron cambios en el archivo y al cerrar la aplicación aparece un aviso para que guarde los cambios.


o

El icono en forma de flecha indica que otro usuario de la red desprotegió el archivo o que se desprotegió en otro directorio de trabajo distinto.

14.2.2 Habilitar control de código fuente

Este comando sirve para habilitar/deshabilitar el control de código fuente para proyectos de UModel y está disponible en el menú **Proyecto | Control de código fuente**. Si se ejecuta desde un archivo/una carpeta, la acción se lleva a cabo en todo el proyecto de UModel.

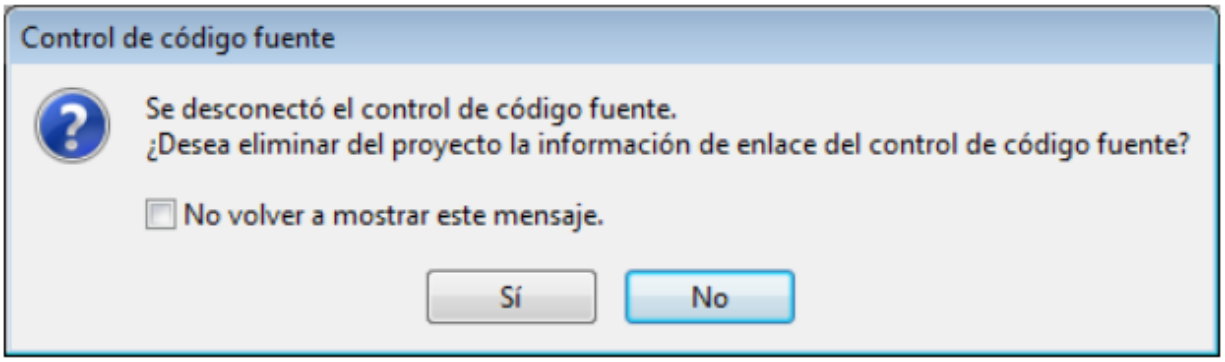
Para habilitar el control de código fuente para un proyecto:

1. Haga clic en **Proyecto | Control de código fuente** y active el comando **Habilitar control de código fuente**.

   La aplicación recupera el estado previo de protección/desprotección de los archivos, lo cual se refleja en la Estructura del modelo.

Para deshabilitar el control de código fuente para un proyecto:

1. Haga clic en el comando **Proyecto | Control de código fuente** y desactive el comando **Habilitar control de código fuente**.



   La aplicación pregunta si quiere quitar la información de enlace del proyecto.

   Haga clic en **No** para deshabilitar el control de código fuente en el proyecto **de forma provisional**.

   Haga clic en **Sí** para deshabilitarlo **permanentemente**.

14.2.3 Obtener la versión más reciente

Este comando **recupera** la versión más reciente del archivo seleccionado del control de código fuente y la **coloca** en el directorio de trabajo. Los archivos se recuperan para solo lectura y no se desprotegen.

Si al ejecutar el comando los archivos están desprotegidos, pueden pasar tres cosas dependiendo del proveedor de control de código fuente: (i) no ocurre nada, (ii) los datos nuevos se combinan con el archivo local (iii) o los cambios se sobrescriben.

Este comando funciona igual que el comando **Obtener**, con la diferencia de que no abre el cuadro de diálogo "Control de código fuente - Obtener". Esto significa, por tanto, que con este comando no se pueden definir opciones avanzadas.

Si se ejecuta en una carpeta, este comando obtiene la versión más reciente recursivamente, es decir, abarca todos los archivos situados bajo el actual.

Para obtener la versión más reciente de un archivo:

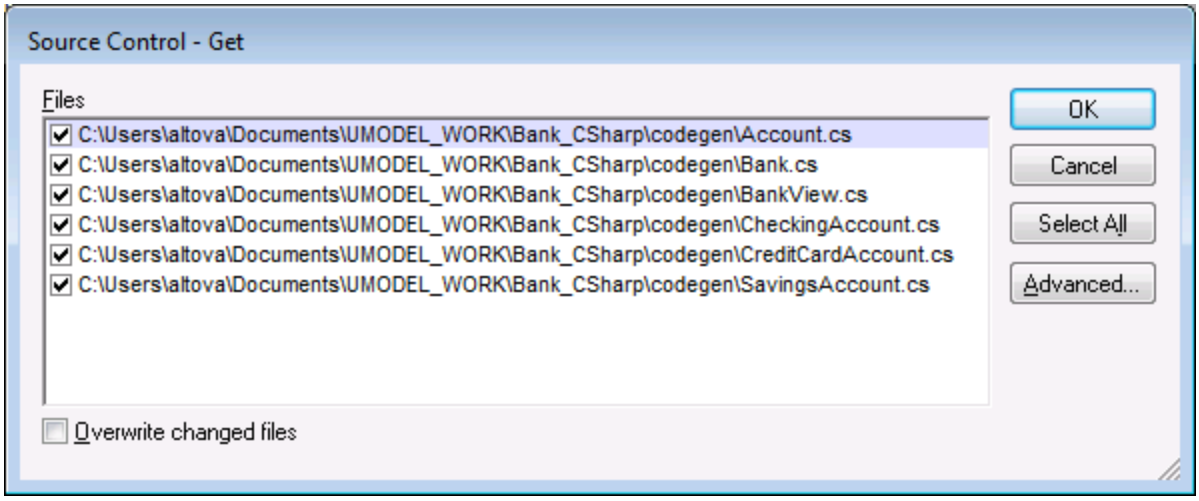
1. En la Estructura del modelo seleccione los archivos cuya versión más reciente desea obtener.
2. Haga clic **Proyecto | Control de código fuente | Obtener la versión más reciente**.

14.2.4 Obtener

Este comando recupera una copia de solo lectura de los archivos seleccionados y los coloca en la carpeta de trabajo. Los archivos no se desprotegen.

Para recuperar una copia de los archivos seleccionados:

1. Seleccione los archivos en la Estructura del modelo.
2. Haga clic en **Proyecto | Control de código fuente | Obtener**.
   Aparece este cuadro de diálogo, cuyas opciones se describen más abajo:

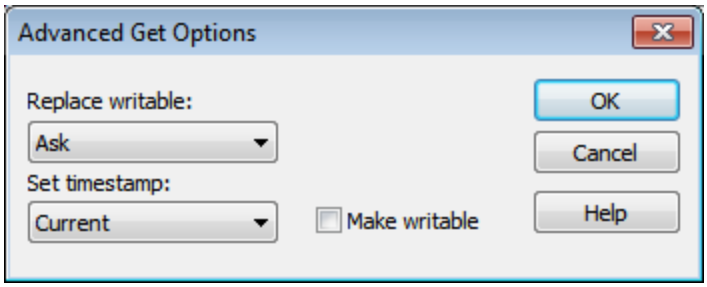


Sobrescribir archivos modificados
Marque esta casilla si quiere sobrescribir los archivos que se modificaron localmente con los archivos de la BD del control de código fuente.

Seleccionar todo
Haga clic en este botón para seleccionar todos los archivos que aparecen en la lista del cuadro de diálogo.

Opciones avanzadas

El cuadro de diálogo "Opciones avanzadas" (*imagen siguiente*) se abre con el botón **Opciones avanzadas** del cuadro de diálogo "Obtener" (*primera imagen de este apartado*).



Aquí puede seleccionar (i) si se reemplazan los archivos que se pueden escribir y que están desprotegidos, (ii) la marca de tiempo y (iii) si la propiedad de solo lectura del archivo recuperado se cambia para que el archivo se pueda escribir.

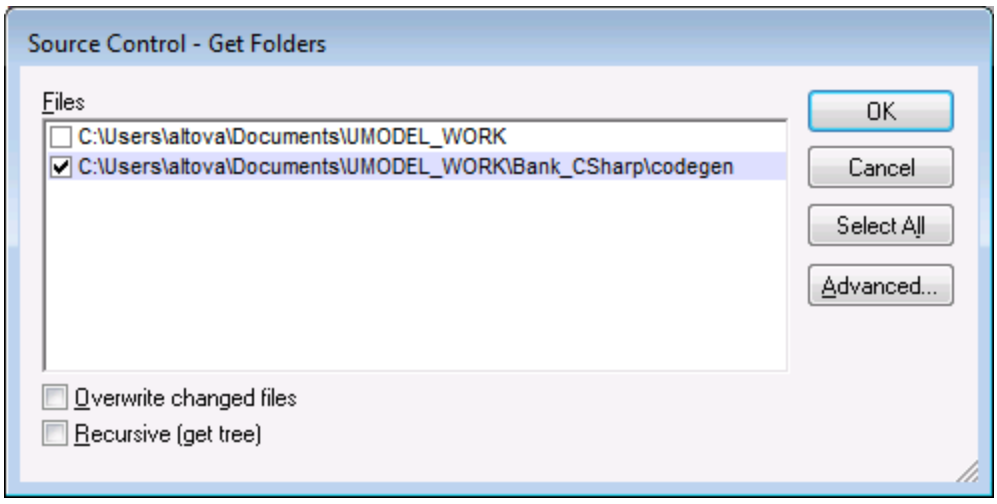
La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

14.2.5 Obtener carpetas

Este comando recupera una copia de solo lectura de los archivos de las carpetas seleccionadas y las coloca en la carpeta de trabajo. Los archivos no se desprotegen.

Para obtener una copia de los archivos de las carpetas seleccionadas:

1. En la Estructura del modelo seleccione qué carpetas desea obtener.
2. Haga clic en **Proyecto | Control de código fuente | Obtener**.
   Aparece este cuadro de diálogo, cuyas opciones se describen más abajo:



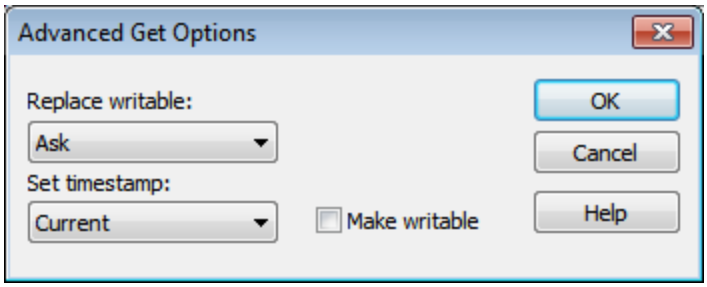
Sobrescribir archivos modificados
Marque esta casilla para sobrescribir los archivos que se modificaron localmente con los archivos de la BD del control de código fuente.

Jerarquía recursiva (obtener árbol)

Marque esta casilla para recuperar todos los archivos de la estructura de carpetas situada bajo la carpeta seleccionada.

Opciones avanzadas

El cuadro de diálogo "Opciones avanzadas" (*imagen siguiente*) se abre con el botón **Opciones avanzadas** del cuadro de diálogo "Obtener" (*primera imagen de este apartado*).



Aquí puede seleccionar (i) si se reemplazan los archivos que se pueden escribir y que están desprotegidos, (ii) la marca de tiempo y (iii) si la propiedad de solo lectura del archivo recuperado se cambia para que el archivo se pueda escribir.

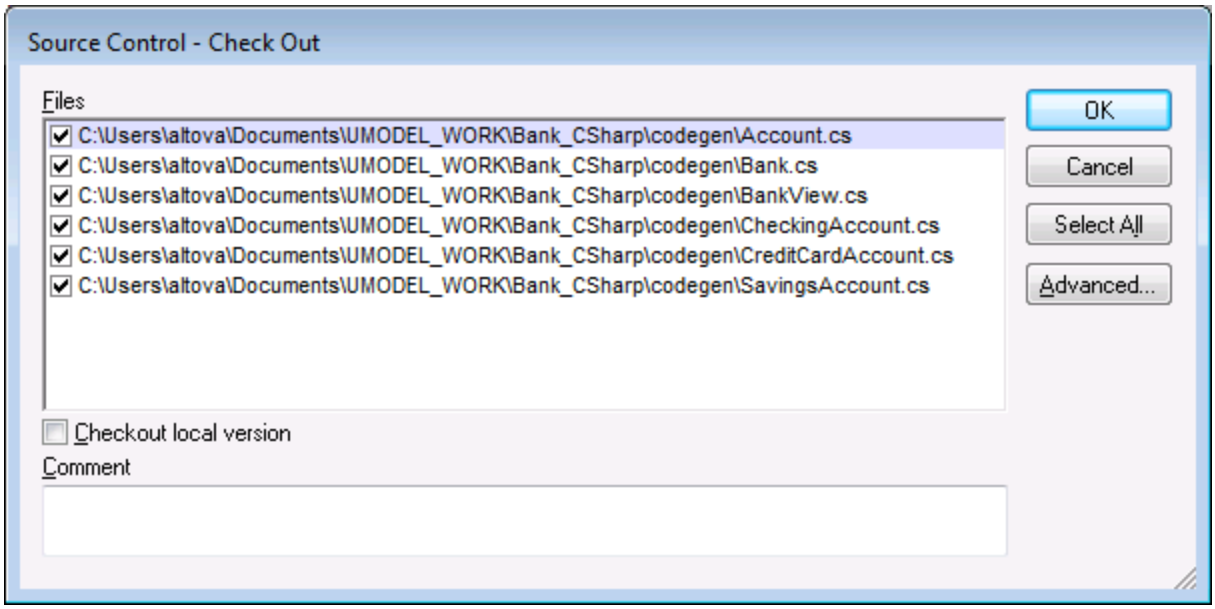
La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

14.2.6 Desproteger

Este comando desprotege la versión más reciente de los archivos seleccionados y coloca una copia editable en el directorio de trabajo. Los otros usuarios ven un icono de "desprotegido" en los archivos desprotegidos.

Para desproteger archivos:

1. En la Estructura del modelo seleccione el archivo o la carpeta que desea desproteger.
2. Haga clic en **Proyecto | Control de código fuente | Desproteger**.

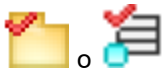


Nota: puede cambiar el número de archivos que quiere desproteger; para ello marque las casillas individuales en la caja de texto Archivos.

Marque esta casilla para desproteger solamente las versiones locales de los archivos y no las versiones que están en la BD del control de código fuente.

Estos son los elementos que se pueden desproteger:

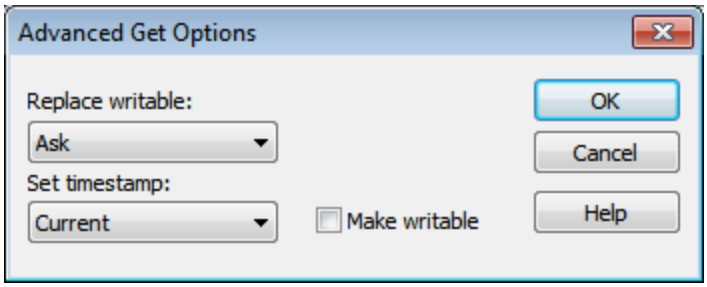
- Archivos (para seleccionar varios, pulse la tecla **Ctrl** mientras hace clic en los archivos en la Estructura del modelo).
- Carpetas (para seleccionar varias, pulse la tecla **Ctrl** mientras hace clic en las carpetas en la Estructura del modelo).

 o

La marca de verificación **roja** indica que el archivo / la carpeta **se desprotegió** para poder editarlo.

Opciones avanzadas

El cuadro de diálogo "Opciones avanzadas" (*imagen siguiente*) se abre con el botón **Opciones avanzadas** del cuadro de diálogo "Obtener" (*primera imagen de este apartado*).



Aquí puede seleccionar (i) si se reemplazan los archivos que se pueden escribir y que están desprotegidos, (ii) la marca de tiempo y (iii) si la propiedad de solo lectura del archivo recuperado se cambia para que el archivo se pueda escribir.

La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

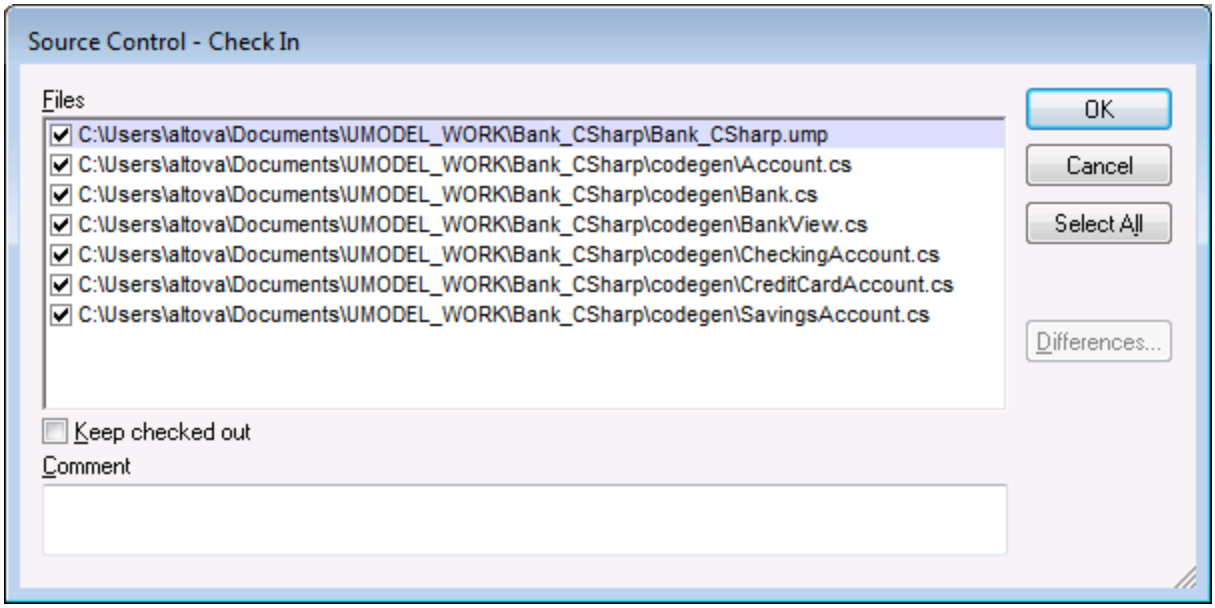
14.2.7 Proteger

Este comando protege los archivos que estén desprotegidos (es decir, los archivos que se modificaron localmente) y los pone en la BD del control de código fuente.

Para proteger archivos:

1. En la Estructura del modelo seleccione los archivos que quiere proteger.
2. Haga clic en **Proyecto | Control de código fuente | Proteger**.

   Nota: también puede hacer clic con el botón derecho en los archivos que quiere proteger y seleccionar **Proteger** en el menú contextual.



Estos son los elementos que se pueden proteger:

- Archivos (para seleccionar varios, pulse la tecla **Ctrl** mientras hace clic en los archivos en la Estructura del modelo).
- Carpetas (para seleccionar varias, pulse la tecla **Ctrl** mientras hace clic en las carpetas en la Estructura del modelo).

 o

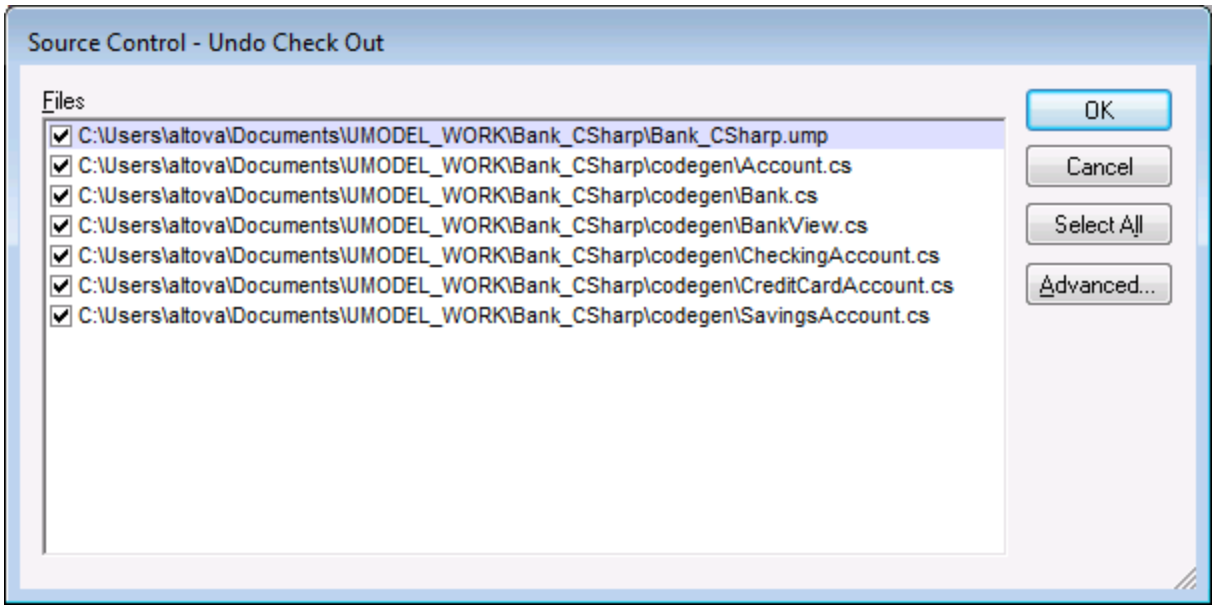
El icono en forma de candado indica que el archivo / la carpeta está **bajo control de código fuente** pero no está desprotegido.

14.2.8 Anular desprotección

Este comando descarta los cambios realizados en archivos desprotegidos (es decir, los archivos que se modificaron localmente) y mantiene la versión previa de la BD del control de código fuente.

Para anular la desprotección:

1. Seleccione los archivos correspondientes en la Estructura del modelo.
2. Haga clic en **Proyecto | Control de código fuente | Anular desprotección**.
   Aparece un cuadro de diálogo (*imagen siguiente*) donde puede seleccionar para qué archivos se anula la desprotección finalmente (marcando sus casillas).

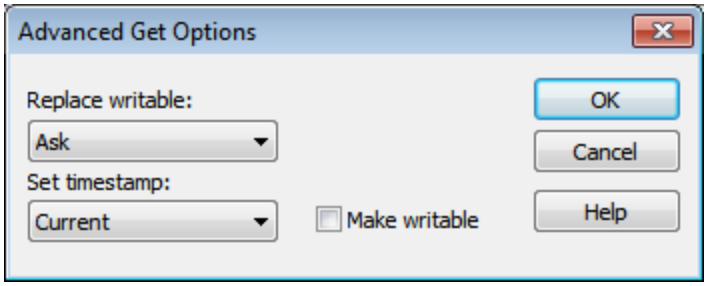


Estos son los elementos cuya desprotección se puede anular:

- Archivos (para seleccionar varios, pulse la tecla **Ctrl** mientras hace clic en los archivos en la Estructura del modelo).
- Carpetas (para seleccionar varias, pulse la tecla **Ctrl** mientras hace clic en las carpetas en la Estructura del modelo).

Opciones avanzadas

El cuadro de diálogo "Opciones avanzadas" (*imagen siguiente*) se abre con el botón **Opciones avanzadas** del cuadro de diálogo "Obtener" (*primera imagen de este apartado*).



Aquí puede seleccionar (i) si se reemplazan los archivos que se pueden escribir y que están desprotegidos, (ii) la marca de tiempo y (iii) si la propiedad de solo lectura del archivo recuperado se cambia para que el archivo se pueda escribir.

La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

14.2.9 Agregar al control de código fuente

Este comando añade los archivos o las carpetas seleccionados a la BD del control de código fuente y los pone bajo su control. Si añade un proyecto de UModel nuevo, deberá indicar la carpeta del espacio de trabajo y la ubicación donde se debe almacenar el proyecto.

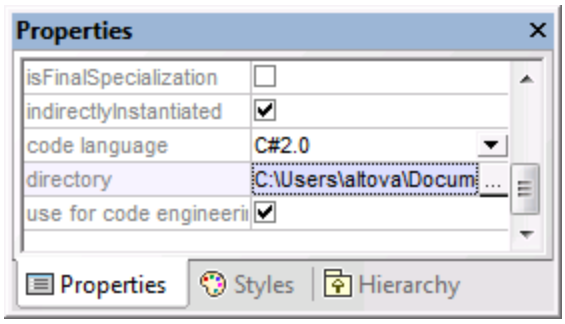
Tras poner el proyecto de UModel (`*.ump`) bajo control de código fuente, puede añadir al control de código fuente los **archivos de código** (creados durante el proceso de generación de código).

Es importante tener en cuenta que para que esto funcione los archivos de código generados y el proyecto de UModel deben ponerse dentro/bajo el mismo **directorio de trabajo** de SourceSafe. El directorio de trabajo que utilizamos para este ejemplo es `C:\Users\Altova\Documents\UMODEL_WORK\`.

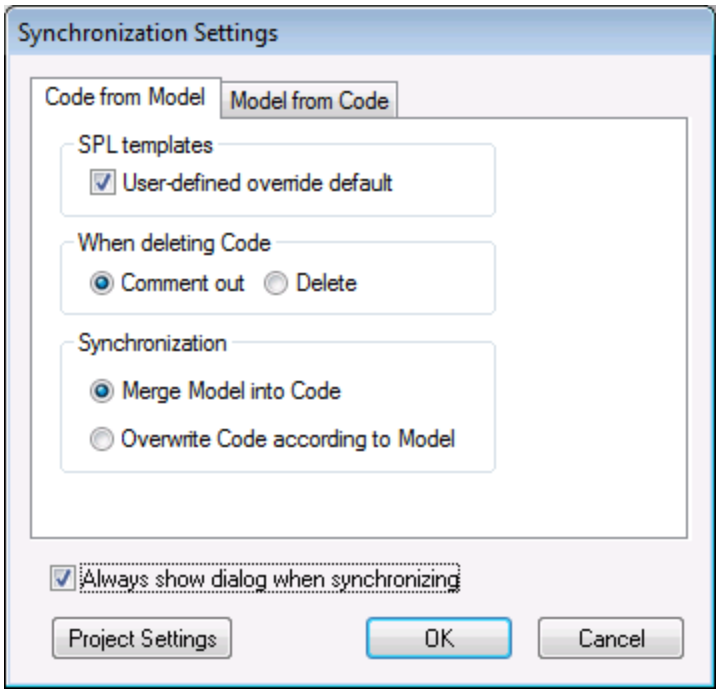
Para agregar archivos de código generados con UModel al control de código fuente:

1. En la Estructura del modelo expanda la carpeta **`Component View`** y navegue hasta el componente **`BankView`**.

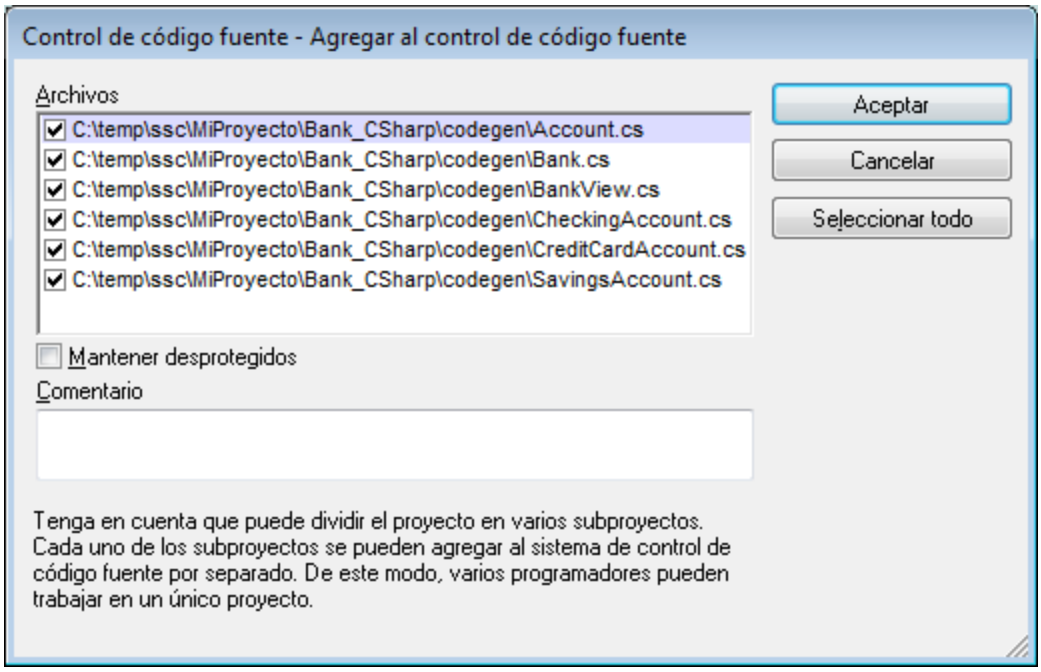
2. Haga clic en el componente **`BankView`**. En la ventana Propiedades haga clic en el icono **Examinar** del campo `directorio`.



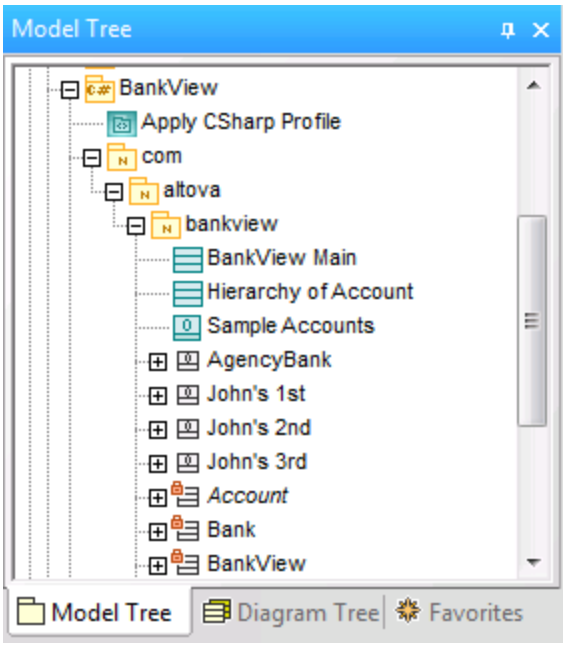
3. Cambie el directorio de ingeniería de código a `C:\Users\Altova\Documents\UMODEL_WORK\codegen`.
4. Ahora haga clic en **Proyecto | Combinar el código de programa con el proyecto de UModel**.
5. Si es necesario, cambie las opciones de configuración y haga clic en **Aceptar**.
   La ventana Mensajes muestra cómo se desarrolla el proceso.
   Aparece un cuadro de mensajes que pregunta si desea poner los archivos recién creados bajo control de código fuente.



6. Haga clic en **Sí**.
7. Aparece el cuadro de diálogo "Agregar al control de código fuente" donde puede seleccionar qué archivos se ponen bajo control de código fuente.



8. Seleccione los archivos y haga clic en **Aceptar**.
   Observe que junto a las clases que ahora están bajo control de código fuente aparece un icono en forma de candado.

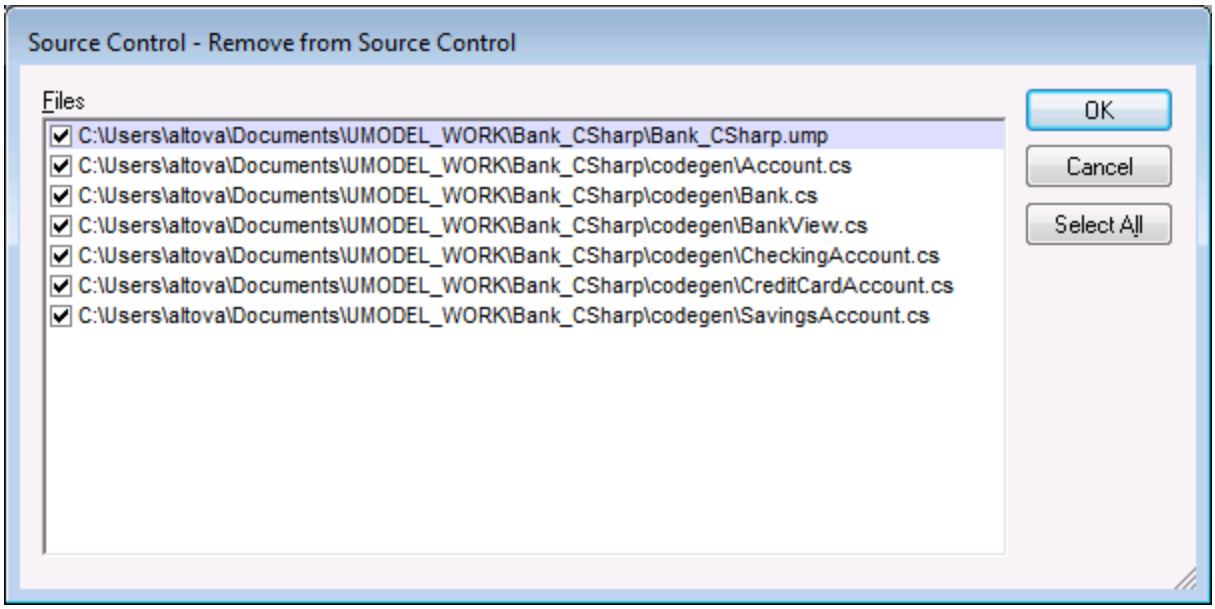


14.2.10 Quitar del control de código fuente

Este comando quita de la BD del control de código fuente los archivos añadidos previamente a la BD. Este tipo de archivos siguen estando visibles en la Estructura del modelo pero no se pueden proteger ni desproteger. Para ponerlos otra vez bajo control de código fuente, utilice el comando **Agregar al control de código fuente**.

Para quitar archivos del control de código fuente:

1. Seleccione los archivos en la Estructura del modelo.
2. Haga clic en **Proyecto | Control de código fuente | Quitar del control de código fuente**. Aparece un cuadro de diálogo (*imagen siguiente*) donde puede seleccionar qué archivos se quitan del control de código fuente finalmente (marcando sus casillas).



Estos son los elementos que se pueden quitar del control de código fuente:

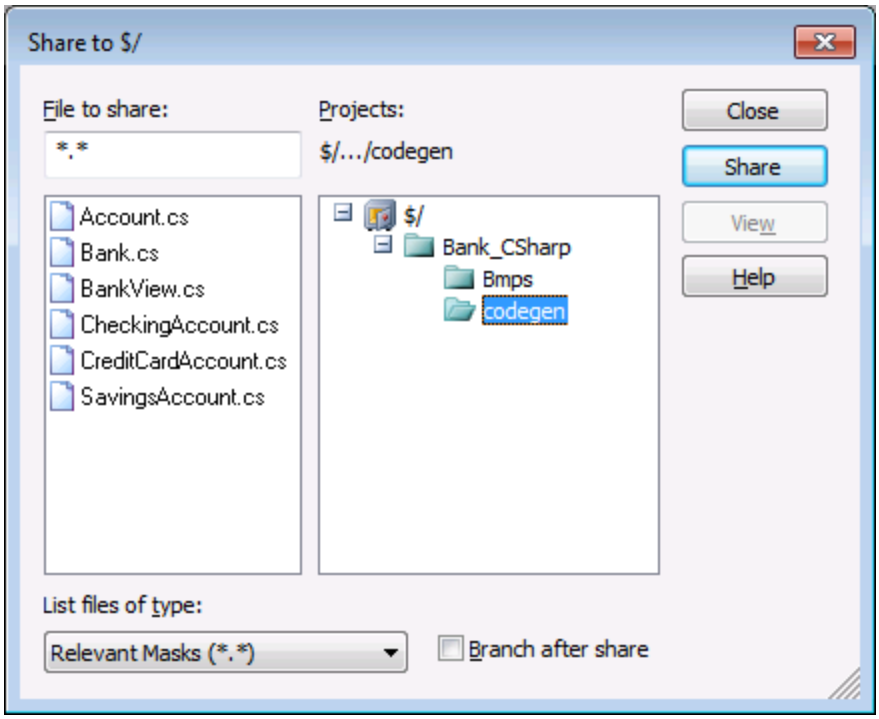
- Archivos (para seleccionar varios, pulse la tecla **Ctrl** mientras hace clic en los archivos en la Estructura del modelo).
- Carpetas (para seleccionar varias, pulse la tecla **Ctrl** mientras hace clic en las carpetas en la Estructura del modelo).

14.2.11 Compartir desde el control de código fuente

Este comando comparte/ramifica archivos de otros proyectos/carpetas del repositorio de control de código fuente con la carpeta seleccionada. Para usar este comando es necesario tener privilegios para proteger/desproteger datos en el proyecto desde el que se comparten los archivos.

Para compartir un archivo desde el control de código fuente:

1. En la Estructura del modelo seleccione la carpeta con la que quiere compartir archivos (p. ej. `BankView Component` de la carpeta `Component View`).
2. Haga clic en **Proyecto | Control de código fuente | Compartir desde el control de código fuente**.
   Aparece un cuadro de diálogo (*imagen siguiente*) donde puede seleccionar qué carpeta de proyecto contiene el archivo que desea compartir.



3. Seleccione el archivo que desea compartir y haga clic en el botón **Share**.
   El archivo se elimina de la lista *File to share*.
4. Haga clic en el botón **Close** para continuar.

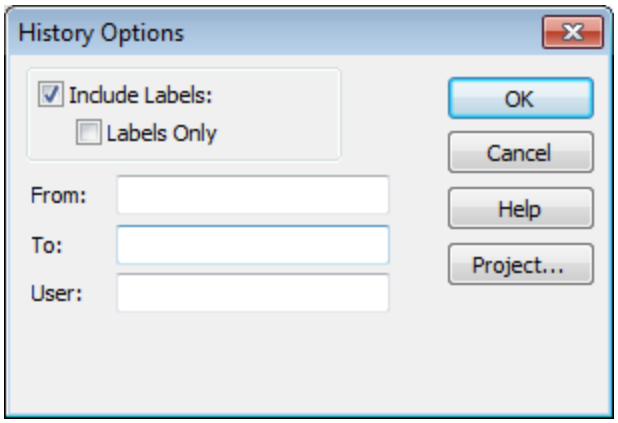
   Nota: marque la casilla *Branch after share* para compartir el archivo y crear una rama nueva para crear una nueva versión.

14.2.12 Mostrar historial

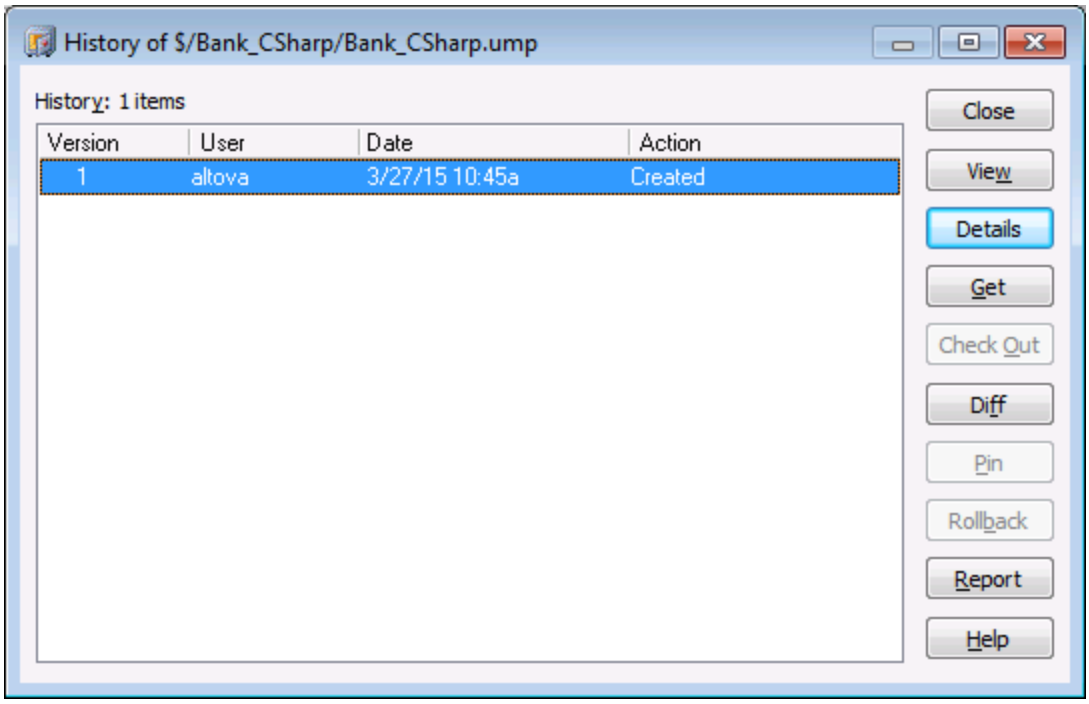
Este comando **muestra el historial** de un archivo que está bajo control de código fuente. El historial permite ver la historia de cambios por la que ha pasado el archivo, ver diferencias entre sus diferentes versiones y recuperar versiones previas.

Para ver el historial de un archivo:

1. Seleccione el archivo en la Estructura del modelo.
2. Haga clic en el comando **Proyecto | Control de código fuente | Mostrar historial**.
   Aparece un cuadro de diálogo pidiendo más información.



3. Seleccione las entradas correspondientes y haga clic en **OK** para confirmar.



En este cuadro de diálogo puede comparar versiones del archivo y obtener versiones previas.

Para ver el historial detallado haga doble clic en una entrada de la lista.

Estos son los botones del cuadro de diálogo del historial:

Close
Cierra el cuadro de diálogo.

View
Abre otro cuadro de diálogo donde puede seleccionar en qué aplicación desea ver el archivo.

Details
Abre un cuadro de diálogo que muestra las propiedades del archivo seleccionado.

Get
Recupera una de las versiones previas del archivo y la coloca en el directorio de trabajo.

Check Out
Desprotege la versión **más reciente** del archivo.

Diff
Abre el cuadro de diálogo "Difference options" donde puede configurar la vista de las diferencias detectadas entre las dos versiones del archivo.
Para marcar dos versiones de un archivo pulse la tecla **Ctrl** mientras hace clic en las entradas. Después pulse el botón **Diff** para ver las diferencias.

Pin
Ancla/desancla una versión del archivo para que pueda definir la versión que se debe usar en la comparación de dos archivos.

Rollback
Revierte a la versión seleccionada del archivo.

Report
Genera un historial que puede imprimirse, guardarse en un archivo o copiarse en el portapapeles.

Help
Abre la ayuda en pantalla del cliente de control de código fuente.

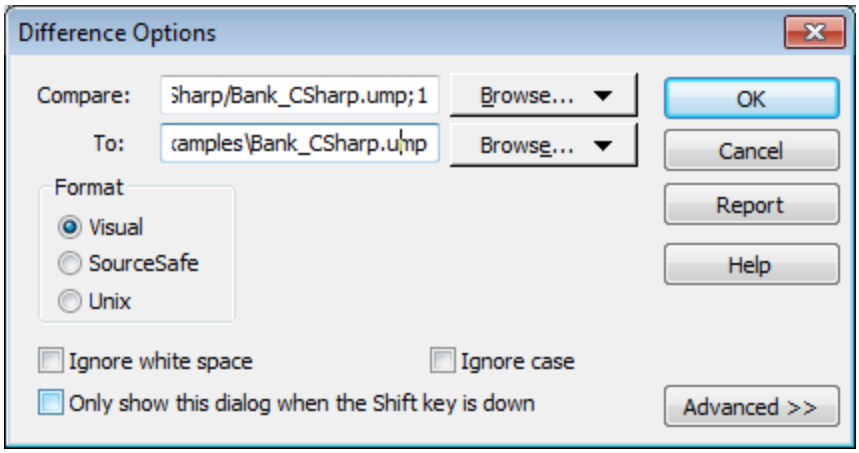
14.2.13 Mostrar diferencias

Este comando muestra las diferencias que existen entre el archivo que está en el repositorio del control de código fuente y el mismo archivo protegido/desprotegido del directorio de trabajo.

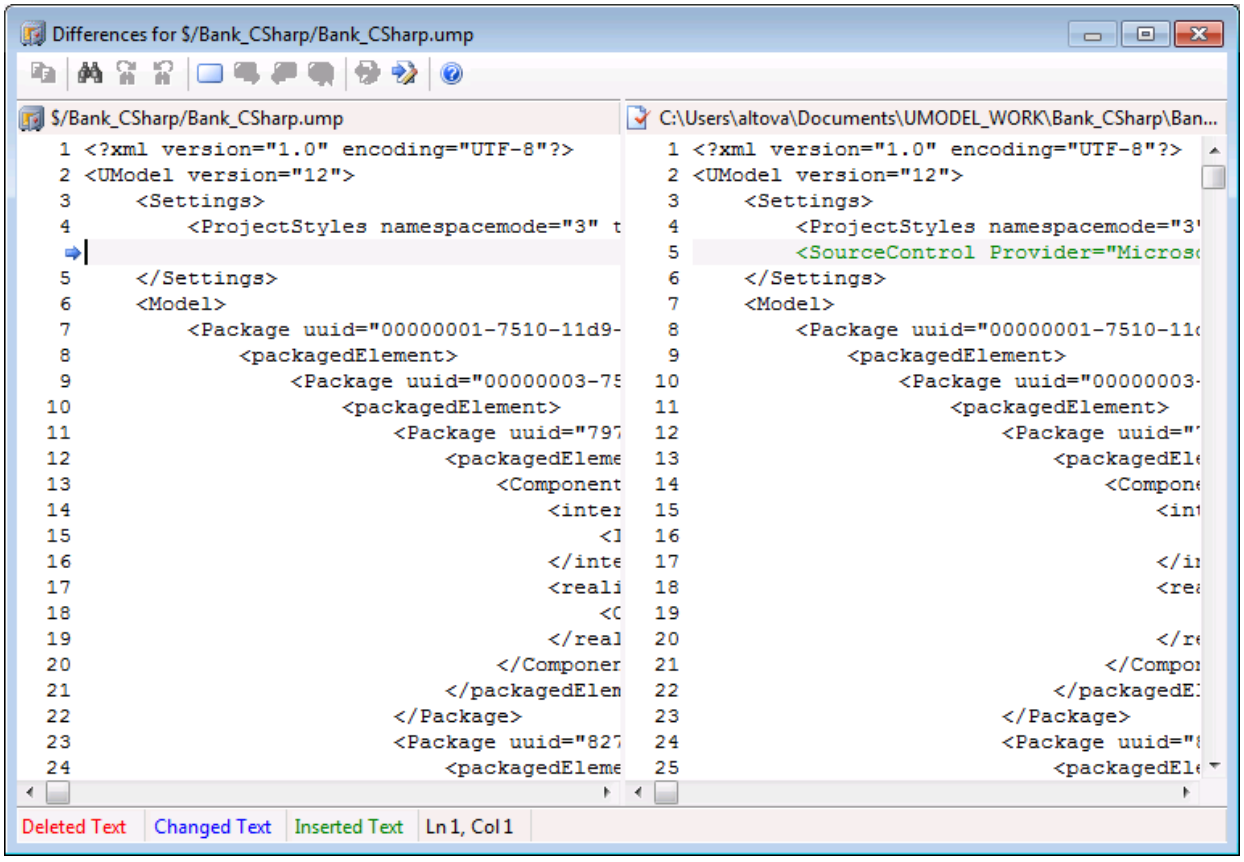
Si ancló uno de los archivos en el cuadro de diálogo del historial, el archivo anclado se inserta automáticamente en el campo de texto *Compare*. Con los botones **Browse** puede buscar los archivos que desea comparar.

Para ver las diferencias que hay entre dos archivos:

1. En la Estructura del modelo seleccione el archivo que desea comparar.
2. Haga clic en **Proyecto | Control de código fuente | Mostrar diferencias**.
   Aparece un cuadro de diálogo que le pide más información.



3. Seleccione las entradas correspondientes y haga clic en OK para **confirmar**.



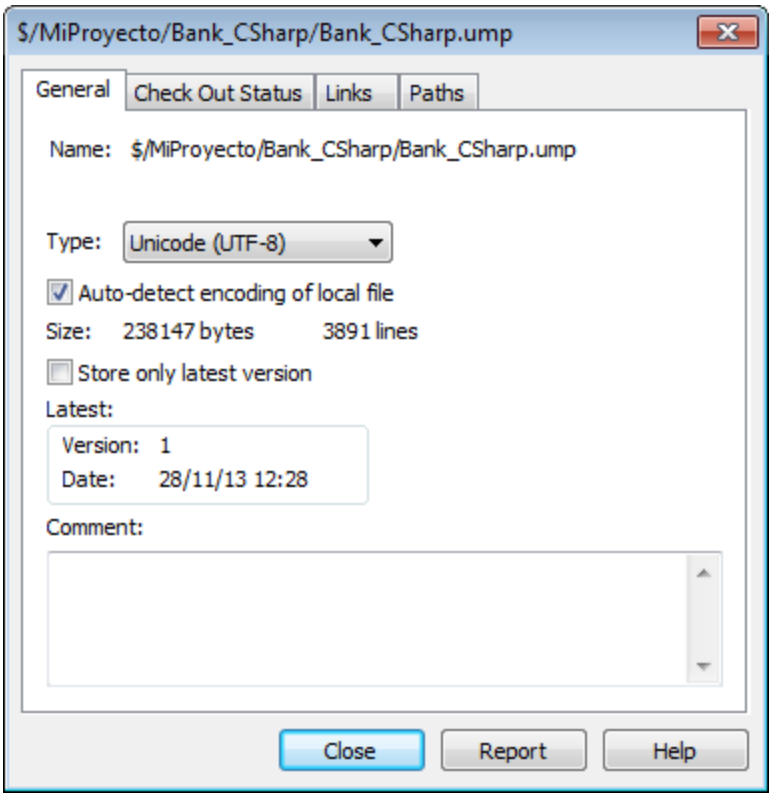
Las diferencias detectadas aparecen resaltadas. Por ejemplo, la imagen anterior muestra los resultados de la comparación en MS SourceSafe.

14.2.14 Mostrar propiedades

Este comando muestra las propiedades del archivo seleccionado y varía de un proveedor de control de código fuente a otro.

Para ver las propiedades del archivo seleccionado haga clic en **Proyecto | Control de código fuente | Mostrar propiedades**.

Tenga en cuenta que este comando no se puede ejecutar en varios archivos a la vez.



14.2.15 Actualizar estado

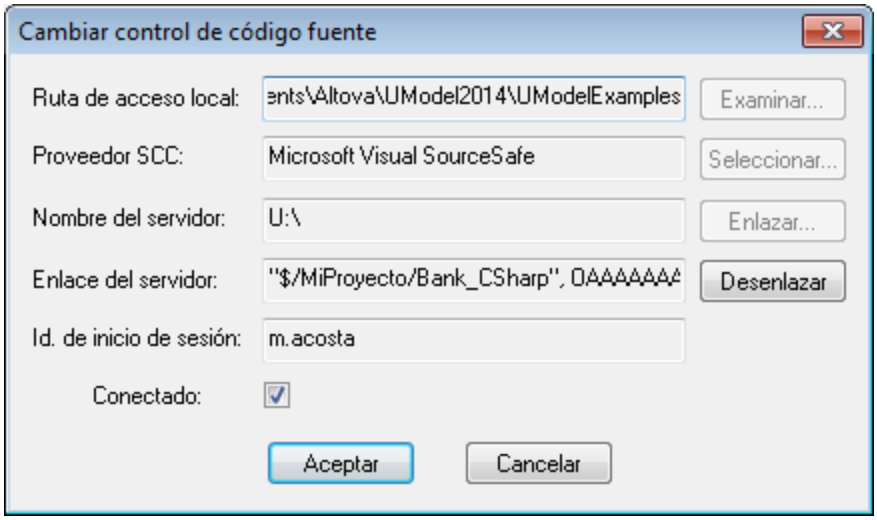
Este comando **actualiza** el estado de todos los archivos de proyecto, independientemente de cuál sea su estado actual.

14.2.16 Administrador del control de código fuente

Este comando **inicia** el software de control de código fuente, con su interfaz de usuario nativa.

14.2.17 Cambiar control de código fuente

Este cuadro de diálogo sirve para cambiar el enlace de control de código fuente activo. Haga clic en el botón **Desenlazar** y después (si quiere) haga clic en el botón **Seleccionar** para seleccionar un proveedor nuevo. Para terminar haga clic en el botón **Enlazar** para enlazar el proyecto con una ubicación nueva del repositorio.

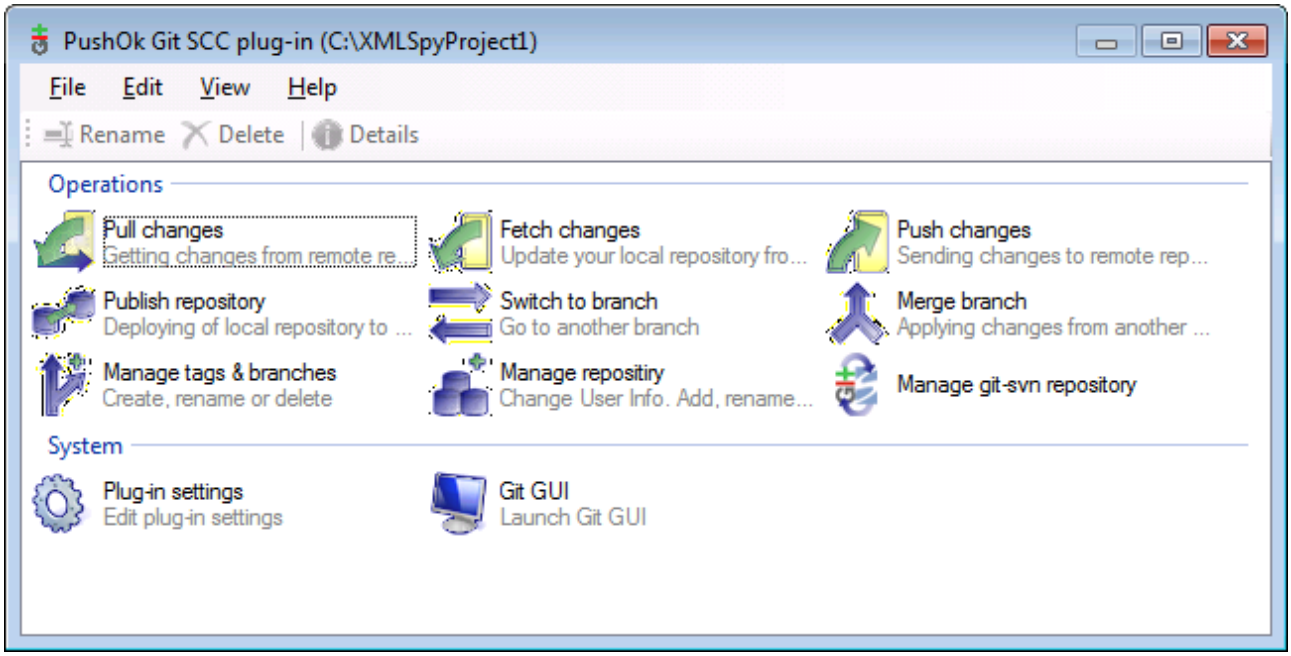


14.3 Control de código fuente con Git

UModel es compatible con el sistema de control de versiones Git por medio de un complemento externo llamado **GIT SCC plug-in** (http://www.pushok.com/software/git.html).

> Cuando se redactó esta documentación, la versión del complemento **GIT SCC plug-in** era una versión experimental. Para usar el complemento es necesario registrarse con el autor del complemento.

El complemento GIT SCC permite trabajar con repositorios Git utilizando los comandos del menú **Proyecto | Control de código fuente** de UModel. Recuerde que los comandos de este menú vienen de la API del complemento Microsoft Source Control, cuyo diseño es diferente al de Git. Como consecuencia, el complemento hace de intermediario entre las funciones tipo Visual Source Safe y las funciones de Git. Esto significa, por un lado, que algunos comandos como **Obtener la versión más reciente** no estarán habilitados cuando trabaje con Git. Por otro lado, hay acciones nuevas propias de Git que están disponibles en el cuadro de diálogo de administración del código fuente (**Proyecto | Control de código fuente | Administrador del control de código fuente** en UModel).



En el menú **Proyecto | Control de código fuente** también encontrará los comandos más frecuentes de Git.

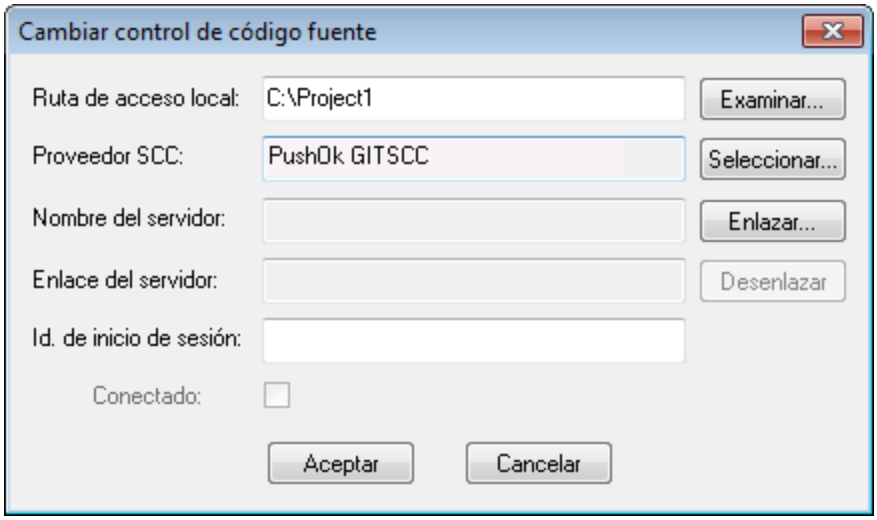
Los diferentes apartados de esta sección describen la configuración inicial del complemento y el flujo de trabajo básico:

- Habilitar Git con el complemento GIT SCC
- Agregar un proyecto al control de código fuente de Git
- Clonar un proyecto desde el control de código fuente de Git

14.3.1 Habilitar Git con el complemento de control de código fuente

Para habilitar el control de código fuente de Git en UModel es necesario tener instalado el complemento externo **PushOK GIT SCC plug-in**, registrarse y seleccionarlo en la lista de proveedores de control de código fuente:

1. Descargue el archivo de instalación del complemento desde el sitio web del autor (http://www.pushok.com), ejecútelo y siga las instrucciones que aparecen en pantalla.
2. En el menú **Proyecto** de UModel, haga clic en **Proyecto | Control de código fuente | Cambiar de control de código fuente** y seleccione **PushOk GITSCC**. Si **Push Ok GITSCC** no aparece en la lista de proveedores, es probable que la instalación del complemento no finalizara correctamente. Consulte la documentación del autor para resolver este problema.



3. Para terminar debe registrar el complemento haciendo clic en **Registration**. Siga los pasos del asistente para terminar de registrar el complemento.

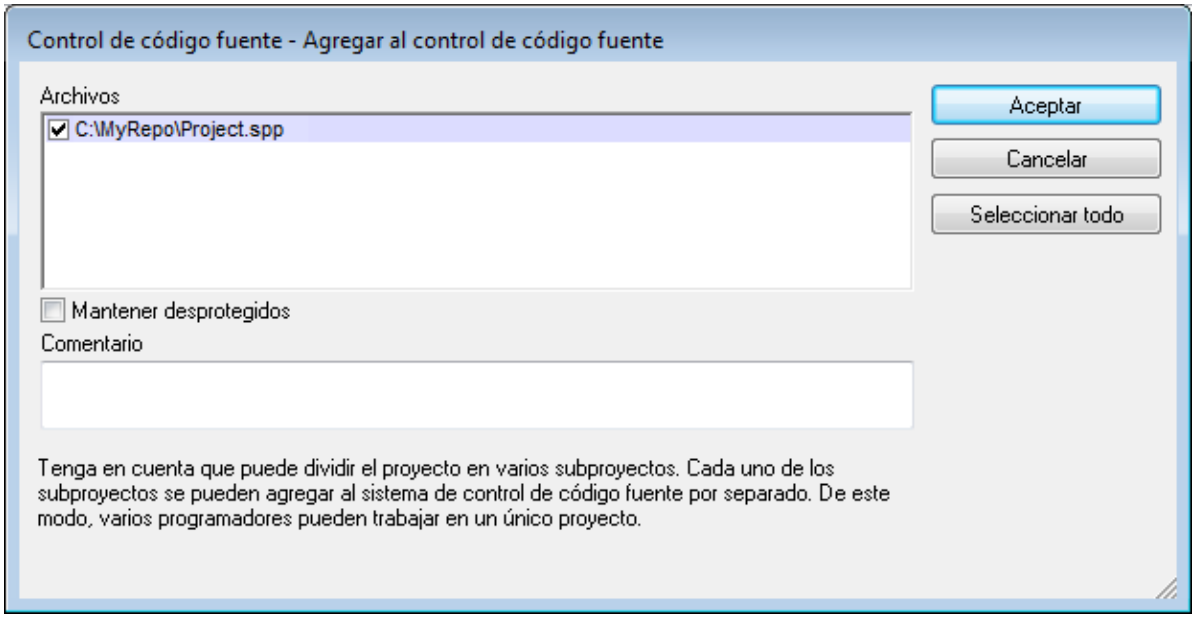
14.3.2 Agregar un proyecto al control de código fuente de Git

Puede guardar proyectos de UModel como repositorios de Git. La estructura de los archivos o carpetas que añada al proyecto se corresponderán con la estructura del repositorio Git.

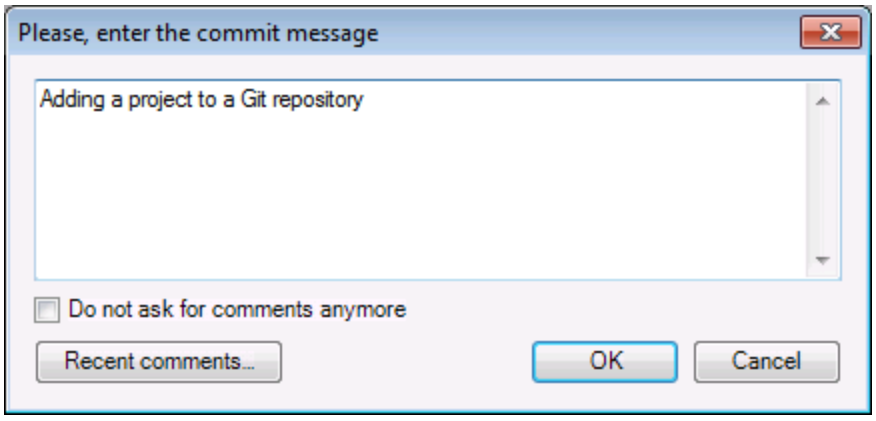
Para agregar un proyecto al control de código fuente de Git:

1. Compruebe que el proveedor de control de código fuente seleccionado es **PushOK GIT SCC Plug-in** (*ver el apartado anterior*).
2. Cree un proyecto nuevo vacío y compruebe que no hay errores de validación (es decir, que no se detectan errores ni advertencias tras ejecutar el comando **Proyecto | Revisar la sintaxis del proyecto**).
3. Guarde el proyecto en una carpeta local (p. ej. `C:\MyRepo\Project.ump`).
4. En el panel Estructura del modelo haga clic en el nodo `Root`.

5. Ahora haga clic en **Proyecto | Control de código fuente | Agregar al control de código fuente**.



6. Haga clic en **Aceptar**.



7. Escriba el texto del mensaje de confirmación y haga clic en **OK** para agregar el proyecto al control de código fuente.

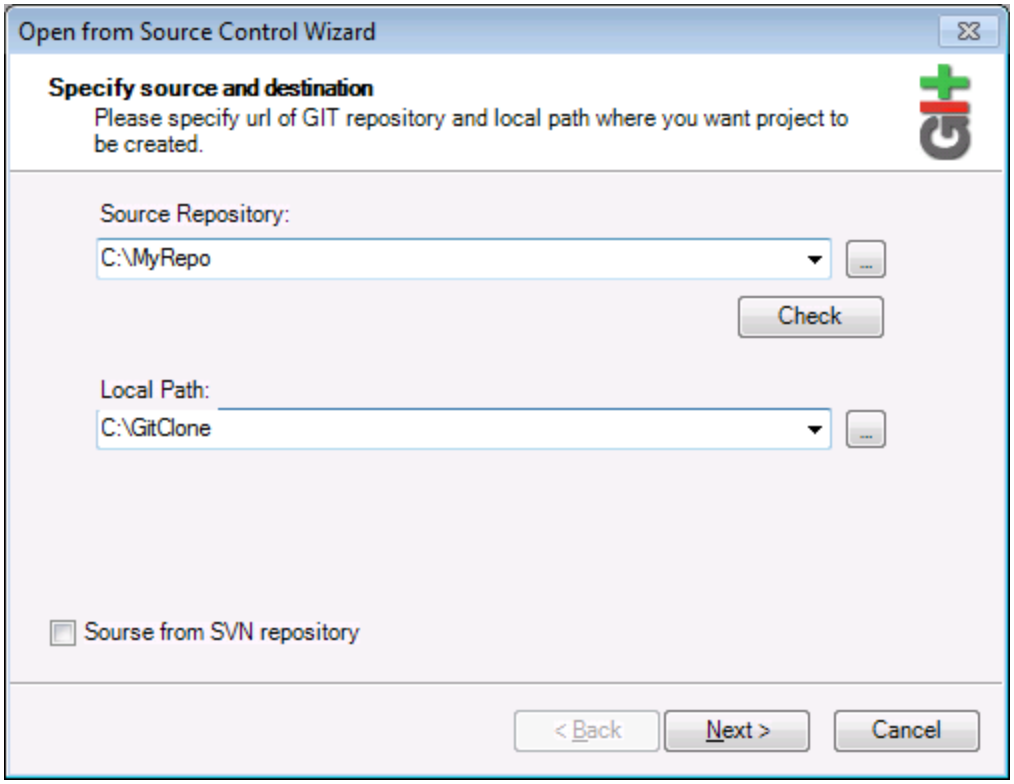
Ahora ya puede añadir elementos de modelado (diagramas, clases, paquetes, etc.) al proyecto. Recuerde que todos los archivos y carpetas del proyecto deben estar bajo la carpeta raíz del proyecto. Por ejemplo, si creó el proyecto en la carpetea `C:\MyRepo`, entonces solamente podrá añadir al proyecto los archivos que estén bajo `C:\MyRepo`. Si intenta añadir archivos de proyecto que estén fuera de la carpeta raíz del proyecto, aparecerá este mensaje de advertencia:

Sólo se pueden agregar archivos a una ubicación bajo la raíz de enlace del proyecto (C:\MyRepo).

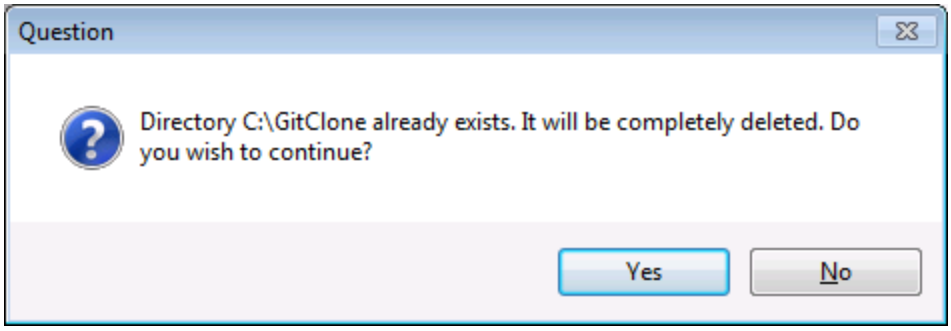
14.3.3 Clonar un proyecto desde el control de código fuente de Git

Los proyectos que ya estén en el control de código fuente de Git (*ver el apartado anterior*) se pueden abrir desde el repositorio Git:

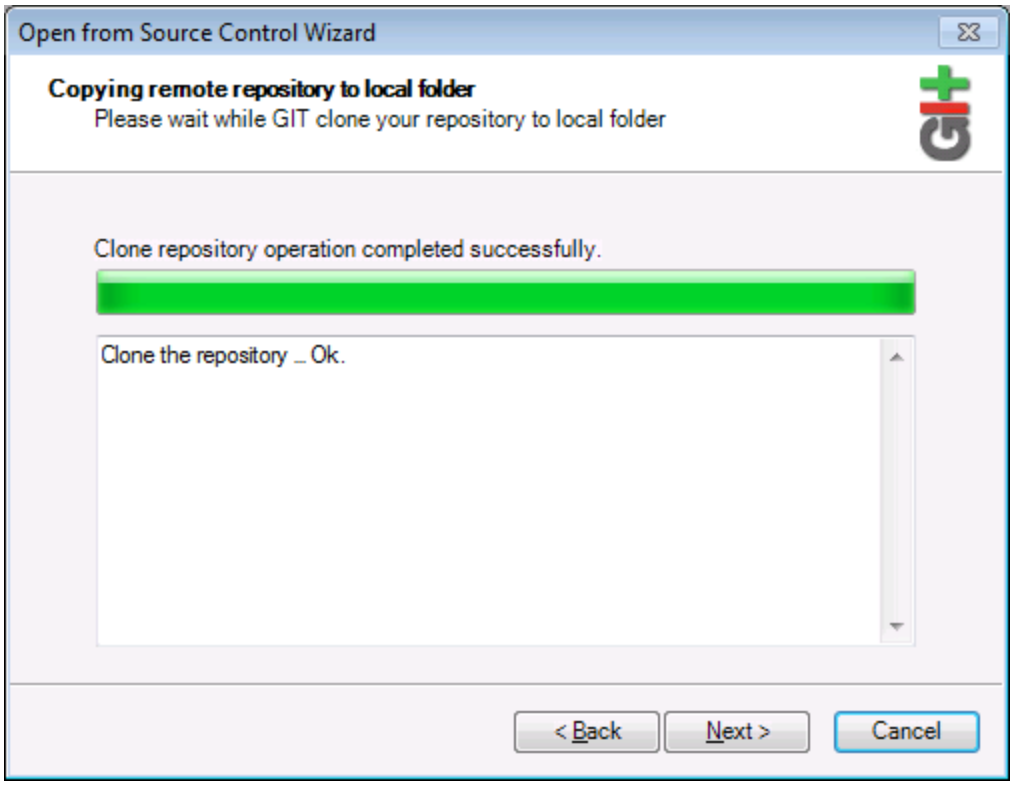
1. Compruebe que el proveedor de control de código fuente seleccionado es **PushOK GIT SCC Plug-in** (*ver el apartado habilitar Git con complemento de control de código fuente GIT SCC*).
2. Haga clic en **Proyecto | Control de código fuente | Abrir desde el control de código fuente**.
3. Escriba la ruta de acceso o la URL del repositorio fuente. Haga clic en el botón **Check** para verificar la ruta de acceso o la dirección URL.



4. En el campo *Local Path* escriba la ruta de acceso de la carpeta local donde desea crear el proyecto y haga clic en **Next** para continuar. Si la carpeta local ya existe (aunque esté vacía), aparece este cuadro de diálogo preguntando si desea borrar totalmente la carpeta:



5. Haga clic en **Yes** para confirmar y después en **Next** para continuar.



6. Siga los pasos del asistente hasta el final.
7. Al final aparece un cuadro de diálogo "Explorar" donde puede abrir el proyecto de UModel (archivo *.ump). Seleccione el archivo de proyecto para cargar el contenido del proyecto en UModel.

15 Iconos en los diagramas de UModel

En UModel cada tipo de diagrama tiene una barra de herramientas distinta con iconos para los elementos compatibles con el tipo de diagrama correspondiente.

Estos iconos pueden ser de dos tipos:

- **Agregar:** en este grupo están los iconos de todos los elementos que se pueden agregar en el diagrama.
- **Relación:** en este grupo están todos los iconos de los tipos de relación que se pueden crear entre los elementos del diagrama.

15.1 Diagramas de actividades

Agregar

Acción (AcciónLlamadaDeComportamiento)
Acción (AcciónOperaciónDeLlamada)
AcciónAceptarEvento
AcciónAceptarEvento (EventoDeTiempo)
AcciónEnviarSeñal

NodoDeDecisión (rama)
NodoDeCombinación
NodoInicial
NodoFinalDeActividad
NodoFinalDeFlujo
NodoDeBifurcación (vertical)
NodoDeBifurcación (horizontal)
NodoDeReunión
NodoDeReunión (horizontal)

PinDeEntrada
PinDeSalida
PinDeValor

NodoDeObjeto
NodoDeBúferCentral
NodoAlmacénDeDatos
ParticiónDeActividades (horizontal)
ParticiónDeActividades (vertical)
ParticiónDeActividades (2D)

FlujoDeControl
FlujoDeObjeto
ControladorDeExcepción

Actividad
NodoParámetroDeActividad
NodoDeActividadEstructurada
RegiónDeExpansión
NodoDeExpansión
RegiónDeActividadInterrumpible

Nota
Enlace de nota

15.2 Diagramas de clases

Relaciones

Asociación
Agregación
Composición
ClaseDeAsociación
Dependencia
Utilización
RealizaciónDeInterfaz
Generalización

Agregar

Paquete
Clase
Interfaz
Enumeración
TipoDeDatos
TipoPrimitivo
Perfil
Estereotipo
AplicaciónDePerfil
EspecificaciónDeInstancia

Nota
Enlace de nota

15.3 Diagramas de comunicación

Agregar

LíneaDeVida
Mensaje (Llamada)
Mensaje (Respuesta)
Mensaje (Creación)
Mensaje (Destrucción)

Nota
Enlace de nota

15.4 Diagramas de estructura de un compuesto

Agregar

Colaboración
UsoDeColaboración
Parte (Propiedad)
Clase
Interfaz
Puerto

Relaciones

Conector
Dependencia (Enlace de roles)
RealizaciónDeInterfaz
Utilización

Nota
Enlace de nota

15.5 Diagramas de componentes

Agregar

Paquete
Interfaz
Clase
Componente
Artefacto

Relaciones

Realización
RealizaciónDeInterfaz
Utilización
Dependencia

Nota
Enlace de nota

15.6 Diagramas de implementación

Agregar

Paquete
Componente
Artefacto
Nodo
Dispositivo
EntornoDeEjecución

Relaciones

Manifestación
Implementación
Asociación
Generalización
Dependencia

Nota
Enlace de nota

15.7 Diagramas global de interacción

Agregar

AcciónLlamadaDeComportamiento (Interacción)
AcciónLlamadaDeComportamiento (UsoDeInteracción)
NodoDeDecisión
NodoDeCombinación
NodoInicial
NodoFinalDeActividad
NodoDeBifurcación
NodoDeBifurcación (Horizontal)
NodoDeReunión
NodoDeReunión (Horizontal)
RestricciónDeDuración

Relaciones

FlujoDeControl

Nota
Enlace de nota

15.8 Diagramas de objetos

Agregar

Paquete
Clase
Interfaz
Enumeración
TipoDeDatos
TipoPrimitivo
EspecificaciónDeInstancia

Relaciones

Asociación
ClaseDeAsociación
Dependencia
Utilización
RealizaciónDeInterfaz
Generalización

Nota
Enlace de nota

15.9 Diagramas de paquetes

Agregar

Paquete
Perfil

Relaciones

Dependencia
ImportaciónDePaquete
CombinaciónDePaquete
AplicaciónDePerfil

Nota
Enlace de nota

15.10 Diagramas de perfil

Agregar

Perfil
Estereotipo

Relaciones

Generalización
AplicaciónDePerfil
ImportaciónDePaquete
ImportaciónDeElemento

Nota
Enlace de nota

15.11 Diagramas de máquina de estados de protocolos

Agregar

Estado
Estado compuesto
Estado ortogonal
Estado de submáquina

EstadoFinal
EstadoInicial

PuntoDeEntrada
PuntoDeSalida
Elección
Unión
Terminar
Bifurcación
Bifurcación (horizontal)
Reunión
Reunión (horizontal)
ReferenciaDePuntoDeConexión

Relaciones

TransiciónDeProtocolo

Nota
Enlace de nota

15.12 Diagramas de secuencia

Agregar

LíneaDeVida
FragmentoCombinado
FragmentoCombinado (Alternativos)
FragmentoCombinado (Bucle)
UsoDeInteracción
Puerta
InvarianteDeEstado
RestricciónDeDuración
RestricciónDeTiempo

Mensaje (Llamada)
Mensaje (Respuesta)
Mensaje (Creación)
Mensaje (Destrucción)

Mensaje asíncrono (Llamada)
Mensaje asíncrono (Respuesta)
Mensaje asíncrono (Destrucción)

Mensajes sin numeración
Mensajes con numeración sencilla
Mensajes con notación decimal anidada

Nota
Enlace de nota

Activar/desactivar el movimiento de mensajes dependientes
Activar/desactivar la creación automática de respuestas para mensajes (Llamada)
Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación

15.13 Diagramas de máquina de estados

Agregar

Estado
Estado compuesto
Estado ortogonal
Estado de submáquina

EstadoFinal
EstadoInicial

PuntoDeEntrada
PuntoDeSalida
Elección
Unión
Terminar
Bifurcación
Bifurcación (horizontal)
Reunión
Reunión (horizontal)
HistorialDetallado
HistorialSuperficial
ReferenciaDePuntoDeConexión

Relaciones

Transición

Nota
Enlace de nota

Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación

15.14 Diagramas de ciclo de vida

Agregar

LíneaDeVida (Estado o Condición)
LíneaDeVida (Valor general)
MarcaDeGraduación
Evento/estímulo
RestricciónDeDuración
RestricciónDeTiempo

Mensaje (Llamada)
Mensaje (Respuesta)
Mensaje asíncrono (Llamada)

Nota
Enlace de nota

15.15 Diagramas de casos de uso

Agregar

Paquete
Actor
CasoDeUso

Relaciones

Asociación
Generalización
Inclusión
Extensión

Nota

15.16 Diagramas de esquema XML

Agregar

targetNamespace XSD
schema XSD
element (global) XSD
group XSD
complexType XSD
complexType (simpleContent) XSD
simpleType XSD
list XSD
union XSD
enumeration XSD
attribute XSD
attributeGroup XSD
notation XSD
import XSD

Relaciones

include XSD
redefine XSD
restriction XSD
extension XSD
substitution XSD

Nota
Enlace de nota

15.17 Diagramas BPMN

Agregar

Evento inicial
Evento intermedio
Evento final

Tarea
Tarea de ciclo
Tarea (Instancia múltiple)
Tarea (Compensación)

Subproceso contraído
Subproceso de ciclo contraído
Subproceso de instancia múltiple contraído
Subproceso ad hoc contraído
Subproceso de compensación contraído

Subproceso expandido
Subproceso de ciclo expandido
Subproceso de instancia múltiple expandido
Subproceso ad hoc expandido
Subproceso de compensación expandido

Compuerta
Compuerta inclusiva (OR)
Compuerta paralela (AND)
Compuerta exclusiva basada en datos (XOR)
Compuerta exclusiva basada en eventos (XOR)
Compuerta compleja (bifurcación/convergencia)

Relaciones

Flujo de secuencia
Flujo condicional
Flujo por defecto
Flujo de mensaje
Asociación

Contenedor
Objeto de datos
Grupo

Anotación textual
Asociación por anotación

15.18 Diagramas BPMN 2.0

Agregar

Evento inicial
Evento de captura
Evento de lanzamiento
Evento final

Tarea
Subproceso expandido
Subproceso contraído
Actividad de llamada
Compuerta

Relaciones

Flujo de secuencia
Flujo de secuencia predeterminado
Flujo de secuencia condicional
Flujo de mensaje
Asociación

Contenedor
Grupo
Objeto de datos
Datos de salida
Datos de entrada
Colección de objetos de datos
Almacén de datos
Mensaje

Anotación textual
Asociación por anotación

15.19 Modelado de bases de datos

Agregar

Tabla
RestricciónDeComprobación
ClavePrincipal
ClaveForánea
ClaveÚnica
Índice

Relaciones

Asociación de relación entre bases de datos
Relación entre bases de datos con atributos

16 Referencia del usuario

Esta sección repasa todos los menús y comandos de menú de UModel, dando una breve descripción de cada uno de ellos.

16.1 Menú Archivo

Nuevo

Si tiene abierto un proyecto, este comando lo cierra y crea un proyecto nuevo.

Abrir

Abre proyectos de modelado previamente definidos. En el cuadro de diálogo "Abrir" seleccione un archivo de proyecto guardado previamente (archivos ***.ump**). Para más información consulte los apartados Crear, abrir y guardar proyectos y Abrir proyectos desde una URL.

Volver a cargar

Este comando sirve para volver a cargar el proyecto y guardar (o descartar) los cambios realizados hasta ese momento.

Guardar

Este comando guarda el proyecto de modelado activo con el nombre de archivo actual.

Guardar como

Este comando guarda el proyecto de modelado activo con otro nombre. También ofrece la opción de darle al proyecto un nombre si es la primera vez que se guarda.

Guardar copia como

Este comando guarda una copia del proyecto de modelado activo con otro nombre de archivo.

Guardar el diagrama como imagen

Este comando abre el cuadro de diálogo "Guardar como" y permite guardar el diagrama activo como archivo .PNG. También puede guardar archivos PNG de gran tamaño (1GB o más).

Guardar todos los diagramas como imagen

Este comando guarda todos los diagramas del proyecto activo como archivos .PNG.

Importar desde un archivo XMI

Este comando importa un archivo XMI exportado con anterioridad. Si el archivo se generó con UModel, toda las extensiones y demás elementos se conservarán.

Exportar a un archivo XMI

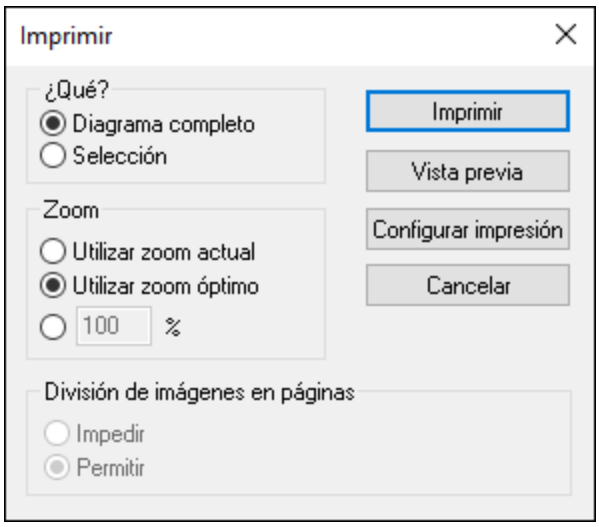
Este comando exporta el modelo como archivo XMI (*imagen siguiente*). Puede seleccionar la versión de UML y los identificadores que desea exportar. Para más información consulte el apartado XMI: intercambio de metadatos XML.

Enviar por correo electrónico

Este comando abre la aplicación predeterminada de correo electrónico e inserta el proyecto de UModel actual como archivo adjunto.

Imprimir

Este comando abre el cuadro de diálogo "Imprimir" (*imagen siguiente*), desde donde puede imprimir una copia en papel del diagrama activo en ese momento (o de una selección del mismo).



- *Utilizar zoom actual:* seleccione esta opción si quiere usar el factor de zoom actual del proyecto de modelado. Si selecciona esta opción se habilita el grupo de opciones *División de imágenes en páginas.*
- *Utilizar zoom óptimo:* elija esta opción para ajustar el tamaño del proyecto de modelado al tamaño de la página. También puede especificar el factor de zoom numéricamente.
- *Impedir:* seleccione esta opción para que los elementos de modelado no se dividan en dos páginas y se mantengan como una unidad.

Imprimir todos los diagramas

Este comando abre el cuadro de diálogo "Imprimir" e imprime todos los diagramas UML que contiene el archivo de proyecto actual.

Vista previa de impresión

Este comando abre el cuadro de diálogo "Imprimir".

Configurar impresión

Este comando abre el cuadro de diálogo "Configurar impresión", donde puede seleccionar la impresora que desea usar y definir la configuración del papel.

16.2 Menú Edición

Deshacer 

UModel permite eliminar todos los cambios realizados y devolver el archivo a versiones anteriores. Todos los cambios se pueden deshacer uno por uno y no hay un límite de operaciones deshacer.

Rehacer 

Permite rehacer las acciones que deshizo con el comando **Deshacer**. Esto significa que puede navegar por el historial de acciones con los comandos **Deshacer** y **Rehacer**.

Cortar/Copiar/Pegar/Eliminar

Estos son los comandos estándar de edición de Windows. Puede usarlos tanto con texto como con elementos de modelado. Para más información consulte el apartado Renombrar, mover y copiar elementos.

Pegar solo en el diagrama

Este comando añade un vínculo (o vista) del elemento copiado al diagrama actual pero no a la *Estructura del modelo*. Para más información consulte el apartado Renombrar, mover y copiar elementos.

Eliminar solo en el diagrama

Este comando elimina los elementos seleccionados del diagrama activo. Sin embargo, los elementos no se eliminan del proyecto de modelado y siguen estando disponibles en la *Estructura del modelo*. Recuerde que este comando no sirve para eliminar propiedades ni operaciones de clase. Las propiedades y operaciones se pueden seleccionar y eliminar en la clase directamente.

Seleccionar todo

Este comando selecciona todos los elementos de modelado del diagrama activo. Equivale a utilizar **Ctrl+A**.

Buscar

Este comando permite buscar texto en la ventana activa. Para más información consulte el apartado Buscar y reemplazar texto.

Buscar siguiente  (F3)

Este comando repite la última búsqueda realizada con el comando **Buscar** y busca la siguiente instancia del término de búsqueda en la ventana activa.

Buscar anterior (Mayús+F3)

Este comando repite la última búsqueda realizada con el comando **Buscar** y busca la instancia anterior del término de búsqueda en el diagrama o en la pestaña activos.

Reemplazar

Este comando busca y reemplaza elementos de modelado en el proyecto. Para más información consulte el apartado Buscar y reemplazar texto.

Copiar como mapa de bits

Este comando copia el diagrama activo en el portapapeles. Después podrá pegar el diagrama en cualquier aplicación.

Copiar la selección como mapa de bits

Este comando copia los elementos de diagrama **seleccionados** en el portapapeles. Después podrá pegarlos en cualquier aplicación.

16.3 Menú Proyecto

Revisar la sintaxis del proyecto...

Este comando sirve para revisar sintaxis del proyecto de UModel (*véase Revisar la sintaxis del proyecto*).

Control de código fuente

Consulte el apartado Sistemas de control de código fuente que ofrece información detallada sobre servidores y clientes de control de código fuente y cómo usarlos.

Importar directorio de código fuente...

Abre el asistente "Importar directorio de código fuente" (*puede ver un ejemplo de uso en el apartado Ingeniería inversa (del código al modelo)*) de la documentación).

Importar proyecto de código fuente...

Abre el asistente "Importar directorio de código fuente" (*véase Importar código fuente*).

Importar tipos binarios

Abre el cuadro de diálogo "Importar tipos binarios", que sirve para importar archivos Java, C# y VB binarios. Para más información consulte el apartado Importar archivos binarios Java, C# y VB.

Importar directorio del esquema XML

Abre el cuadro de diálogo "Importar el directorio del esquema XML", que sirve para importar todos los esquemas XML del directorio seleccionado y también los de sus subdirectorios.

Importar archivo de esquema XML

Abre el cuadro de diálogo "Importar archivo de esquema XML", que sirve para importar archivos de esquema (*véase Diagramas de esquema XML*).

Importar base de datos SQL

Abre el cuadro de diálogo "Importar BD", desde donde puede importar estructuras de BD en el modelo, consulte Importar bases de datos SQL en UModel.

Generar diagramas de secuencia a partir del código

Véase Crear varios diagramas de secuencia.

Combinar/sobrescribir el código de programa con el proyecto de UModel

Actualiza el código de programa con el modelo, suponiendo que su proyecto esté configurado para la ingeniería de código (véase Generar código de programa). El nombre de este comando puede ser **Combinar el código de programa con el proyecto de UModel** o **Sobrescribir el código de programa con el proyecto de UModel**, dependiendo de la configuración elegida en el cuadro de diálogo "Configurar sincronización). El comportamiento predeterminado de la aplicación es abrir el cuadro de diálogo "Configurar

sincronización" cada vez que se ejecute este comando. Para más información consulte el apartado Configurar la sincronización del código.

Combinar/sobrescribir el proyecto de UModel con el código de programa

Actualiza el modelo (el proyecto de UModel) con el código de programa. El nombre de este comando puede ser **Combinar el proyecto de UModel con el código de programa** o **Sobrescribir el proyecto de UModel con el código de programa**, dependiendo de la configuración elegida en el cuadro de diálogo "Configurar sincronización). El comportamiento predeterminado de la aplicación es abrir el cuadro de diálogo "Configurar sincronización" cada vez que se ejecute este comando. Para más información consulte el apartado Configurar la sincronización del código.

Configurar sincronización...

Abre el cuadro de diálogo "Configurar sincronización" (*véase Configurar la sincronización del código*).

Transformación del modelo

Inicia un asistente que permite pasar el modelo de un lenguaje a otro (por ejemplo, de Java a C#), consulte Transformar modelos UML.

Combinar el proyecto...

Combina dos proyectos de UModel en uno solo modelo. El primer archivo que se abre es con el que se combina el segundo archivo. Consulte el apartado Combinar proyectos de UModel para obtener más información.

Incluir un subproyecto

Consulte el apartado Incluir otros proyectos de UModel.

Abrir en forma de proyecto

Este comando abre el subproyecto seleccionado como un proyecto nuevo.

Borrar mensajes

Este comando borra los mensajes, errores y advertencias de revisión de sintaxis y de combinación de código de la ventana Mensajes.

Nota: los errores informan de problemas que deben solucionarse inmediatamente para poder generar código o para poder actualizar el código del modelo. Las advertencias, por lo general, se pueden dejar para más tarde. En UModel los errores y las advertencias los generan la función de revisión de sintaxis, el compilador de cada lenguaje de programación, el analizador de UModel que lee los archivos fuente generados y la función de importación XMI.

Generar documentación

Este comando sirve para generar documentación para el archivo activo en formato HTML, Word y RTF. (*Consulte el apartado Generar documentación UML para obtener más información*).

Mostrar elementos no utilizados en ningún diagrama

Este comando crea una lista con todos los elementos que no se utilizan en ningún diagrama del proyecto (*véase* *Comprobar si se están usando ciertos elementos y dónde*).

Mostrar paquetes compartidos

Este comando crea una lista con todos los paquetes compartidos del proyecto actual.

Mostrar paquetes incluidos

Este comando crea una lista con todos lo paquetes incluidos en el proyecto actual.

16.4 Menú Diseño

Los comandos del menú **Diseño** sirven para alinear y poner en fila los elementos de los diagramas de modelado (*véase* *Alinear y ajustar el tamaño de elementos de modelado*).

Alinear

Este grupo de comandos sirve para alinear los elementos de modelado con el borde elegido o en el centro, según la opción elegida.

Espaciar uniformemente

Este grupo de comandos sirve para espaciar los elementos seleccionados en horizontal o en vertical de manera uniforme.

Igualar tamaño

Este grupo de comandos sirve para ajustar el ancho y el alto de los elementos seleccionados al ancho o alto del elemento activo.

Poner en fila

Este grupo de comandos sirve para poner los elementos seleccionados en fila vertical u horizontal.

Estilo de la línea

Este grupo de comandos permite elegir el tipo de línea que se utiliza para conectar los diferentes elementos de modelado. Las líneas pueden ser líneas de dependencia o de asociación.

Tamaño automático

Este comando ajusta el tamaño de los elementos seleccionados a las dimensiones óptimas para cada uno de ellos.

Aplicar diseño automático a todo

Este comando sirve para elegir el tipo de presentación para los elementos de modelado del diagrama UML activo:

Diseño dirigido por fuerzas	Muestra los elementos de modelado desde un punto de vista céntrico.
Diseño jerárquico	Muestra los elementos en función de sus relaciones jerárquicas. Por ejemplo, una superclase se colocará por encima de cualquiera de sus clases derivadas. Se pueden ajustar las opciones del diseño jerárquico en el menú **Herramientas | Opciones**, pestaña **Vista**, grupo **Autodiseño de la jerarquía**.
Diseño por bloques	Muestra los elementos en un rectángulo y agrupados por tamaño.

Ajustar la posición de las etiquetas de texto

Este comando pone el nombre de los elementos (de todos o de los seleccionados) en su posición predeterminada.

16.5 Menú Vista

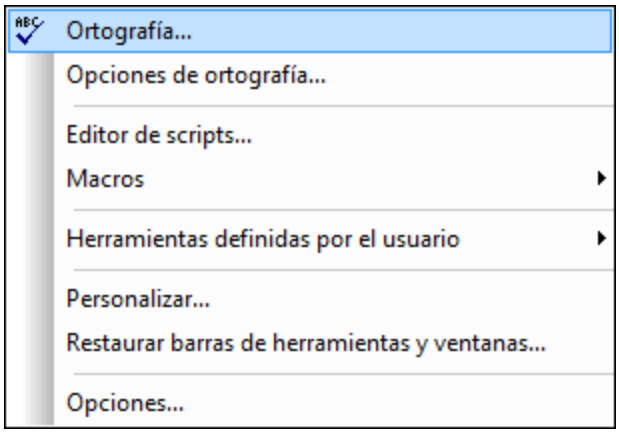
Los comandos de este menú sirven para:

- mostrar u ocultar las ventanas auxiliares de UModel (*véase* *Interfaz gráfica del usuario de UModel*).
- definir el criterio de ordenación en los elementos que están dentro de los paneles *Estructura del modelo* y *Favoritos*.
- definir el criterio de agrupación de los diagramas en el panel *Árbol de diagramas*.
- mostrar/ocultar determinados elementos UML en los paneles *Estructura del modelo* y *Favoritos*.
- definir el factor de zoom del diagrama actual (*véase* *Acercar y alejar diagramas*).

16.6 Menú Herramientas

Los comandos del menú **Herramientas** sirven para:

- Revisar la ortografía del proyecto de UModel y configurar la revisión ortográfica.
- Abrir el entorno de scripting de UModel. En el entorno de scripting puede crear, gestionar y almacenar formularios, macros y controladores de eventos.
- Ver y ejecutar las macros que ya están definidas.
- Personalizar la aplicación definiendo barras de herramientas, teclas de acceso rápido, menús y macros personales.
- Restaurar las barras de herramientas y ventanas de la aplicación a su estado predeterminado de instalación.
- Definir opciones de configuración globales para la aplicación.

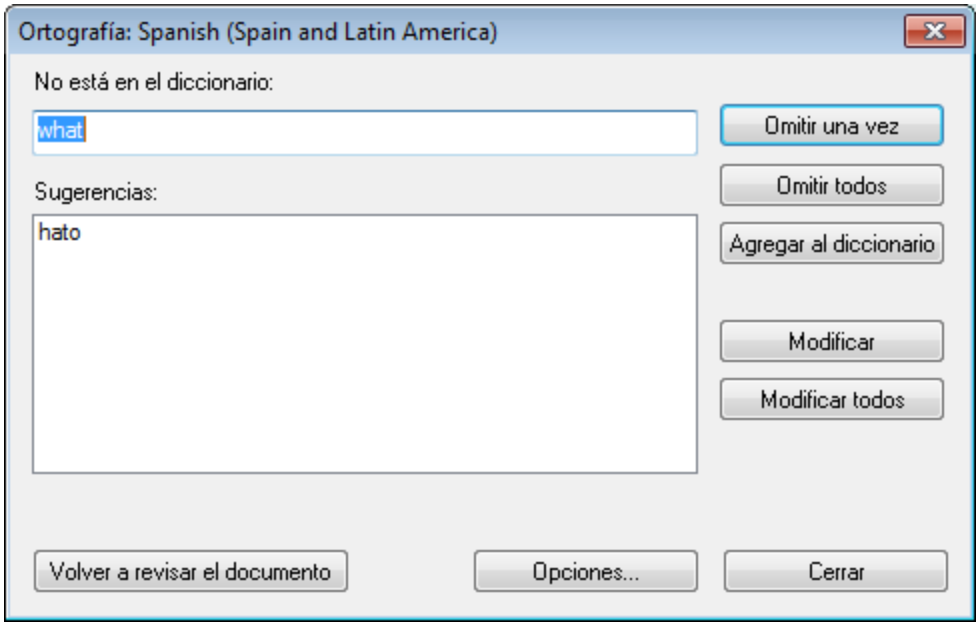


16.6.1 Ortografía

Este comando del menú **Herramientas** sirve para revisar la ortografía del proyecto. Al hacer clic en él, aparece el cuadro de diálogo "Ortografía", que incluye opciones de revisión ortográfica estándar.

Para configurar opciones más específicas haga clic en **Opciones...** o seleccione el comando **Herramientas | Opciones de ortografía**.

Puede revisar la ortografía de las entradas de la *Estructura del modelo* y de los diagramas UML. Si hace clic con el botón derecho en la *Estructura del modelo* y selecciona **Ortografía de la documentación** en el menú contextual, se revisan la ortografía de los comentarios y las notas de la *Estructura del modelo*.



No está en el diccionario

Este cuadro de texto contiene la palabra que no se encontró ni en el diccionario del idioma seleccionado ni en el diccionario del usuario.

Sugerencias

Este cuadro de lista muestra palabras similares a la palabra desconocida. Haga doble clic en una palabra de esta lista para insertarla automáticamente en el documento y continuar la revisión ortográfica.

Omitir una vez

Pulse este botón para continuar revisando la ortografía del documento y pasar por alto la primera instancia de la palabra desconocida. Si esa palabra vuelve a aparecer, la revisión ortográfica la señalará.

Omitir todos

Pulse este botón para pasar por alto todas las instancias de la palabra desconocida en todo el documento.

Agregar al diccionario

Pulse este botón para añadir la palabra desconocida al diccionario del usuario. Puede acceder al diccionario del usuario desde el cuadro de diálogo "Opciones".

Modificar

Pulse este botón para reemplazar la palabra que está resaltada en el documento con la palabra editada del cuadro *No está en el diccionario*.

Modificar todos

Pulse este botón para reemplazar todas las instancias de la palabra que está resaltada con la palabra editada del cuadro *No está en el diccionario*.

Volver a revisar el documento

Pulse este botón para reiniciar la revisión ortográfica desde el principio del documento.

Agregar diccionarios para el corrector ortográfico

Por cada idioma hay dos archivos de diccionario Hunspell que funcionan conjuntamente: un archivo `.aff` y un archivo `.dic`. Los diccionarios se instalan en la carpeta `Lexicons` de este directorio: `C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons`.

Dentro de la carpeta `Lexicons` se crea una carpeta por idioma: `<nombre del idioma>\<archivos del diccionario>`. Por ejemplo, los archivos de los dos diccionarios de español (de España y de Hispanoamérica y España) se almacenan así:

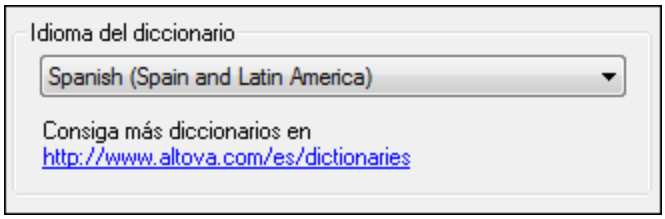
```
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (British)
\en_GB.aff
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (British)
\en_GB.dic
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)\en_US.aff
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)\en_US.dic
```

La lista desplegable *Idioma del diccionario* del cuadro de diálogo "Opciones de ortografía" muestra los diccionarios disponibles. Se trata de los diccionarios de la carpeta `Lexicons` y reciben el nombre de las subcarpetas de la carpeta `Lexicons`. Por ejemplo, los dos diccionarios de inglés del ejemplo anterior aparecerían así en el cuadro combinado: *Spanish (Spain)* y *Spanish (Spain and Latin America)*.

Todos los diccionarios instalados son compartidos por los diferentes usuarios del equipo y por las diferentes versiones de los productos de Altova (tanto en 64 como en 32 bits).

Hay dos maneras de agregar diccionarios nuevos para el corrector ortográfico. En ninguno de los dos casos es necesario registrar los archivos en el sistema:

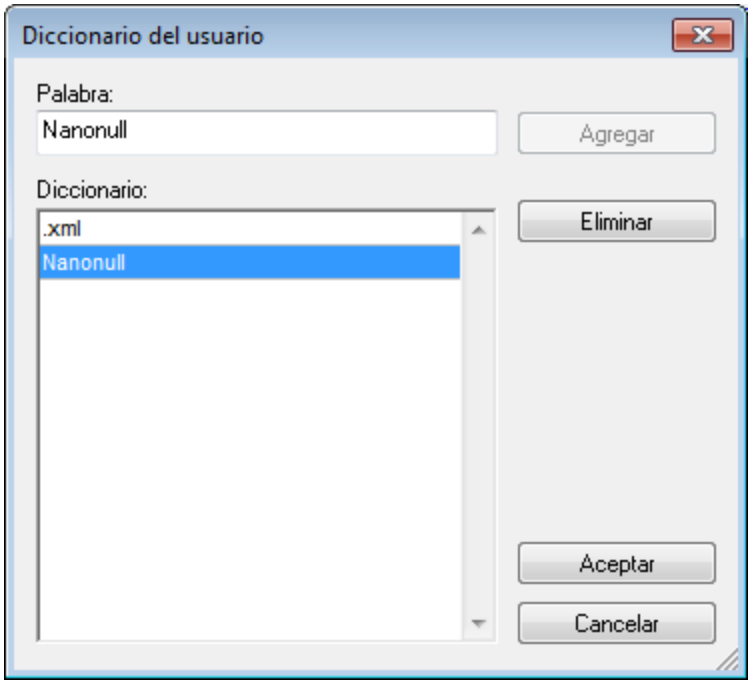
- Puede añadir diccionarios Hunspell a una subcarpeta nueva de la carpeta `Lexicons`. Los diccionarios Hunspell se pueden descargar desde https://wiki.services.openoffice.org/wiki/Dictionaries o desde https://extensions.services.openoffice.org/en/dictionaries, por ejemplo. (Recuerde que OpenOffice utiliza el formato comprimido `OXT`. Cambie la extensión a `.zip` y descomprima los archivos `.aff` y `.dic` en las subcarpetas correspondientes de la carpeta `Lexicons`. También puede usar diccionarios Myspell, ya que los diccionarios Hunspell están basados en Myspell.)
- Puede usar el instalador de diccionarios de Altova, que instala un paquete con varios diccionarios en el directorio adecuado del equipo. En el cuadro de diálogo "Opciones de ortografía", bajo el panel *Idioma del diccionario*, aparece un enlace a la página de Altova de descarga de diccionarios (*imagen siguiente*). Si no usa derechos de administrador para instalar los diccionarios, se producirá un error de instalación.



Nota: recuerde que es decisión suya aceptar o no las condiciones de uso de la licencia del diccionario elegido. También es responsabilidad suya comprobar si el diccionario puede utilizarse en su equipo o no.

Trabajar con el diccionario del usuario

Cada usuario tiene un diccionario del usuario propio, donde se almacenan las palabras aprobadas por el usuario. Durante la revisión ortográfica, el corrector compara la ortografía con una lista compuesta por palabras del diccionario integrado y del diccionario del usuario. Puede añadir o eliminar palabras del diccionario del usuario en el cuadro de diálogo "Diccionario del usuario" (*imagen siguiente*) Para abrir este cuadro de diálogo pulse el botón **Diccionario del usuario** del cuadro de diálogo "Opciones de ortografía" (*segunda imagen de este apartado*).



Para añadir una palabra al diccionario del usuario escriba la palabra en el recuadro *Palabra* y pulse el botón **Agregar**. La palabra se añade a la lista alfabética del panel *Diccionario.* Para eliminar una palabra del diccionario, seleccione la palabra del panel *Diccionario* y pulse el botón **Eliminar**. La palabra se elimina del panel *Diccionario*. Cuando termine de editar el cuadro de diálogo "Diccionario del usuario" haga clic en el botón **Aceptar**. Los cambios se guardan en el diccionario del usuario.

También puede añadir palabras al diccionario del usuario durante la revisión ortográfica. Si el corrector encuentra una palabra desconocida, aparece el cuadro de diálogo Ortografía. Pulse el botón **Agregar al diccionario** para añadir la palabra desconocida al diccionario del usuario.

El diccionario del usuario se encuentra en este directorio: `C:\Users\<usuario>\Documentos\Altova\SpellChecker\Lexicons\user.dic`

16.6.2 Opciones de ortografía...

Este comando del menú **Herramientas** abre el cuadro de diálogo "Opciones de ortografía UML" (*imagen siguiente*). Este cuadro de diálogo sirve para configurar la revisión ortográfica de elementos UML. A continuación se describen sus opciones.

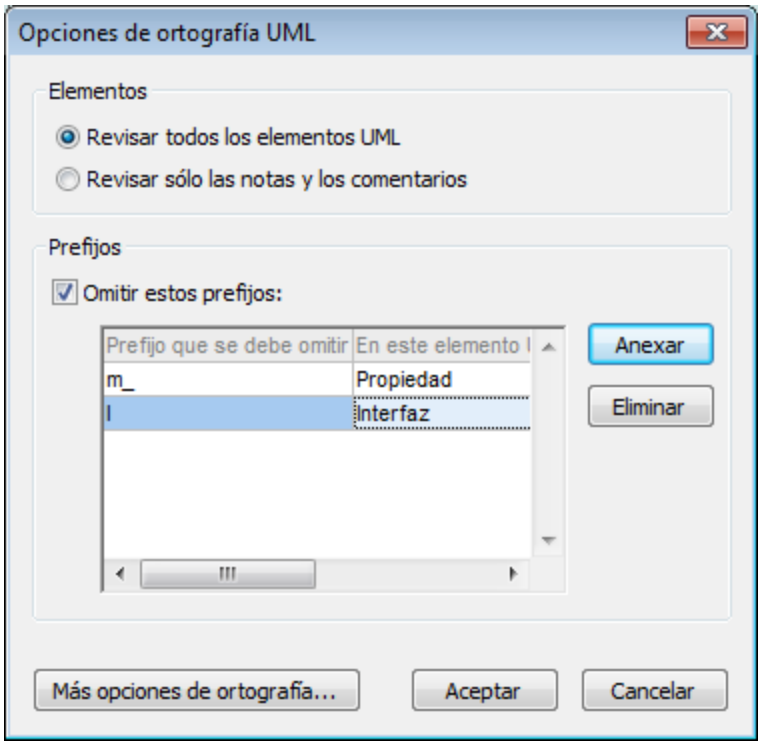
Elementos

En este grupo de opciones puede elegir si se revisan todos los elementos UML o solamente las notas y los comentarios.

Prefijos

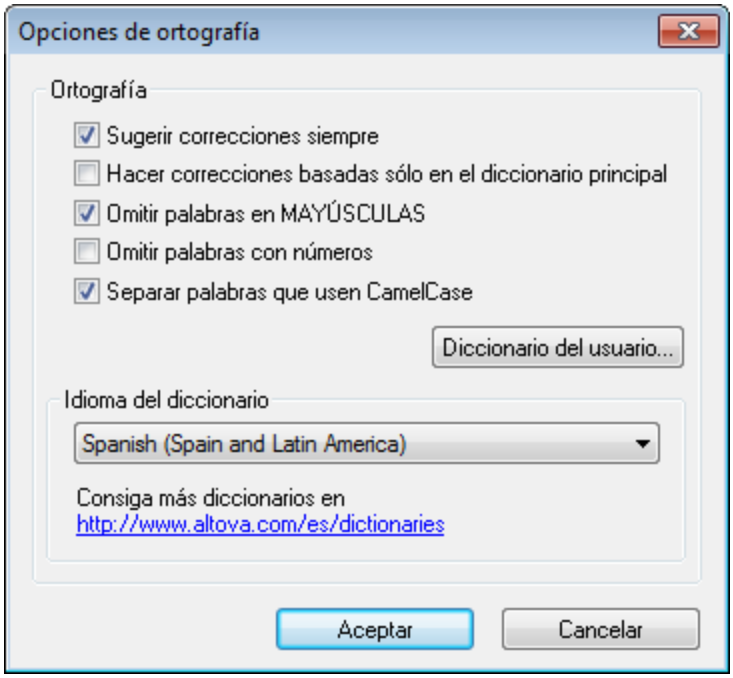
Haga doble clic en la columna *Prefijo que se debe omitir* para insertar los prefijos de los elementos UML que desea pasar por alto durante la revisión ortográfica (p. ej. m_ para las propiedades y I_ para las interfaces).

El botón **Anexar** añade una fila nueva a la tabla de prefijos. El botón **Eliminar** elimina la fila seleccionada de la tabla.



Más opciones de ortografía...

Este botón abre el cuadro de diálogo "Opciones de ortografía" (*imagen siguiente*), que sirve para modificar la configuración global del corrector ortográfico. Sus opciones se describen más abajo.



Sugerir correcciones siempre
Si marca esta casilla, el cuadro *Sugerencias* siempre mostrará opciones del diccionario integrado elegido y del diccionario del usuario. Si desactiva esta opción, el corrector no ofrecerá ninguna sugerencia.

Hacer correcciones basadas sólo en el diccionario principal
Si marca esta casilla, solamente se usan sugerencias del diccionario integrado elegido (diccionario principal). El diccionario del usuario no se utilizará para ofrecer sugerencias. Si marca esta casilla, se deshabilita el botón **Diccionario del usuario**. Es decir, mientras esté activa esta opción el diccionario del usuario no se podrá editar.

Omitir palabras en MAYÚSCULAS
Si marca esta casilla, el corrector ortográfico pasa por alto las palabras que están en mayúsculas.

Omitir palabras con números
Si marca esta casilla, el corrector ortográfico pasa por alto las palabras que contienen números.

Separar palabras que usen CamelCase
Las palabras que usan CamelCase están formadas por palabras que empiezan por mayúsculas, unidas unas a otras sin espacios. Por ejemplo, la palabra *CamelCase* está formada por las palabras *Camel* y *Case*, ambas en mayúsculas y unidas sin espacios. Este tipo de palabras no suele aparecer en los diccionarios y, por tanto, el corrector ortográfico las marcará como erróneas. Marque la casilla *Separar palabras que usen CamelCase* para que el corrector revise por separado cada una de las palabras que forman la palabra CamelCase. Esta opción está marcada por defecto.

Idioma del diccionario
Use este cuadro combinado para seleccionar el idioma del diccionario integrado del corrector ortográfico. La opción predeterminada es **US English** (inglés EE UU). Para descargar gratis más diccionarios en otros idiomas, visite el sitio web de Altova.

16.6.3 Editor de scripts...

Este comando del menú **Herramientas** abre la ventana del editor de script (*véase el apartado Editor de scripts*).

Nota: es necesario tener instalado .NET Framework versión 2.0 (o superior) para poder ejecutar el editor de scripts.

16.6.4 Macros

Muestra un submenú con todas las macros definidas en el proyecto de script activo (*véase el apartado Editor de scripts*). El proyecto de script activo se define en la pestaña *Script* del cuadro de diálogo "Opciones locales".

16.6.5 Herramientas definidas por el usuario

Al pasar el cursor por en el comando **Herramientas definidas por el usuario** aparece un submenú con comandos hechos a medida que usan aplicaciones externas. Para crear estos comandos, use la pestaña *Herramientas* del cuadro de diálogo "Personalizar". Al hacer clic en uno de estos comandos personalizados, se ejecuta la acción asociada al comando.

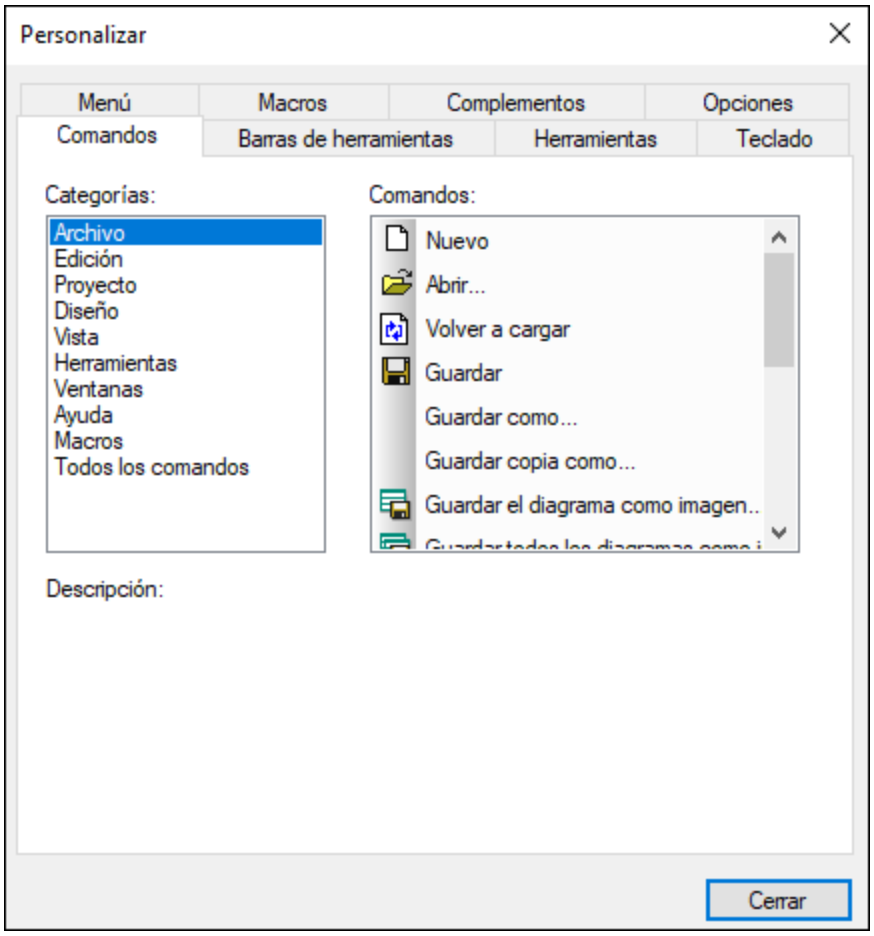
16.6.6 Personalizar

El comando **Personalizar** abre el cuadro de diálogo "Personalizar", que sirve para configurar los siguientes elementos de la interfaz gráfica de UModel:

- *Comandos*
- *Barras de herramientas*
- *Herramientas*
- *Teclado*
- *Menú*
- *Macros*
- *Complementos*
- *Opciones*

16.6.6.1 Comandos

En la pestaña *Comandos* puede personalizar los menús y las barras de herramientas de UModel.



Para añadir un comando a una barra de herramientas o menú:

1. Seleccione el comando **Herramientas | Personalizar**.
2. Seleccione la pestaña ***Comandos***. En el cuadro de lista *Categorías* seleccione la opción **Todos los comandos**. Todos los comandos disponibles aparecen en el cuadro de lista *Comandos*.
3. Haga clic en un comando del cuadro de lista *Comandos* y arrástrelo a un menú o barra de herramientas que ya existe. Al pasar el puntero por encima de una posición donde se puede colocar el comando aparece el icono **I**.
4. Cuando encuentre la posición donde desea colocar el comando, suelte el botón del ratón.
5.

Tenga en cuenta que:

- Si pasa el cursor por un menú que está cerrado, el menú se abre y puede insertar el comando en cualquier parte del menú.
- Si el comando no se puede colocar en la posición actual del cursor, debajo del puntero aparece una **X**.

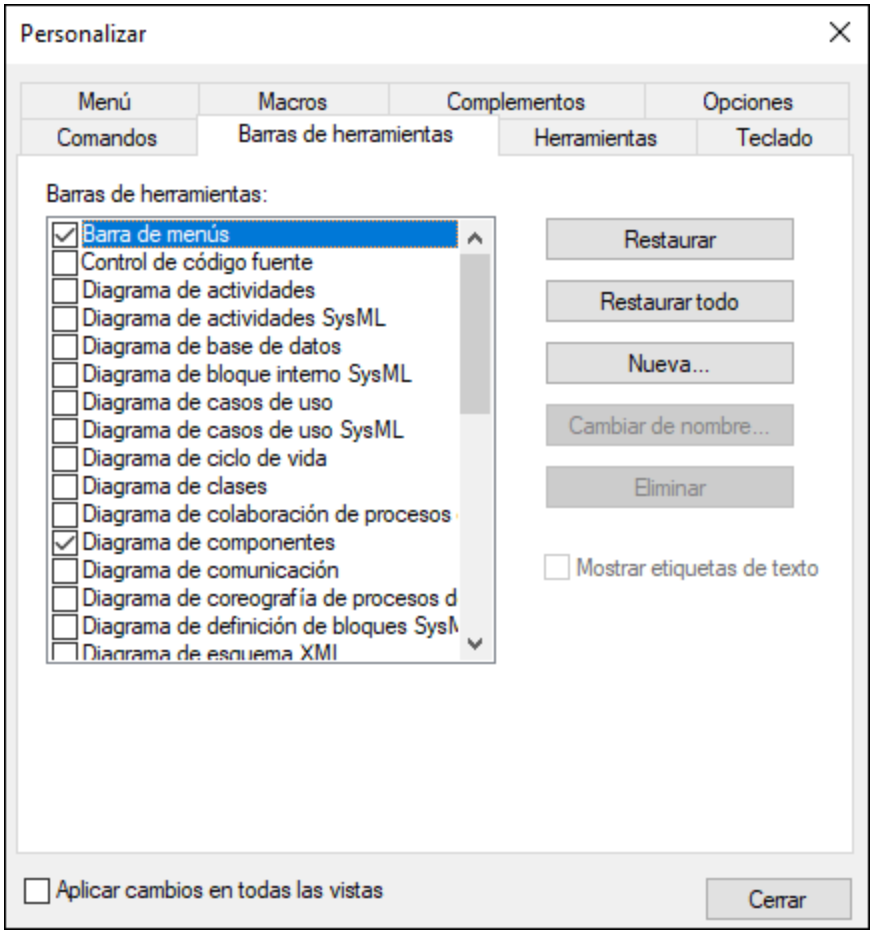
- Si el cursor está en una posición donde se puede colocar el comando (en una barra de herramientas o en un menú), la **X** desaparece y el icono **I** indica que la posición es válida.
- Los comandos se pueden colocar en menús o barras de herramientas. Si creó una barra de herramientas nueva, puede usar este mecanismo de personalización para rellenar la barra de herramientas con comandos.
- También puede añadir comandos a los menús contextuales (los que se abren al hacer clic con el botón derecho en cualquier parte) haciendo clic en la pestaña *Menú* del cuadro de diálogo "Personalizar" y seleccionando un menú contextual del panel *Menús contextuales,* o bien arrastrando comandos del cuadro de lista *Comandos* hasta el menú contextual.

Para eliminar un comando o menú:

1. Seleccione el comando **Herramientas | Personalizar**.
2. Seleccione cualquier pestaña del cuadro de diálogo "Personalizar". Haga clic con el botón derecho en un menú o comando de menú y seleccione **Eliminar** en el menú contextual que aparece. Si lo prefiere, también puede arrastrar el menú o comando de menú hasta que aparezca el icono **X** debajo del puntero del ratón y suelte el menú o comando de menú. Como resultado se elimina el menú o comando de menú.

16.6.6.2 Barras de herramientas

En la pestaña *Barras de herramientas* puede: (i) activar o desactivar barras de herramientas (es decir, decidir qué barras de herramientas aparecen en la interfaz), (ii) definir qué iconos aparecen en cada barra de herramientas y (iii) crear barras de herramientas personalizadas.



Las barras de herramientas incluyen iconos para los comandos de menú más utilizados. Además, al pasar el puntero sobre un icono, se ofrece información rápida sobre el icono en un mensaje emergente y en la barra de estado de la aplicación. Las barras de herramientas se pueden colocar en cualquier posición de la pantalla, donde aparece como ventana flotante.

Para activar/desactivar una barra de herramientas:

- Marque la casilla de la barra de herramientas en el cuadro de lista *Barras de herramientas*.

Para añadir una barra de herramientas nueva:

1. Pulse el botón **Nueva...** y escriba el nombre de la barra de herramientas nuevas en el cuadro de diálogo "Nombre de la barra de herramientas" que aparece.
2. Arrastre comandos desde la pestaña *Comandos* hasta la barra de herramientas nueva.

Para restaurar la barra de menús:

1. Seleccione `Barra de menús` en el panel *Barras de herramientas* y pulse el botón **Restaurar**.

2. La barra de menús vuelve a su estado original de instalación.

Para restaurar todas las barras de herramientas y comandos de menú:

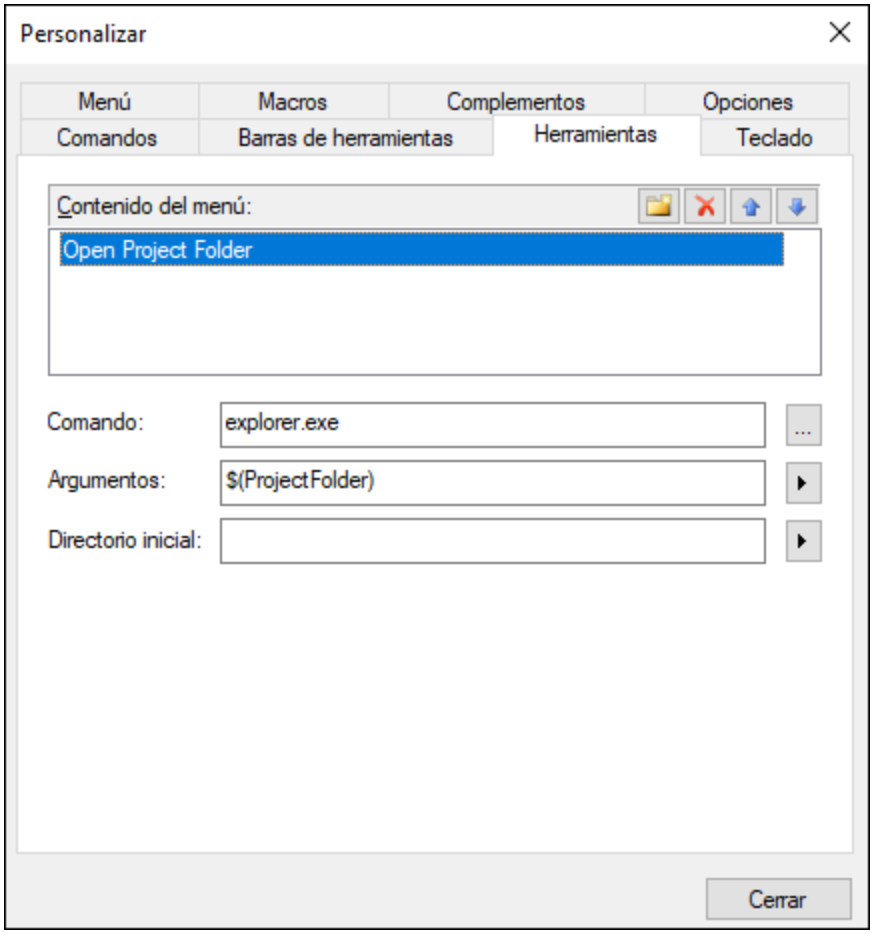
1. Pulse el botón **Restaurar todo**.
2. Todas las barras de herramientas y menús vuelven a su estado original de instalación.

Si marca la casilla *Mostrar etiquetas de texto* parece texto explicativo junto a los iconos de la barra de herramientas.

16.6.6.3 Herramientas

La pestaña *Herramientas* permite crear comandos de menú personalizados que pueden activar herramientas externas directamente desde UModel. Los comandos de menú personalizados que defina en ella aparecen en el menú **Herramientas | Herramientas definidas por el usuario**. Las herramientas externas pueden ser programas que vienen con Windows, como el explorador de Windows (`explorer.exe`), Notepad (`notepad.exe`) u otros archivos ejecutables. Puede asignar argumentos a cada una de las herramientas definidas por el usuario e indicar la carpeta en la que debe inicializarse la herramienta externa (para poder buscar rutas relativas).

Por ejemplo, la configuración de la imagen siguiente añade un nuevo comando llamado "Abrir carpeta de proyecto". Al ejecutarlo, este comando abre el directorio del proyecto actual de UModel en el explorador de Windows.

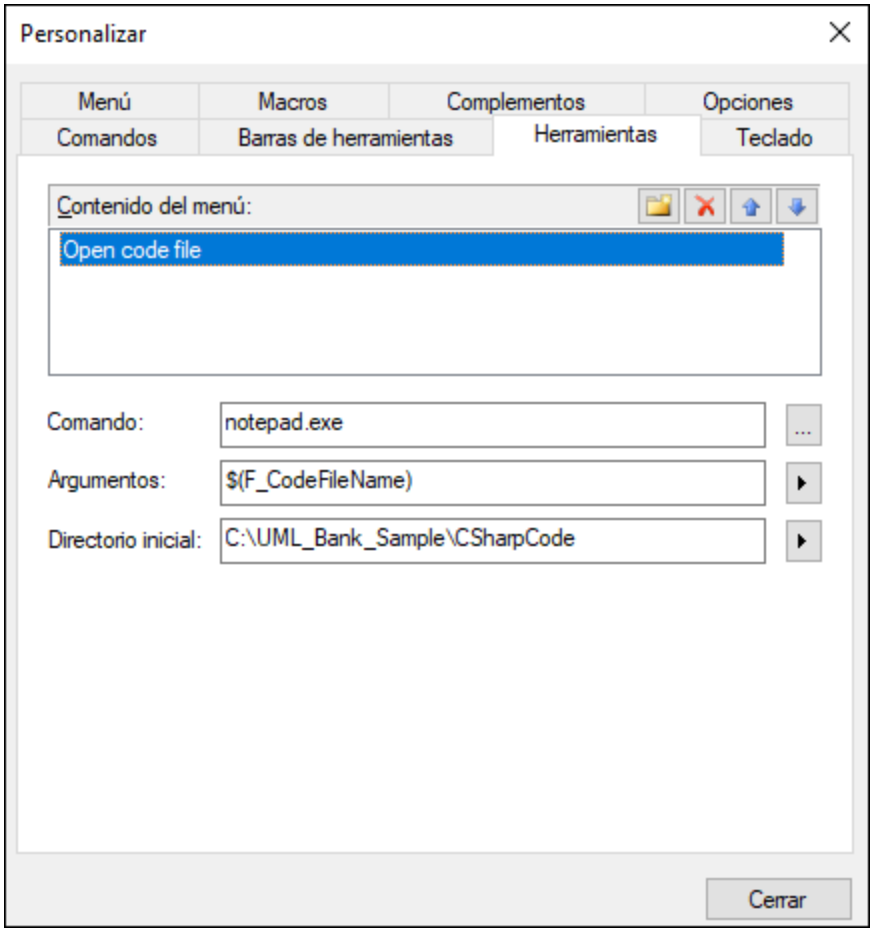


Cuando una herramienta externa toma argumentos (como el explorador de Windows en la imagen anterior), estos se pueden introducir en el campo *Argumentos*. Para usar varios argumentos, sepárelos con espacios. Los valores que introduzca como argumentos pueden ser texto simple (valores incrustados) o se pueden seleccionar con el botón de una lista de variables predefinidas de UModel. Puede usar cualquiera de las siguientes variables predefinidas como argumentos:

Variable predefinida de UModel	Objetivo
Nombre de archivo de proyecto	El nombre del archivo de proyecto de UModel. Por ejemplo **Test.ump**.
Ruta de acceso del archivo de proyecto	La ruta absoluta del archivo de proyecto de UModel. Por ejemplo: **C:\MyDirectory\Test.ump**.
Datos UML resaltados: nombre	El nombre del elemento UML activo en ese momento. Por ejemplo: **Class1**.
Datos UML resaltados: nombre UML completo	El nombre completo del elemento UML activo en ese momento. Por ejemplo: **Package1::Package2::Class1**.

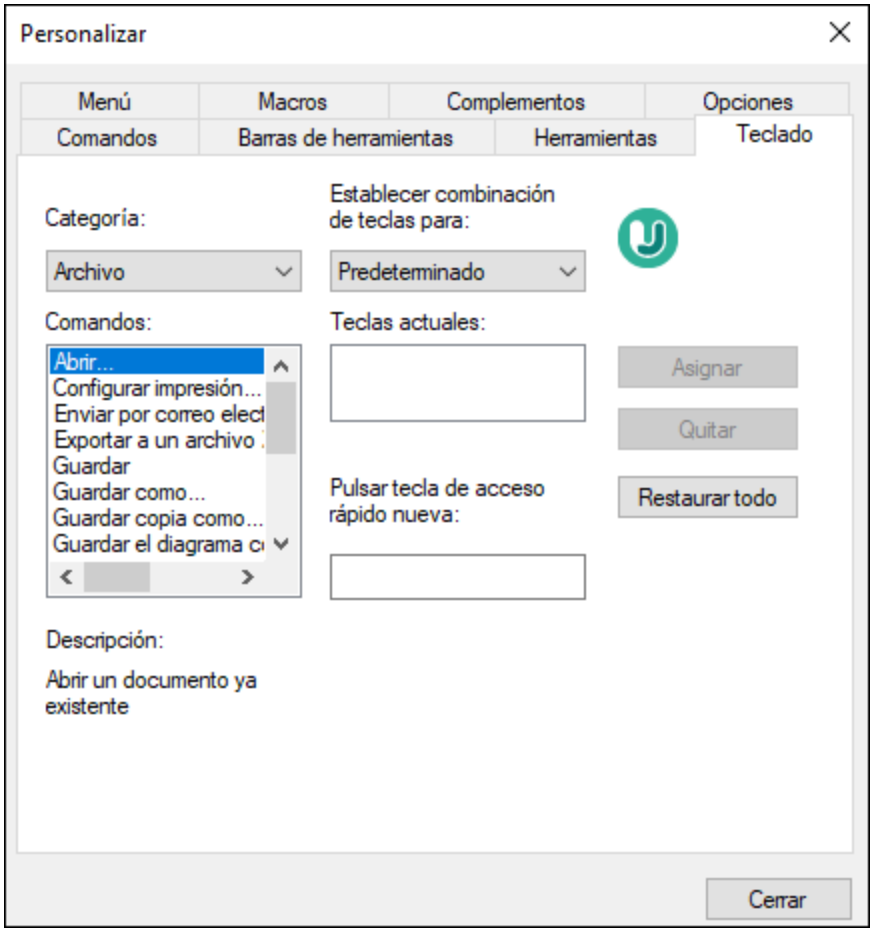
Variable predefinida de UModel	Objetivo
Datos UML resaltados: nombre del archivo de código	El nombre del archivo de código de la clase, interfaz o enumeración UML activa en ese momento, como se muestra en la ventana Propiedades (relativo a los componentes de realización). Por ejemplo: **Class1.cs** or **MyNamespace\Class1.Java**.
Datos UML resaltados: ruta de acceso del archivo de código	La ruta de acceso del archivo de código de la clase, interfaz o enumeración UML activa en ese momento, como se muestra en la ventana Propiedades. Por ejemplo: **C:\Temp\MySource\Class1.cs**.
Datos UML resaltados: nombre del archivo de proyecto de código	El nombre del archivo de proyecto de código al que pertenece la clase, interfaz o enumeración UML activa en ese momento. El nombre del archivo de proyecto de código puede ser relativo al archivo de proyecto de UModel y es el mismo que el que aparece en la ventana Propiedades del componente. Por ejemplo: **C:\Temp\MySource\MyProject.vcproj** or **MySource\MyProject.vcproj**.
Datos UML resaltados: ruta de acceso del archivo de proyecto de código	La ruta de acceso del archivo de proyecto de código al que pertenece la clase, interfaz o enumeración UML activa en ese momento. Por ejemplo: **C:\Temp\MySource\MyProject.vcproj**.
Carpeta de proyecto	La carpeta en la que se guarda el proyecto actual de UModel. Por ejemplo: **C:\Users\<user>\Documents\Altova\UModel2023\UModelExamples**.
Carpeta temporal	La carpeta en la que se guardan los archivos temporales de la aplicación. Por ejemplo: **C:\Users\<user>\AppData\Local\Temp**.

En algunos casos puede que también tenga que introducir un valor en el campo *Directorio inicial*. Por ejemplo, la configuración de la imagen siguiente abre en Notepad el archivo de código del elemento seleccionado en un diagrama. Tenga en cuenta que para que funcione este comando el elemento seleccionado en el diagrama debe tener un valor (nombre de archivo) definido en el campo *nombre del archivo de código* de la ventana Propiedades y la carpeta **C:\UML_Bank_Sample\CSharpCode** debe existir.



16.6.6.4 Teclado

En la pestaña *Teclado* puede crear teclas de acceso rápido nuevas o cambiar las teclas de acceso rápido que ya existen para cualquier comando de la aplicación.



Para asignar una tecla de acceso rápido nueva a un comando o cambiar una tecla de acceso rápido que ya existe:

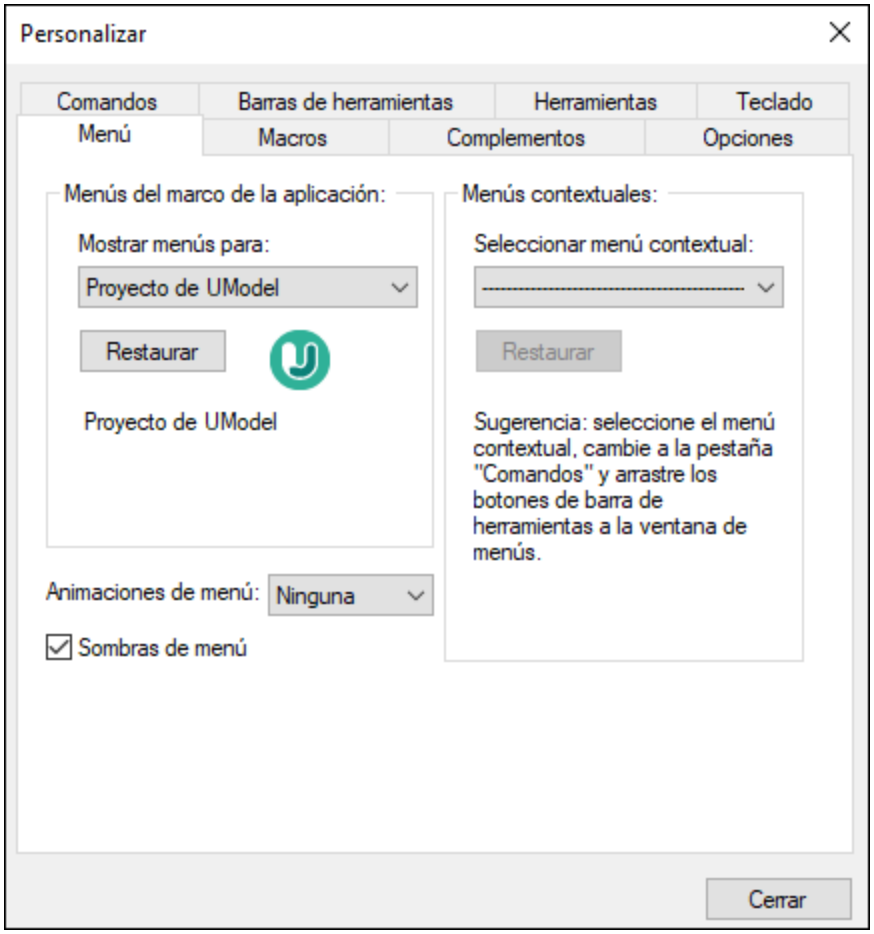
1. En el cuadro combinado *Categoría* seleccione la opción *Todos los comandos*.
2. En el cuadro de lista *Comandos* seleccione el comando al que desea asignar una tecla de acceso rápido nueva o el comando cuya tecla de acceso rápido desea cambiar.
3. Haga clic dentro del cuadro *Pulsar tecla de acceso rápido nueva* y pulse la tecla de acceso rápido que desea asignar al comando. La tecla de acceso rápido aparece en el cuadro *Pulsar tecla de acceso rápido nueva.* Si la tecla de acceso rápido no se asignó todavía a ningún comando, se habilita el botón **Asignar**. Si la tecla ya se asignó a un comando, el comando aparece debajo del cuadro y el botón **Asignar** está deshabilitado. (Para borrar el contenido del cuadro *Pulsar tecla de acceso rápido nueva* pulse **Ctrl**, **Alt** o **Mayús**).
4. Haga clic en **Asignar**. La tecla de acceso rápido aparece ahora en el cuadro de lista *Teclas actuales*. Puede asignar varias teclas de acceso rápido al mismo comando si lo desea.
5. Para confirmar los cambios pulse **Cerrar**.

Para eliminar una tecla de acceso rápido:

1. En el cuadro de lista *Teclas actuales* seleccione la tecla de acceso rápido que desea eliminar.
2. Pulse el botón **Quitar**.
3. Para confirmar los cambios pulse el botón **Cerrar**.

16.6.6.5 Menú

En la pestaña *Menú* puede personalizar las barras de menú principales, así como los menús contextuales de la aplicación.



Personalizar un menú

Puede personalizar las dos barras de menú principales: la *barra de menú predeterminada* y la *barra de menú de UModel*. (La *barra de menú predeterminada* es la barra de menú que aparece cuando no hay ningún tipo de documento XML abierto en la ventana principal. La barra de menú de la aplicación es la barra que aparece cuando hay un documento *.ump abierto en la ventana principal.)

Para personalizar un menú seleccione *Mostrar menús para* de la lista desplegable. Después haga clic en la pestaña *Comandos* y arrastre los comandos hasta la barra de menús que prefiera.

Eliminar comandos de menús y restablecer barras de menú

Para eliminar un menú entero o uno de sus componentes:

1. En la lista desplegable **Mostrar menús para** seleccione la barra de herramientas que quiere personalizar.
2. En cuadro de diálogo "Personalizar" abierto, seleccione (i) el menú que quiere borrar de la barra de herramientas de la aplicación o (ii) el comando que quiere eliminar de uno de estos menús.
3. Ahora puede (i) arrastrar el menú fuera desde la barra de herramientas o el comando fuera del menú, o (ii) hacer clic con el botón derecho en el menú o el comando de menú y seleccionar **Eliminar**.

Puede restaurar cualquier barra de menú a su estado original de instalación; para ello selecciónela en la lista desplegable **Mostrar menús para** y haga clic en el botón **Restaurar**

Personalizar los menús contextuales de la aplicación

Los menús contextuales son aquellos que aparecen si hace clic con el botón derecho en ciertos objetos de la interfaz de la aplicación. Para personalizar esos menús contextuales:

1. Seleccione el menú contextual de la lista desplegable **Seleccionar menú contextual**. Se abre el menú contextual.
2. Haga clic en la pestaña **Comandos**.
3. Arrastre un comando desde la lista Comandos hasta el menú contextual.
4. Para eliminar un comando de un menú contextual, haga clic con el botón derecho en ese comando del menú contextual y seleccione **Eliminar**. Otra opción es arrastrar el comando fuera del menú contextual.

Puede restaurar cualquier menú contextual a su estado original de instalación; para ello selecciónela en la lista desplegable **Seleccionar menú contextual** y haga clic en el botón **Restaurar**.

Sombras de menú

Marque la casilla *Sombras de menú* para que todos los menús tengan sombra.

Puede elegir de entre varias animaciones de menú. La lista desplegable **Animaciones de menú** contiene estas opciones:

- Ninguna (opción predeterminada)
- Desplegar
- Deslizar
- Desvanecer

16.6.6.6 Macros

En la pestaña *Macros* puede elegir las macros definidas en el proyecto de script que está activo en UModel.

El proyecto de script activo aparece en la pestaña *Script* del cuadro de diálogo "Opciones locales" o en la pestaña *Script* del cuadro de diálogo "Configuración del proyecto".

16.6.6.7 Complementos

En la pestaña *Complementos* puede añadir o eliminar un archivo de complemento de UModel (.dll). Para más información consulte el apartado Complementos para entornos IDE.

16.6.6.8 Opciones

En la pestaña *Opciones* puede definir las opciones generales del entorno.

Si marca la opción *Mostrar información en pantalla en las barras de herramientas*, cuando pase el puntero sobre los botones de las barras de herramientas aparecerá una etiqueta con información. La etiqueta incluirá una descripción breve de la función del botón.

Si marca la opción *Mostrar teclas de acceso rápido en la información en pantalla*, la etiqueta de información rápida incluirá la tecla de acceso rápido asociada al botón (siempre y cuando se asignara uno).

Si marca la opción *Iconos grandes*, la interfaz gráfica mostrará los iconos en un tamaño más grande.

16.6.7 Restaurar barras de herramientas y ventanas

El comando **Restaurar barras de herramientas y ventanas** cierra UModel y lo reinicia con su configuración predeterminada. Antes de cerrarse, UModel le pregunta si desea cerrar o no la aplicación.

Este comando es muy práctico si movió ventanas o barras de herramientas de sitio, si las ocultó o si ajustó su tamaño y desea poner todas estas barras de herramientas y ventanas como estaban en un principio.

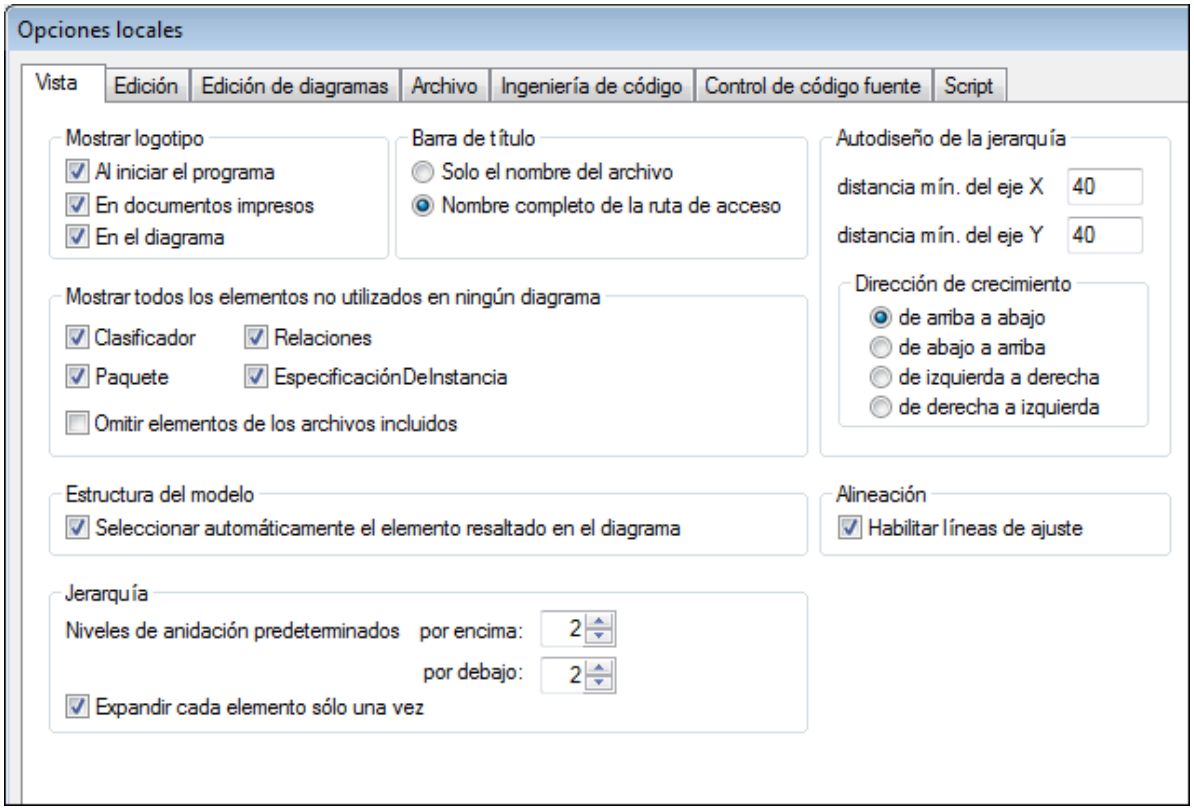
16.6.8 Opciones

El comando **Herramientas | Opciones** abre el cuadro de diálogo "Opciones locales".

En la **pestaña *Vista*** puede definir:

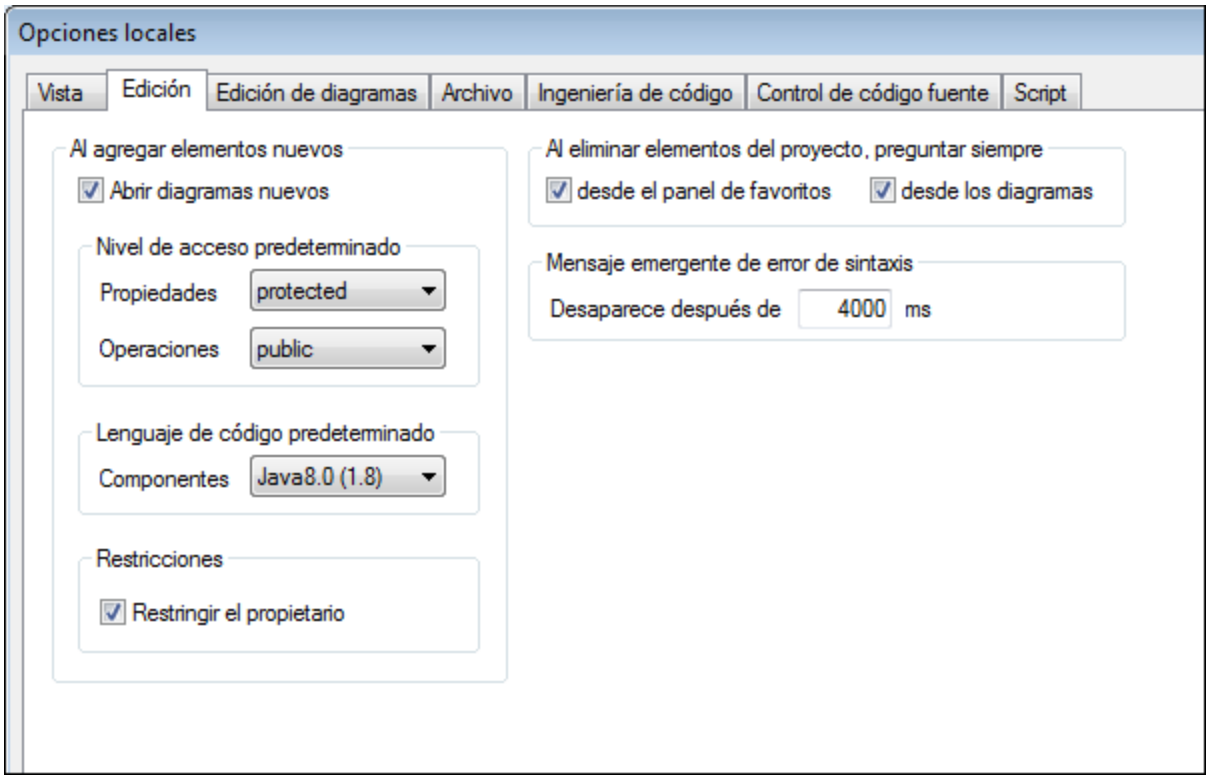
- donde debe aparecer el logotipo del programa.
- el contenido de la barra de título de la aplicación.
- el tipo de elementos que deben aparecer en la lista generada por el comando *Mostrar elementos no utilizados en ningún diagrama* del menú contextual de los paneles *Estructura del modelo* y *Favoritos*. También tiene la opción de omitir los elementos de los archivos incluidos.
- si un elemento seleccionado de un diagrama se selecciona/sincroniza automáticamente o no en la *Estructura del modelo*.
- la profundidad predeterminada de la vista jerárquica generada por el comando **Mostrar en forma de diagrama** en la ventana *Jerarquía*.

- la profundidad de los niveles de la ventana *Jerarquía* (con las opciones del grupo *Autodiseño de la jerarquía*).
- que se expanda solo uno de los clasificadores del mismo tipo de la misma imagen / del mismo diagrama (con la opción *Expandir cada elemento solo una vez*).
- si se habilitan las líneas de ajuste para ayudarle durante la alineación de elementos en el diagrama.



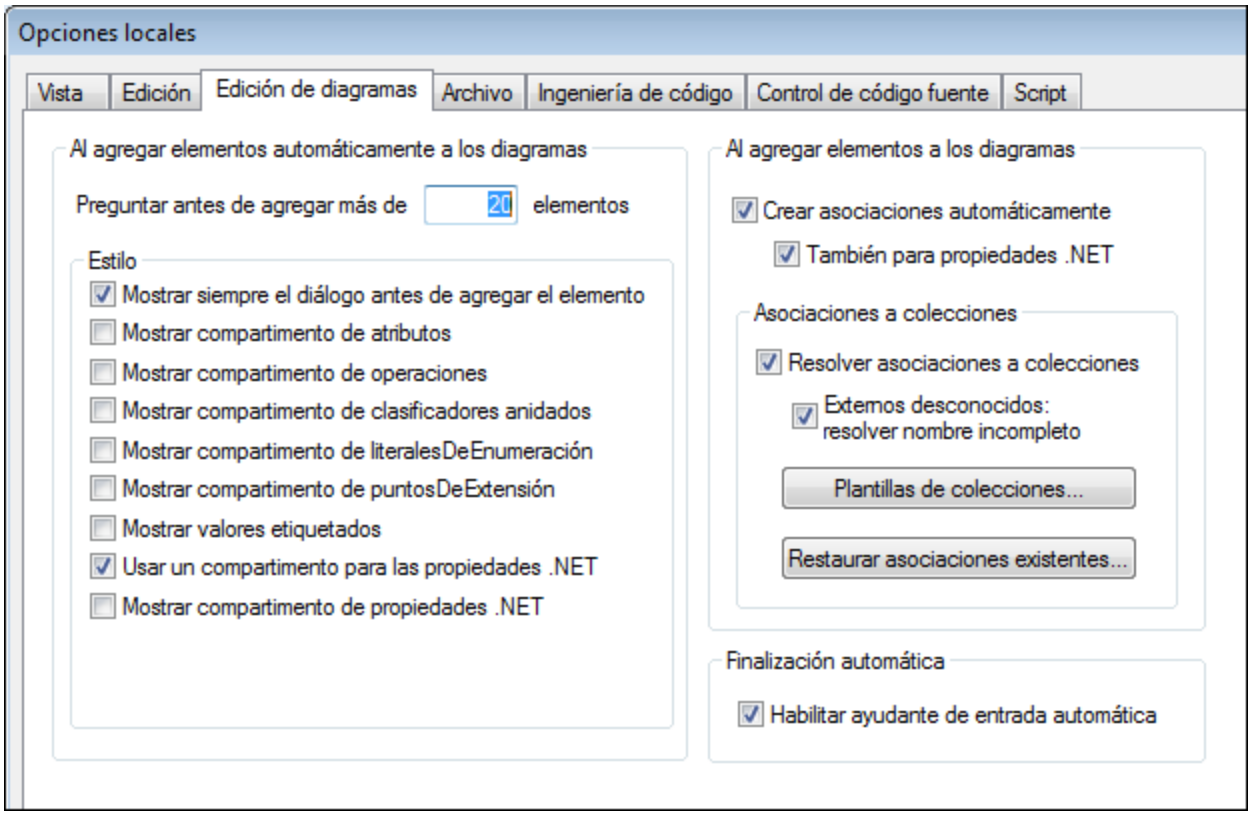
En la **pestaña *Edición*** puede definir:

- que cuando se cree un diagrama nuevo en la *Estructura del modelo*, el diagrama se abra automáticamente en el área de trabajo.
- el nivel de acceso predeterminado de los elementos nuevos (propiedades y operaciones).
- el lenguaje de código predeterminado para los componentes nuevos.
- si las restricciones también deben restringir automáticamente su propietario.
- si debe aparecer un aviso cuando se **eliminen** elementos del proyecto, de la ventana *Favoritos* o de un diagrama. Este aviso se puede desactivar cuando se eliminen elementos.
- cuánto tardan en cerrarse los mensajes emergentes de error de sintaxis.



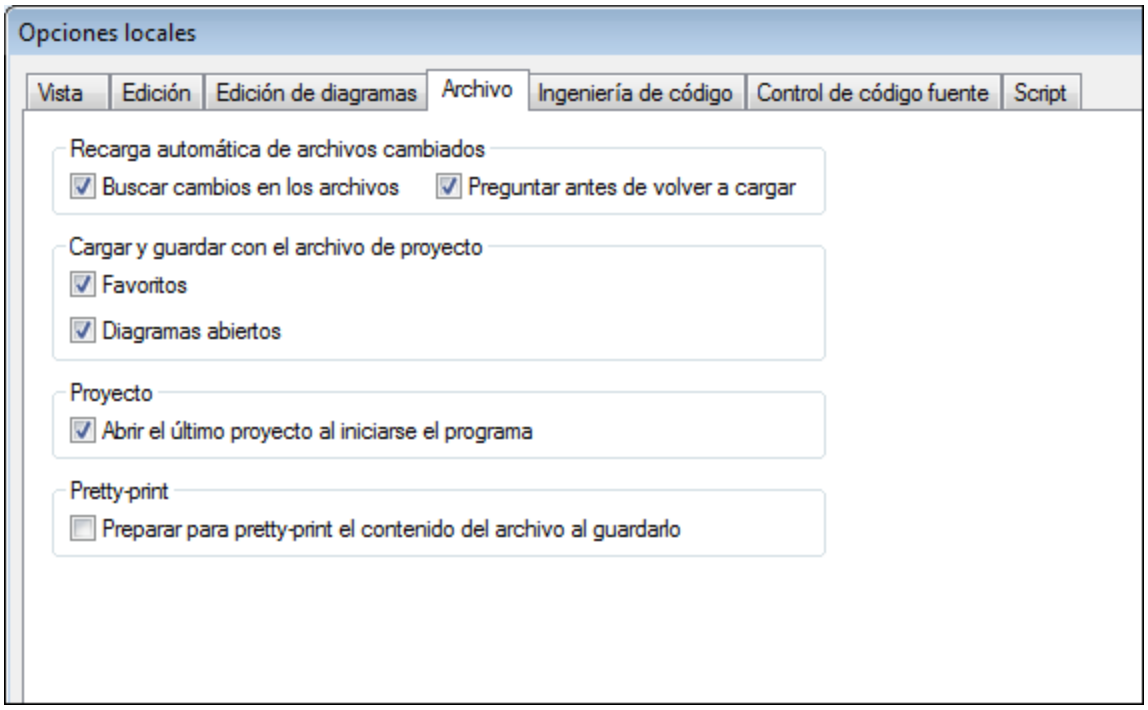
En la **pestaña *Edición de diagramas*** puede definir:

- cuántos elementos se pueden añadir automáticamente a un diagrama sin que aparezca un aviso.
- la presentación de los estilos cuando se añaden automáticamente a un diagrama.
- si se debe crear automáticamente asociaciones entre los elementos de modelado cuando se añaden elementos a un diagrama.
- si se deben resolver las asociaciones a colecciones.
- si las plantillas de elementos externos desconocidos se deben resolver como plantillas no completas.
- o si se deben usar plantillas de colecciones ya disponibles o definir plantillas nuevas.
  Debe definir las plantillas de colecciones como plantillas completas (es decir, a.b.c.Lista). Si la plantilla tiene este espacio de nombres, UModel crea una asociación de colecciones automáticamente. Excepción: si la plantilla pertenece al paquete **`Elementos externos desconocidos`** y está habilitada la opción Externos desconocidos: resolver nombre incompleto, entonces se tiene en cuenta el nombre de la plantilla solamente (es decir, Lista en lugar de a.b.c.Lista).
- si la ventana de finalización automática está disponible mientras se editan atributos u operaciones en los diagramas de clases.



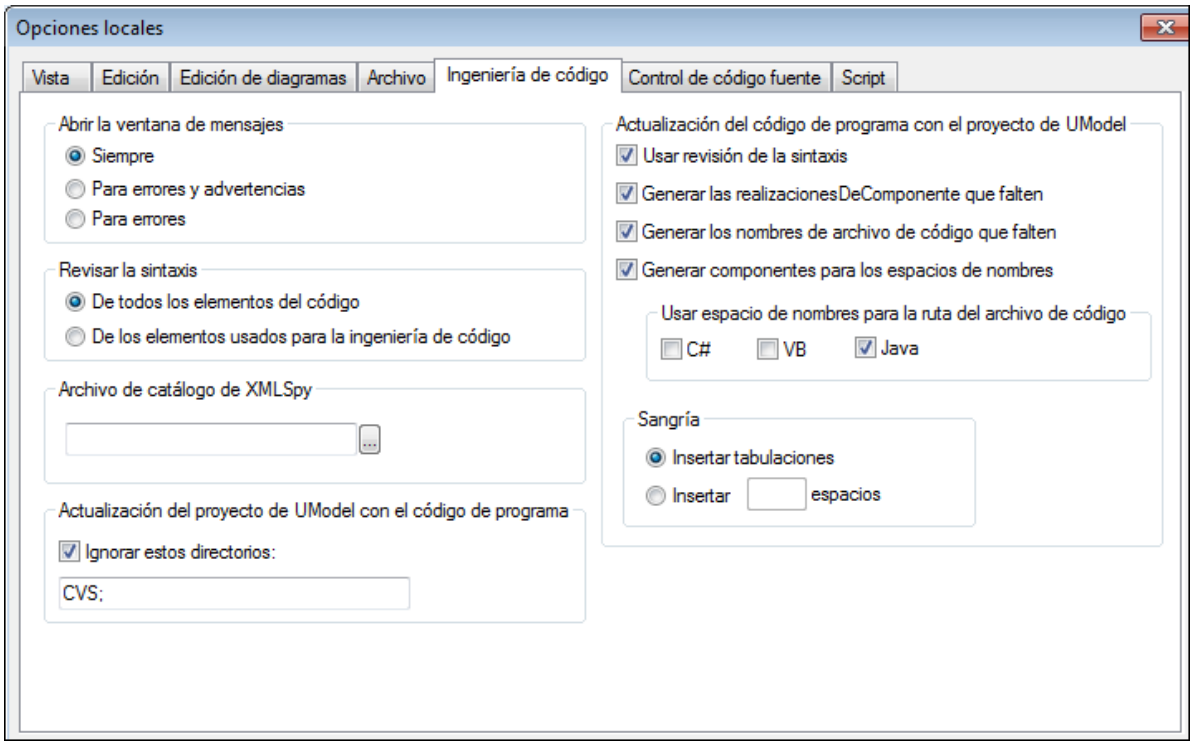
En la **pestaña *Archivo*** puede definir:

- qué ocurre cuando los archivos cambian.
- si el contenido de la ventana *Favoritos* y los diagramas abiertos deben cargarse y guardarse con el proyecto actual.
- si el último proyecto que se abrió se debe abrir automáticamente cuando se inicia la aplicación.
- si se añaden retornos de carro/saltos de línea y tabulaciones al archivo de proyecto para darle formato **pretty-print**.



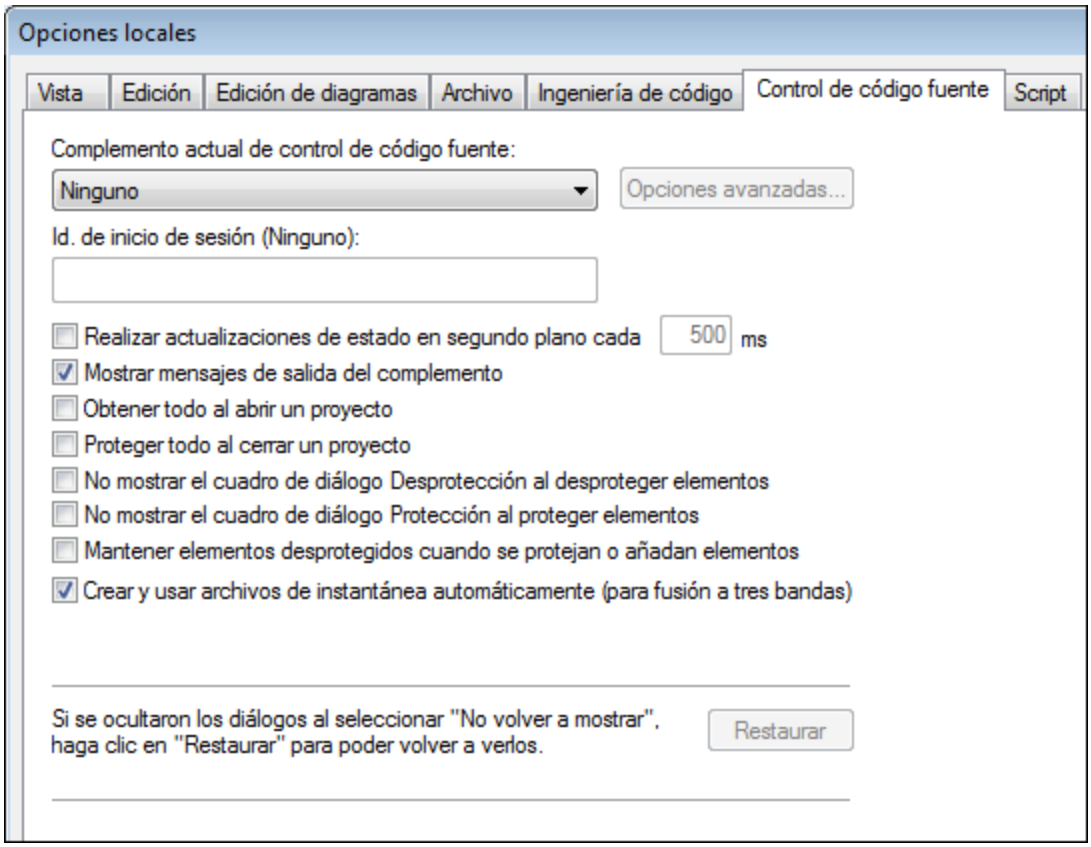
En la **pestaña *Ingeniería de código*** puede definir:

- en qué circunstancias se abre la ventana Mensajes.
- si se revisan **todos los elementos** de código (es decir, los que están en una raíz de espacio de nombres Java / C# / VB y los que están asignados a un componente Java / C# / VB) o **solo los elementos utilizados para ingeniería de código** (es decir, los que tienen activa la casilla Usar para ingeniería de código).
- si durante la actualización del código de programa:
  - se revisa la sintaxis o no.
  - se generan automáticamente las RealizacionesDeComponente que faltan o no.
  - se generan los nombres de archivo de código que faltan en el código combinado o no.
  - se utilizan espacios de nombres en la ruta de acceso del archivo de código o no.
- el tipo de sangría que se aplica al código (es decir, tabulaciones o espacios).
- qué directorios se omiten cuando se actualice un proyecto de UModel con código. Separe los directorios con un punto y coma. Los directorios secundarios que se llamen igual también se pasarán por alto.
- la ubicación del archivo de catálogo de XMLSpy RootCatalog.xml, que permite a UModel y a XMLSpy recuperar esquemas, hojas de estilos y otros archivos utilizados con frecuencia desde carpetas de usuario locales. Esto aumenta la velocidad de procesamiento y permite al usuario trabajar sin conexión.
- también puede indicar si quiere hacer una copia de seguridad de los archivos C++ modificados.



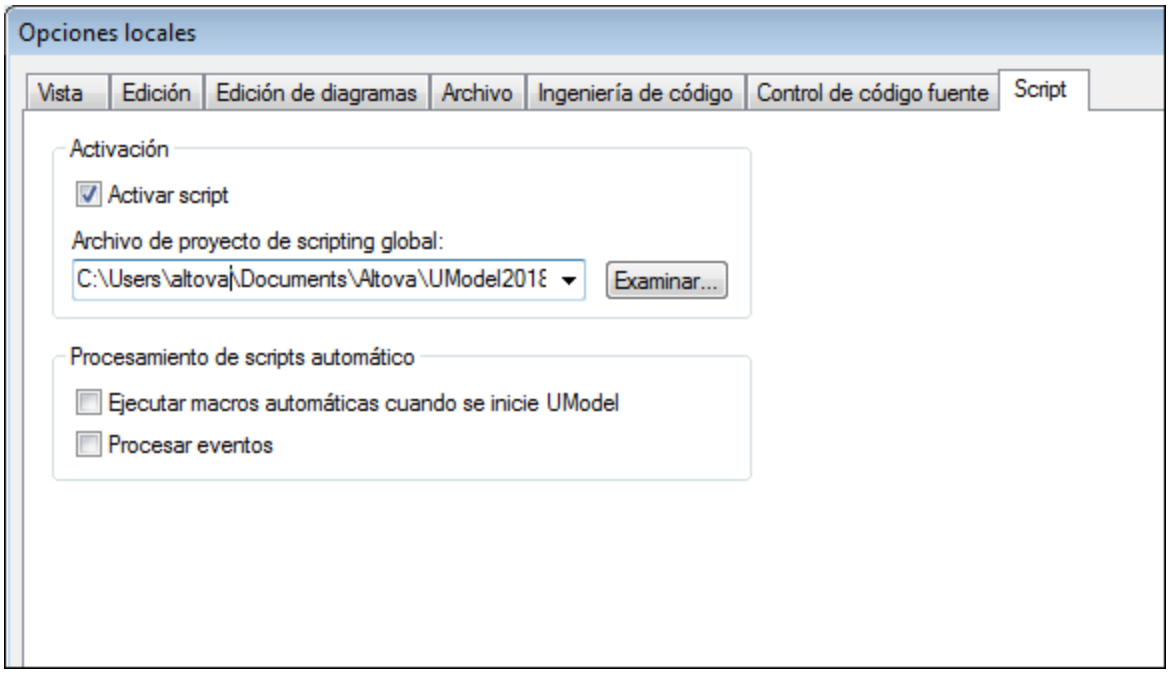
En la **pestaña *Control de código fuente*** puede definir:

- el complemento de control de código fuente actual. Con el botón **Opciones avanzadas** puede configurar algunas opciones avanzadas del sistema de control de código seleccionado. Estas opciones avanzadas dependen del control de código fuente elegido.
- el id. de inicio de sesión para el proveedor de control de código fuente.
- otras opciones relacionadas con la protección y desprotección de archivos.
- el botón **Restaurar** se habilita si el usuario marca la casilla *No volver a mostrar* en un cuadro de diálogo. Con el botón **Restaurar** puede volver a habilitar los avisos.



En la **pestaña *Script*** puede definir:

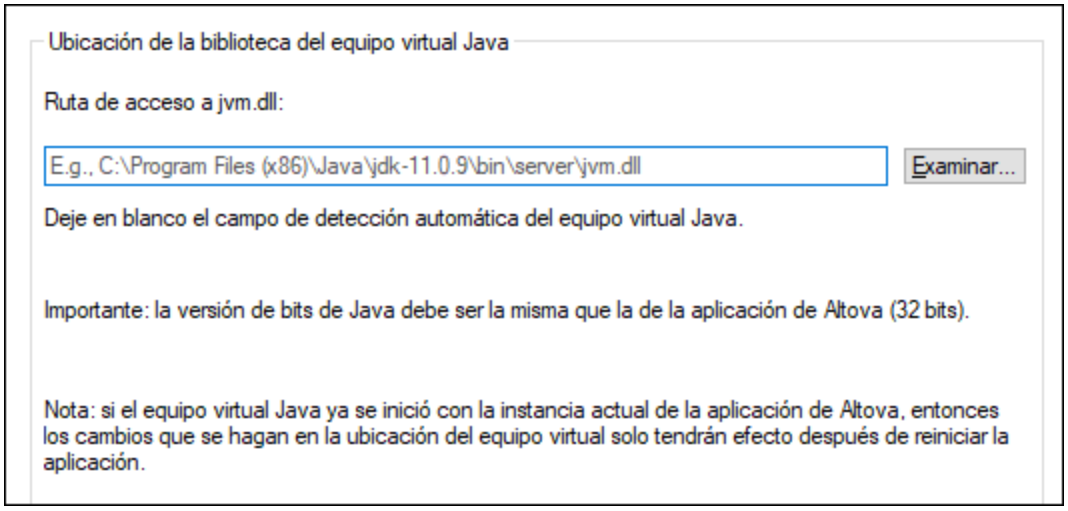
- si el entorno de scripting debe activarse para el proyecto actual.
- qué archivo de scripting global se utiliza.
- si se deben ejecutar las macros automáticas cuando se inicia UModel.
- si se deben procesar los eventos de scripting.



16.6.8.1 Configuración de equipos virtuales Java

En la pestaña *Java* puede introducir la ruta de acceso a un equipo virtual java en su sistema de archivos. Tenga en cuenta que no siempre es necesario agregar una ruta de acceso personal a un equipo virtual. Por defecto, UModel intenta detectar esta ruta automáticamente leyendo (en este orden) el registro de Windows y la variable de entorno JAVA_HOME. Si se detecta automáticamente cualquier otra ruta de equipo virtual java, tendrá prioridad la ruta personal que se indica en este cuadro de diálogo.

Puede que necesite añadir esta ruta personal de acceso a un equipo virtual java si está usando un equipo virtual java que no tiene instalador ni crea entradas de registro (por ejemplo, OpenJDK, de Oracle). También puede querer usar esta ruta para suprimir, por la razón que fuere, cualquier otra ruta que UModel haya detectado automáticamente.



Observe lo siguiente:

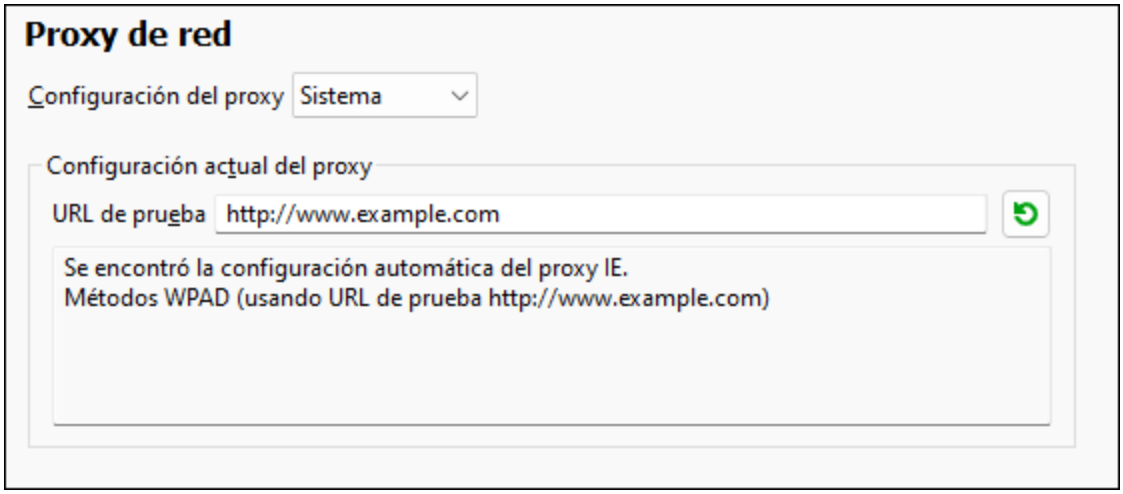
- la ruta de acceso al equipo virtual java es común a todas las aplicaciones de escritorio de Altova (no a las de servidor). En consecuencia, si cambia esta ruta en una de ellas, el cambio afectará automáticamente al resto de aplicaciones de Altova.
- la ruta debe apuntar al archivo jvm.dll desde los directorios **\bin\server** o **\bin\client**, relativos al directorio en el que está instalado el JDK.
- la plataforma de UModel (versión de 31 o de 64 bits) debe ser la misma que la del JDK.
- después de cambiar la ruta de acceso al escritorio virtual java debe reiniciar UModel para que surta efecto la nueva configuración.

Cambiar la ruta de acceso al escritorio virtual java afecta a la conectividad BD con JDBC.Esta opción no afecta a la generación ni a la importación de código Java. Tenga en cuenta que los tiempos de ejecución Java usados para importar binarios Java en UModel se pueden configurar por separado (*consulte Añadir tiempos de ejecución Java personalizados*).

16.6.8.2 Configuración del proxy de red

El cuadro de diálogo **Proxy de red** (*imagen siguiente*) permite personalizar la configuración del proxy de red. El sistema viene con una configuración predeterminada para el proxy, por lo que este funcionará sin necesidad de configurarlo, pero si quiere usar un proxy de red alternativo puede usar estas opciones para cambiar la configuración como quiera.

Nota: la configuración del proxy de red es común a todas las aplicaciones de Altova MissionKit. En consecuencia, si cambia esta configuración en cualquiera de esas aplicaciones, el cambio afectará automáticamente a todas las demás.



Usar la configuración del proxy del sistema
Usa los parámetros de Internet Explorer (IE), que se pueden configurar desde las opciones del proxy de red. También consulta los parámetros configurados con **`netsh.exe winhttp`**.

Configuración automática del proxy
Existen las siguientes opciones:

- *Configuración de detección automática:* consulta un script WPAD (**`http://wpad.LOCALDOMAIN/wpad.dat`**) vía DHCP o DNS y lo usa para configurar el proxy.
- *URL del script*: indica una HTTP URL a un script (`.pac`) de configuración automática del proxy cuyos parámetros se aplican para configurar el proxy.
- *Volver a cargar*: reinicia y vuelve a cargar la configuración automática actual del proxy. Esta acción requiere Windows 8 o superior y puede llegar a tardar 30 segundos en tener efecto.

Configuración manual del proxy
Puede indicar manualmente el nombre completo de host y el puerto para los proxys de los respectivos protocolos. Es posible que haya un esquema compatible incluido en el nombre de host (por ejemplo: **`http://hostname`**). Si el proxy es compatible no es necesario que el esquema sea el mismo que el protocolo correspondiente.

Existen las siguientes opciones:

- *Usar este servidor de proxy para todos los protocolos:* usa el nombre de host y el puerto del Proxy HTTP para todos los protocolos.
- *Ningún proxy para:* muestra una lista de elementos separados por punto y coma (`;`) que pueden ser nombres de host, nombres de dominios o direcciones IP para hosts para los que no hay que usar proxy. Las direcciones IP no se pueden truncar y las direcciones IPv6 deben colocarse entre corchetes (por ejemplo: `[2606:2800:220:1:248:1893:25c8:1946]`). Los nombres de dominio deben empezar por punto (por ejemplo: `.example.com`).
- *No use el servidor proxy para direcciones locales*: si se marca esta opción, se añade el elemento `<local>` a la lista *Ningún proxy para*. Si se selecciona esta opción no se usará proxy para: (i) `127.0.0.1`, (ii) [::1], (iii) todos los nombres de host que no contengan punto (`.`).

16.7 Menú Ventanas

En cascada

Este comando reorganiza todos las ventanas de documento que están abiertas en forma de **cascada** (es decir, las ventanas se apilan una encima de otra).

En mosaico horizontal

Este comando reorganiza todas las ventanas de documento que están abiertas en forma de **mosaico horizontal** (es decir, se pueden ver todas las ventanas a la vez y se distribuyen de forma horizontal).

En mosaico vertical

Este comando reorganiza todas las ventanas de documento que están abiertas en forma de **mosaico vertical** (es decir, se pueden ver todas las ventanas a la vez y se distribuyen de forma vertical).

Ordenar iconos

Este comando reorganiza los elementos que estén dispersos por el diagrama y los coloca en la parte inferior del área de trabajo.

Cerrar

Este comando cierra la pestaña del diagrama activo.

Cerrar todas

Este comando cierra todas las pestañas de diagrama que están abiertas.

Cerrar ventanas inactivas

Este comando cierra todas las pestañas de diagrama que están abiertas excepto la pestaña activa.

Adelante

Este comando abre la siguiente pestaña de diagrama o el siguiente elemento con hipervínculo.

Atrás

Este comando abre la pestaña de diagrama anterior o el elemento con hipervínculo anterior.

Lista de ventanas abiertas (1, 2)

Esta lista muestra todas las ventanas que están abiertas en cada momento y permite cambiar de una ventana a otra rápidamente.
También puede usar las teclas de acceso rápido **Ctrl+Tabulador** o **Ctrl+F6** para recorrer todas las ventanas que están abiertas.

Ventanas

Muestra un cuadro de diálogo en el que puede exponer o cerrar varias ventanas de diagrama simultáneamente. Consulte también el apartado panel *Diagramas*.

16.8 Menú Ayuda

Contenido

Abre la ayuda en pantalla de UModel por la tabla de contenido, que aparece en el panel izquierdo de la ventana de ayuda. Esta tabla de contenido ofrece un resumen de la documentación. Haga clic en una entrada de la tabla para abrir la sección correspondiente de la documentación.

Índice

Abre la ayuda en pantalla de UModel por el índice de palabras clave, que aparece en el panel izquierdo de la ventana de ayuda. Este índice enumera todas las palabras clave de la ayuda y permite navegar a un tema con solo hacer doble clic en la palabra clave correspondiente. Si una palabra clave está asociada a varios temas, la ventana de ayuda muestra todos estos temas en pantalla.

Buscar

Abre la ayuda en pantalla de UModel por la función de búsqueda, que aparece en el panel izquierdo de la ventana de ayuda. Para buscar un término introduzca el término de búsqueda en el campo de consulta y pulse **Entrar**. El sistema de ayuda realiza una búsqueda en toda la documentación y devuelve una lista de resultados. Haga doble clic en una entrada para abrir la sección correspondiente de la documentación.

Activación del software

Tras descargar el producto de software de Altova puede registrarlo o activarlo con una clave de evaluación gratuita o con una clave de licencia permanente.

- ***Licencia gratuita:*** cuando inicie el software por primera vez aparecerá el cuadro de diálogo "Activación del software". Este cuadro de diálogo incluye un botón para solicitar una licencia de evaluación gratuita. Introduzca su nombre, el nombre de su compañía y su dirección de correo electrónico y haga clic en **Enviar solicitud**. Altova le enviará un archivo de licencia al correo electrónico proporcionado. Guarde el archivo en su equipo. Al hacer clic en **Enviar solicitud** aparece un campo en la parte inferior del cuadro de diálogo, que es donde debe introducir la ruta de acceso al archivo de licencia. Puede escribir la ruta de acceso o navegar hasta el archivo de licencia. Por último, haga clic en **Aceptar**. (En el cuadro de diálogo de activación del software también puede hacer clic en **Cargar licencia nueva** para acceder al cuadro de diálogo en el que se introduce la ruta de acceso.) El software permanecerá desbloqueado durante 30 días.
- ***Clave de licencia permanente:*** el cuadro de diálogo "Activación del software" también incluye un botón para comprar una clave de licencia permanente. Este botón conduce a la tienda en línea de Altova, donde podrá adquirir una clave de licencia permanente para el producto. Recibirá por correo electrónico un archivo que contiene los datos de la licencia. Existen tres tipos de licencias permanentes: de tipo instalado, de usuario concurrente y de usuario designado. Las *licencias de tipo instalado* son cada una para un único equipo. En el caso de las *licencias de usuario concurrentes*, si adquiere una de estas licencias para `N` usuarios podrá instalar la licencia en un máximo de `10N` equipos pero solo `N` usuarios podrán usar el software al mismo tiempo. Las *licencias de usuario designado* autorizan a un usuario específico a usar el software en un máximo de 5 equipos distintos. Para activar su software haga clic en **Cargar una licencia nueva** e introduzca la ruta de acceso al archivo de licencia en el cuadro de diálogo "Activación del software" que aparece. Por último, haga clic en **Aceptar**.

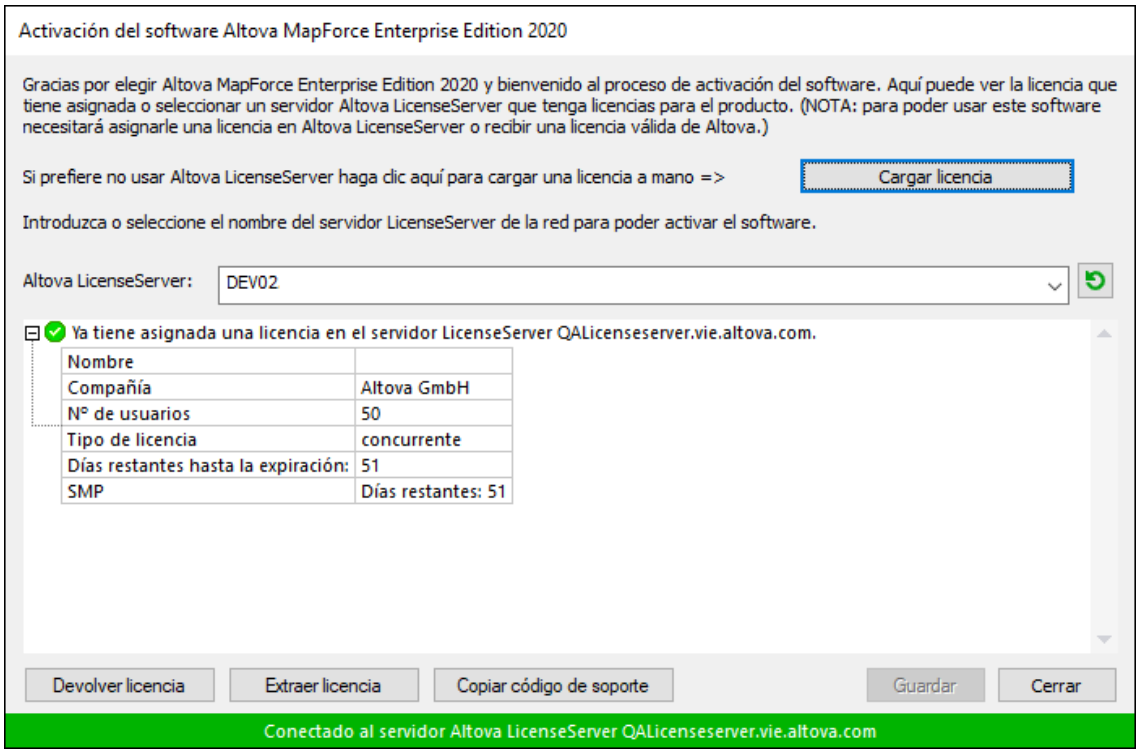
Nota: en el caso de licencias para varios usuarios, se le pedirá a cada usuario que introduzca su nombre.

> *Claves por correo electrónico y las distintas formas de activar las licencias de los productos de Altova*
> El correo electrónico que recibirá de Altova contiene, en un adjunto, el archivo de la licencia, que tiene la extensión **.altova_licenses**.
>
> Para activar su producto de Altova, puede optar por una de las siguientes opciones:
>
> - Guardar el archivo de licencia (**.altova_licenses**) en su equipo, hacer doble clic en el archivo de licencia, introducir los detalles necesarios en el cuadro de diálogo que aparece y finalmente hacer clic en **Aplicar claves**.
> - Guardar el archivo de licencia (**.altova_licenses**) en su equipo. En su producto de Altova seleccione el comando de menú **Ayuda | Activación del software** y después **Cargar una licencia nueva**. Navegue hasta el archivo de licencia o introduzca la ruta de acceso al archivo de licencia en el cuadro de diálogo "Activación del software" y haga clic en **Aceptar**.
> - Guardar el archivo de licencia (**.altova_licenses**) en su equipo y cargarlo desde esa ubicación a su Altova LicenseServer. Puede: (i) adquirir la licencia de su producto Altova con el cuadro de diálogo de activación de software del producto (*véase más abajo*) o (ii) asignar la licencia al producto de Altova LicenseServer. Para obtener más información sobre la gestión de licencias con el LicenseServer, lea el resto de esta sección.

El cuadro de diálogo "Activación del software" (*imagen siguiente*) se abre con el comando **Ayuda | Activación del software**.

Hay dos maneras de activar el software:

- registrando la licencia en el cuadro de diálogo "Activación del software". Para ello haga clic en **Cargar una licencia nueva** y navegue hasta el archivo de la licencia. Haga clic en **Aceptar** para confirmar la ruta de acceso al archivo de licencia y para confirmar los datos que haya introducido (su nombre, en el caso de licencias para más de un usuario); a continuación, haga clic en **Guardar** para finalizar el proceso.
- asignando una licencia a través de un servidor Altova LicenseServer de la red: para adquirir una licencia a través de un servidor Altova LicenseServer de la red haga clic en el botón **Usar Altova LicenseServer**, situado al final del cuadro de diálogo. Seleccione el equipo en el que está instalado el LicenseServer que quiere usar. Tenga en cuenta que la autodetección de los License Servers funciona con emisiones enviadas por LAN. Este tipo de emisiones se limitan a una subred, por lo que Altova License Server debe estar en la misma subred que el equipo del cliente para que funcione la autodetección. Si esta no funciona, introduzca el nombre del servidor. Para ello es necesario que el servidor LicenseServer tenga una licencia para su producto en el repositorio de licencias. Si así es, el cuadro de diálogo "Activación del software" emite un mensaje a tal efecto (*imagen siguiente*). Haga clic en el botón **Guardar** para adquirir la licencia.



Una vez se ha adquirido una licencia para un equipo específico (es decir, "instalada") del servidor LicenseServer, no se puede devolver al mismo hasta 7 días después. Transcurridos estos 7 días podrá devolver la licencia de ese equipo (con el botón **Devolver licencia**) para que pueda ser adquirida por otro cliente. No obstante, el administrador de LicenseServer puede anular asignaciones de licencias desde la interfaz web del servidor LicenseServer en cualquier momento. Observe que únicamente se pueden devolver las licencias instaladas en equipos específicos, no las licencias concurrentes.

Extracción de licencias

Puede extraer una licencia del repertorio durante un período máximo de 30 días de modo que la licencia se almacene en el equipo donde se ejecuta el producto. Esto le permitirá trabajar sin conexión a Internet, lo cual puede ser útil si desea trabajar en un entorno que no dispone de acceso a su servidor Altova LicenseServer (p. ej. cuando el producto servidor de Altova está instalado en un equipo portátil y el usuario se encuentra de viaje). Mientras la licencia esté extraída, LicenseServer indicará que la licencia está en uso y no podrá ser utilizada por ningún otro equipo. La licencia vuelve automáticamente a su estado insertado cuando finaliza el período de extracción de la licencia. La licencia extraída también se puede insertar en el servidor en cualquier momento con el botón **Insertar** del cuadro de diálogo "Activación del software".

Siga estas instrucciones para extraer una licencia:

1. En el cuadro de diálogo "Activación del software" haga clic en el botón **Extraer licencia** (*imagen anterior*).
2. Aparece el cuadro de diálogo "Extracción de licencias". Seleccione el periodo de extracción y haga clic en **Extraer**.
3. La licencia se extrae y ocurren dos cosas: (i) el cuadro de diálogo "Activación del software" muestra información sobre la extracción de la licencia, incluida la fecha y la

hora en la que expira el plazo de extracción y (ii) en lugar del botón **Extraer licencia**, aparece el botón **Insertar licencia**. Para insertar la licencia basta con hacer clic en este botón. Como la licencia vuelve automáticamente a su estado de inserción cuando finaliza el plazo de extracción, compruebe que el plazo seleccionado coincide con el período de tiempo que tiene pensado trabajar sin conexión a Internet.

Para devolver una licencia esta debe ser de la misma versión principal que el producto de Altova con el que se extrajo. Por tanto, es recomendable devolver la licencia antes de actualizar el producto de Altova correspondiente a la siguiente versión principal.

Nota: para poder extraer licencias esta característica debe estar habilitada en el servidor LicenseServer. Si esta característica no está habilitada, recibirá un mensaje de error a tal efecto cuando trate de extraer una licencia. Cuando esto ocurra, póngase en contacto con el administrador de su servidor LicenseServer.

Copiar código de soporte
Haga clic en **Copiar código de soporte** para copiar los detalles de la licencia en el portapapeles. Esta es la información que deberá introducir al ponerse en contacto con el equipo de soporte técnico a través del formulario de soporte técnico.

Altova LicenseServer es una práctica herramienta para administrar en tiempo real todas las licencias de Altova de la red y ofrece información detallada sobre cada licencia, asignaciones a clientes y uso de las licencias. La ventaja de usar este producto está en sus características administrativas. Altova LicenseServer puede descargarse gratis del sitio web de Altova. Para más información consulte la documentación de Altova LicenseServer.

Formulario de pedido

Hay dos maneras de comprar licencias para los productos de Altova: con el botón **Comprar una licencia permanente** del cuadro de diálogo "Activación del software" (*ver apartado anterior*) o con el comando **Ayuda | Formulario de pedido**, que le lleva directamente a la tienda en línea de Altova, donde puede adquirir claves de licencias para los productos de Altova.

Registro del software

Este comando abre la página de registro de productos de Altova en una pestaña del explorador. Si registra el software, recibirá información sobre actualizaciones y versiones nuevas del producto.

Buscar actualizaciones

Comprueba si existe una versión más reciente del producto en el servidor de Altova y emite un mensaje a tal efecto.

Centro de soporte técnico

Es un enlace al centro de soporte técnico del sitio web de Altova. El centro de soporte técnico incluye preguntas frecuentes, foros de debate y un formulario para ponerse en contacto con el equipo de soporte técnico de Altova.

Preguntas más frecuentes

Es un enlace a la página de preguntas frecuentes del sitio web de Altova. Esta página se actualiza constantemente con las preguntas que recibimos de nuestros clientes.

- Descargar herramientas gratis y componentes

  Es un enlace al centro de descargas de componentes del sitio web de Altova. Aquí puede descargar software adicional para usarlo con los productos de Altova, como procesadores XSLT y XSL-FO y paquetes de integración. Estos componentes suelen ser totalmente gratis.

- UModel en Internet

  Es un enlace al sitio web de Altova, donde encontrará más información sobre UModel, otros productos de Altova y tecnologías relacionadas.

- Acerca de UModel

  Abre la pantalla de presentación de la aplicación, que incluye el número de versión del producto e información sobre copyright. Si usa la versión de 64 bits de la aplicación, esto se ve en el nombre de la aplicación, que lleva el sufijo (x64). La versión de 32 bits no lleva ningún sufijo.

17 Referencia del programador

UModel es un servidor de automatización. Esto significa que es una aplicación que expone objetos programables a otras aplicaciones (llamadas clientes de automatización). Como resultado, el cliente de automatización tiene acceso directo a los objetos y a las funciones que el servidor de automatización pone a su disposición. Esto es una ventaja para el cliente de automatización porque puede usar todas las funciones de UModel, como la función de ingeniería inversa. Por tanto, los programadores pueden mejorar sus propias aplicaciones usando las funciones de UModel.

Los objetos programables de UModel se ponen a disposición de los clientes de automatización mediante la API de UModel, que es una API de COM. El modelo de objetos de la API se describe en la referencia de la API de UModel, que también describe todos los objetos disponibles.

Puede acceder a la API de la aplicación desde estos entornos:

- Desde el editor de scripts
- Desde complementos creados en entornos IDE
- Desde programas externos

A continuación ofrecemos una breve descripción de estos tres entornos.

Editor de scripts

Si quiere puede personalizar su versión de UModel modificando y añadiéndole funciones. También puede crear formularios y modificar la interfaz del usuario añadiéndole comandos de menú nuevos e iconos nuevos en las barras de herramientas. Para ello basta con escribir scripts que interactúen con objetos de la API de la aplicación. Para ayudarle a escribir estos scripts, UModel le ofrece un editor de scripts integrado. Las funciones de este editor se describen detalladamente en el apartado Editor de scripts de este manual. Los lenguajes de programación compatibles son **JScript** y **VBScript**.

Complementos creados en entornos IDE

Con UModel puede crear complementos propios como archivos DLL e integrarlos en UModel. La interfaz gráfica del usuario de UModel contiene comandos que permiten habilitar o deshabilitar estos complementos. Algunos de los lenguajes que se suelen usar para implementar complementos IDE son **C#** y **C++**. Para más información consulte Complementos para entornos IDE.

Programas externos

También puede manipular UModel usando scripts externos. Por ejemplo, puede escribir un script para abrir UModel en un momento determinado, después abrir un proyecto de UModel, generar su documentación e imprimirla. Al igual que los entornos anteriores, los scripts externos utilizan la API de la aplicación para realizar estas tareas. Para más información consulte el apartado API de UModel.

Para poder usar la API de UModel desde programas externos es necesario iniciar una instancia de UModel primero. Consulte Accessing the API.

Básicamente UModel se iniciará a través de su registro de COM. Después se devuelve el objeto Application asociado con la instancia de UModel. Dependiendo de la configuración de COM, se puede devolver un objeto asociado con una instancia que ya está en ejecución. Además se puede usar cualquier lenguaje de programación que permita crear e invocar objetos COM. Los más frecuentes son:

- Los archivos JScript y VBScript tienen una sintaxis sencilla y están diseñados para acceder a objetos COM. Se pueden ejecutar desde la línea de comandos directamente o haciendo doble clic en ellos desde el explorador de Windows. Son muy útiles para tareas de automatización sencillas.
- C# es un lenguaje de programación de pleno derecho con una amplísima gama de funciones. El acceso a objetos COM se puede encapsular automáticamente usando C#..
- Java: los productos de Altova vienen con clases Java nativas que encapsulan la API de la aplicación y ofrecen un aspecto Java total
- También puede usar Visual Basic for Applications, Perl y Python.

17.1 Notas sobre la versión

A continuación se enumeran todos los cambios importantes incorporados en cada versión de la API de UModel. Si en la versión principal de la biblioteca de tipos se produce un cambio (p. ej. 1.0 => 2.0 => 3.0), significa que no se debería volver a compilar clientes como los complementos creados en C#, VB.NET, C++, etc. desde entornos IDE.

Interfaz de automatización para UModel 2021 (biblioteca de tipos versión 5.8)

API de UModel	• La enumeración `ENUMCodeLangVersion` tiene miembros nuevos para la versión 14.0 del lenguaje Java y para la base de datos MariaDB. • La enumeración `ENUMUMLPredefinedElement` tiene varios miembros nuevos, que incluyen los miembros necesarios para las versiones más recientes de SysML.

Interfaz de automatización para UModel 2020r2 (biblioteca de tipos versión 5.7)

API de UModel	• La enumeración `ENUMCodeLangVersion` tiene un nuevo miembro que corresponde a la versión de lenguaje "Java 13.0". • La enumeración `ENUMExportXMIType` tiene nuevos miembros: `eXMI24ForUML25` y `eXMI251ForUML251`. Ambos admiten la exportación XMI a las versiones de XMI correspondientes (consulte también XMI: intercambio de metadatos XML). • La interfaz `IApplication` tiene operaciones nuevas: `LogMessage`, `LogMessageWithUMLDataLink`. También hay disponible una enumeración `ENUMMessageLogType` nueva que permite mostrar mensajes de error (que provengan, por ejemplo, de un complemento IDE para UModel) en la ventana Mensajes de UModel.
API de UModel: UMLData	• Hay disponible una interfaz `IUMLReception` nueva, así como varias propiedades y varios métodos compatibles con UML Receptions. La enumeración `ENUMUMLGuiStyleKind` tiene un nuevo miembro `eUMLGuiStyle_ShowReceptions`. • Ahora puede añadir **ValuePin** a la acción **CallBehaviorAction** con la operación nueva `InsertArgumentOfKindAt`.

Interfaz de automatización para UModel 2020 (biblioteca de tipos versión 5.6)

API de UModel	• La enumeración `ENUMCodeLangVersion` tiene nuevos miembros que corresponden a los lenguajes C# 8.0 y C++17.

Interfaz de automatización para UModel 2019r3 (biblioteca de tipos versión 5.5)

API de UModel	• La propiedad `ImageFormat` se ha eliminado de la interfaz `ISaveAllDiagramsAsImagesDlg`. • La enumeración `ENUMCodeLangVersion` tiene un nuevo miembro que corresponde a la versión de lenguaje "Java 12".

Interfaz de automatización para UModel 2018r2 (biblioteca de tipos versión 5.4)

API de UModel	• La interfaz `IBinaryTypeEntry` tiene el nuevo método `TypesToImport`, que puede usar para indicar una lista de tipos binarios para importar (para separar los tipos bonarios se pueden usar la coma, el punto y coma o el espacio). • La interfaz `IProjectSettingsDlg` tiene varios métodos nuevos que se pueden aplicar a la ingeniería de código C++. • La enumeración `ENUMCodeLang` tiene el nuevo miembro `eCodeLang_Cpp`, que indica el lenguaje C++ para ingeniería de código. • La enumeración `ENUMCodeLangVersion` tiene nuevos miembros, que indican las versiones del lenguaje C++. • La enumeración `ENUMUMLPredefinedElement` tiene nuevos miembros aplicables a ingeniería de código C++. Nota: para la ingeniería de código C++ se necesita la edición UModel Enterprise.

Interfaz de automatización para UModel 2017 (biblioteca de tipos versión 5.3)

API de UModel: UMLData	Se añadieron estos métodos: • `IUMLOpaqueAction::Body` • `IUMLOpaqueAction::Language`

Interfaz de automatización para UModel 2016 (biblioteca de tipos versión 5.2)

API de UModel: UMLData	Se añadieron estos métodos: • `IUMLInstanceSpecification::SetSlotInstanceValueAt` • `IUMLSlot::InsertSlotInstanceValueAt` • `IUMLDataAll::InsertSlotInstanceValueAt` • `IUMLDataAll::SetSlotInstanceValueAt`

Interfaz de automatización para UModel 2015r4 (biblioteca de tipos versión 5.1)

API de UModel: UMLData	Se añadieron estos métodos: • `IUMLGuiNodeLink::AddOwnedGuiNodeLink` • `IUMLDataAll::AddOwnedGuiNodeLink`

Interfaz de automatización para UModel 2013 (biblioteca de tipos versión 5.0)

Estos son los cambios incorporados desde la versión anterior (UModel 2012, biblioteca de tipos versión 4.1):

API de UModel	• `IDocument` tiene el método nuevo `GenerateSequenceDiagramsForAllOperations`

API de UModel: UMLData	• ENUMExportXMIType tiene una entrada nueva para UML2.4 (eXMI24ForUML24). • ENUMUMLGuiTextLabelKind tiene el literal nuevo eTextLabel_DotNetPropertyName • ENUMUMLPredefinedElement tiene literales nuevos para SysML 1.2. • IUMLGuiSequenceDiagram tiene las propiedades nuevas UseForForwardEngineering y CodeOperation para generación de código • IUMLExecutionEvent, IUMLCreationEvent, IUMLDestructionEvent, IUMLSendOperationEvent, IUMLSendSignalEvent, IUMLReceiveOperationEvent and IUMLReceiveSignalEvent se eliminaron porque las clases correspondientes clases ya no son parte de UML2.4.

Interfaz de automatización para UModel 2012 (biblioteca de tipos versión 4.1)

Estos son los cambios incorporados desde la versión anterior (UModel 2011r3, biblioteca de tipos versión 4.0):

API de UModel	• Se introducen IModelTransformationDlg, IModelTransformationTypeMappings y IModelTransformationTypeMapping para las transformaciones de modelos. IDocument tiene un método nuevo: ModelTransformation. • ILocalOptionsView tiene una propiedad nueva: EnableSnapLines.
API de UModel: UMLData	• ENUMCodeLang tiene un literal nuevo: eCodeLang_UML. • ENUMUMLPredefinedElement tiene varios literales nuevos para las transformaciones de modelos.

Interfaz de automatización para UModel 2011r3 (biblioteca de tipos versión 4.0)

Estos son los cambios incorporados desde la versión anterior (UModel 2011r2, biblioteca de tipos versión 3.2):

API de UModel	• ILocalOptionsDiagramEditing tiene propiedades nuevas para controlar propiedades .NET: **UseDotNetPropertyCompartment** y **ShowDotNetPropertyCompartment**. • IDialog tiene dos propiedades nuevas: **Application** y **Parent**. • IImportSourceDlg tiene propiedades nuevas para controlar propiedades .NET: **Content_UseDotNetPropertyCompartment** y **Content_ShowDotNetPropertyCompartment**.
API de UModel: UMLData	• Se puede establecer la propiedad **BehaviorSpecification** para IUMLBehavior • ENUMUMLGuiStyleKind tiene un literal nuevo para controlar propiedades .NET: **eUMLGuiStyle_ShowDotNetPropertyCompartment.**

Interfaz de automatización para UModel 2011r2 (biblioteca de tipos versión 3.2)

Estos son los cambios incorporados desde la versión anterior (UModel 2011, biblioteca de tipos versión 3.1):

API de UModel	• Se introduce IGenerateStateMachineCodeDlg para la generación de código de máquina de estados y IDocument tiene un método nuevo: **GenerateStateMachineCode**. • IGenerateDocumentationDlg tiene propiedades nuevas para la generación de documentación con archivos SPS (**UseFixedDesign** y **SPSFile**), una propiedad nueva (**Include_IncludedPredefinedSubprojects**) y un método nuevo (**Fonts_SetDefaults**). • ENUMDocumentationOutputFormat tiene un literal nuevo para generar documentación en formato PDF. • ENUMUMLPredefinedElement tiene varios literales nuevos para funciones relacionadas con BPMN2.
API de UModel: UMLData	• IUMLGuiBPMN2Diagram, IUMLGuiBPMN2ChoreographyDiagram y IUMLGuiBPMN2CollaborationDiagram se introducen para BPMN2

Interfaz de automatización para UModel 2011 (biblioteca de tipos versión 3.1)

Estos son los cambios incorporados desde la versión anterior (UModel 2010r3, biblioteca de tipos versión 3.0):

API de UModel	• IImportDatabaseDlg se introduce para la importación de bases de datos. • IDocument tiene un método nuevo para importar bases de datos (**ImportDatabase**) y otro para fusiones de proyecto a 3 bandas (**MergeProject3Way**).
API de UModel: UMLData	• ENUMCodeLangVersion y ENUMUMLPredefinedElement tienen literales nuevos para funciones relacionadas con bases de datos. • ENUMUMLDBDataSourceMethod se introduce para funciones relacionadas con bases de datos.

Interfaz de automatización para UModel 2010r3 (biblioteca de tipos versión 3.0)

Estos son los cambios incorporados desde la versión anterior (UModel 2010r2, biblioteca de tipos versión 2.1):

API de UModel	• IGenerateSequenceDiagramDlg tiene una propiedad nueva para omitir operaciones diferentes cuando se generen diagramas de secuencia a partir de código fuente (**OperationIgnoreList**). • La interfaz IGenerateDocumentationDlg tiene la propiedad nueva **Details_Constraints**. • ENUMDiagramLayoutKind tiene una entrada nueva para el tipo de diseño “Block” (por bloques). • ENUMCodeLangVersion tiene una entrada nueva para C# 4.0.
API de UModel: UMLData	• IUMLElement tiene un método nuevo para recuperar todos los elementos propios de un tipo determinado (**GetOwnedElementsOfKind**). • IUMLClass, IUMLEnumeration y IUMLInterface tienen un método nuevo para obtener la ruta de acceso completa del archivo de

	código (**GetCodeFilePath**). Véase también GetCodeFileName, que devuelve el nombre del archivo solamente). • IUMLInterface tiene la propiedad nueva **Protocol** y el método nuevo **SetNewProtocol** para remitir a un IUMLProtocolStateMachine. • IUMLConstraint tiene dos propiedades nuevas: **OwningTransition** y **OwningState**. • IUMLState tiene la propiedad nueva **StateInvariant** y el método nuevo **SetNewStateInvariant**. • IUMLPort tiene una propiedad nueva: **Protocol.** • IUMLStructuredClassifier tiene un método nuevo: **InsertOwnedPortAt**. • ENUMUMLPredefinedElement tiene varios literales nuevos para C# 4.0. • Además se introdujeron estas nuevas interfaces: IUMLValueSpecificationAction, IUMLProtocolStateMachine, IUMLProtocolTransition, IUMLGuiProtocolStateMachineDiagram

Interfaz de automatización para UModel 2010r2 (biblioteca de tipos versión 2.1)

Estos son los cambios incorporados desde la versión anterior (UModel 2010, biblioteca de tipos versión 2.0):

API de UModel	• IGenerateSequenceDiagramDlg tiene una propiedad nueva para la nueva división de diagramas de secuencia (**SplitIntoSmallerDiagrams**).
API de UModel: UMLData	• ENUMExportXMIType tiene una entrada nueva para UML2.3. • IUMLAction tiene una propiedad nueva: **IsLocallyReentrant**. • IUMLPort tiene una propiedad nueva: **IsConjugated**. • La propiedad **ConnectorKind** de IUMLConnector ahora es de solo lectura. • IUMLClassifier tiene una propiedad nueva: **IsFinalSpecialization**. • IUMLActivityGroup ahora se deriva de IUMLNamedElement.

Interfaz de automatización para UModel 2010 (biblioteca de tipos versión 2.0)

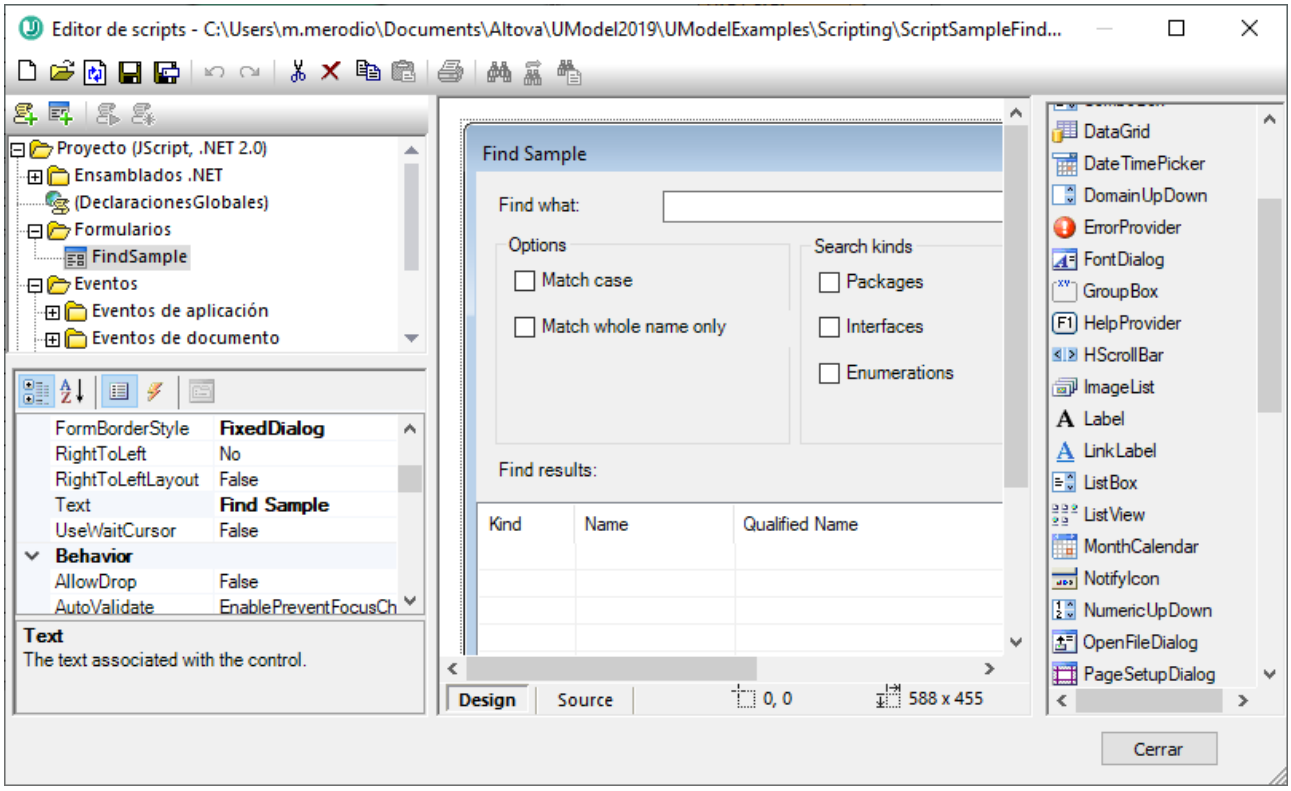
Estos son los cambios incorporados desde la versión anterior (UModel 2009, biblioteca de tipos versión 1.0):

Complementos para UModel	• Compatibilidad con controles ActiveX. Los complementos que también sean controles ActiveX aparecen en una barra de control de diálogos de UModel (véase también Controles ActiveX o el ejemplo StatisticsActiveX)
API de UModel	• IApplication tiene una propiedad nueva (**ServicePackVersion**) un método nuevo (**RunMacro**) para iniciar una macro de un proyecto de script ya cargado. • IDocument tiene métodos nuevos: **SaveCopyAs**, **CanFocusUMLDataInModelTree**, **FocusUMLDataInModelTree**, **Reload**. • La propiedad **FocusedUMLDataNotifier** de IDocument se puede usar para obtener la interfaz nueva IFocusedUMLDataEvents.

	• El método **OnModifiedFlagChanged** de IDocumentEvents tiene la interfaz IDocument como segundo parámetro. • La interfaz IDiagramWindow tiene estos métodos nuevos para la función de diseño automático: **Autolayout**, **AutolayoutSelection**. • La interfaz IProjectSettingsDlg tiene dos propiedades nuevas: **CSharp_ResolveAliases** y **VBasic_ResolveAliases**. • La interfaz IGenerateDocumentationDlg tiene dos propiedades nuevas: **EmbedCSSinHTML** y **CreateFolderForDiagrams**. • ILocalOptionsCodeEngineering tiene estas propiedades nuevas: **CodeFromModel_Indentation_InsertTabs**, **CodeFromModel_Indentation_InsertNSpaces**.
API de UModel: UMLData	• La nueva interfaz IUMLHyperlink2Model permite crear hipervínculos a elementos de modelado (en la *Estructura del modelo*). IUMLNamedElement tiene **InsertOwnedHyperlink2ModelAt**, IUMLGuiTextHyperlink tiene **SetHyperlinkModelElementAddress** para definir vínculos con elementos de modelado. • IUMLCommentTextHyperlink es nueva y habilita los hipervínculos para IUMLComments (véase también Cómo crear y usar hipervínculos). IUMLComment se amplía con **InsertOwnedCommentTextHyperlinkAt** y **OwnedHyperlinks** para insertar y acceder a estos hipervínculos. • Igualmente IUMLGuiDiagram se amplía con **InsertOwnedGuiTextHyperlinkAt** y **OwnedHyperlinks**. • La interfaz IUMLElement tiene dos propiedades nuevas (**OwnedDocComment** y **OwnedDocCommentBody**) para acceder directamente al cuerpo del comentario, que aparece en la ventana *Documentación* cuando el elemento está resaltado. • IUMLGuiDiagram tiene un método nuevo (**AddUMLGuiContainmentLink**) para insertar líneas de contención en los diagramas. • ENUMUMLGuiTextLabelKind tiene un literal nuevo: **eTextLabel_InformationFlow**. • ENUMUMLPredefinedElement tiene literales nuevos para tipos de datos XSD y el perfil SysML. • Además se añadieron estas nuevas interfaces: IUMLInformationFlow, IUMLGuiContainmentLink, IUMLGuiSysMLActivityDiagram, IUMLGuiSysMLBlockDefinitionDiagram, IUMLGuiSysMLInternalBlockDiagram, IUMLGuiSysMLPackageDiagram, IUMLGuiSysMLParametricDiagram, IUMLGuiSysMLRequirementDiagram, IUMLGuiSysMLSequenceDiagram, IUMLGuiSysMLStateMachineDiagram, IUMLGuiSysMLUseCaseDiagram

17.2 Editor de scripts

El Editor de scripts un entorno de desarrollo integrado en UModel desde donde puede personalizar las funciones de UModel con ayuda de los scripts JScript o VBScript. Por ejemplo, puede añadir un elemento nuevo de menú que ejecute una tarea personalizada en el proyecto o puede hacer que UModel desencadene algún comportamiento cada vez que se abra o cierre un documento.Para ello debe crear proyectos de scripting, que son archivos con la extensión `.asprj` (Altova Scripting Project).



Editor de scripts

Los proyectos de scripting suelen incluir una o varias macros, que son programas que ejecutan diversas tareas personalizadas cuando se invocan. Puede ejecutar macros bien de forma explícita desde un elemento de menú (o un botón de la barra de herramientas, si está configurado para ello) o configurar una para que se ejecute automáticamente siempre que se inicie UModel. El entorno de scripting también se puede integrar con la API COM de UModel. Por ejemplo, los scripts que cree con VBScript o JScript pueden gestionar eventos de aplicación o documentos como iniciar o cerrar UModel, abrir o cerrar un proyecto, etc. Los proyectos de scripting pueden incluir formularios de Windows Forms, que puede diseñar de forma visual de forma parecida a como cuando trabaja con Visual Studio. También existen varios comandos integrados de los que se puede ayudar para instanciar y usar clases .NET de código VBScript o JScript.

Una vez haya completado el proyecto de scripting puede habilitarlo de forma global en todo UModel o solamente para proyectos específicos.

El Editor de scripts necesita que instale .NET Framework 2.0 o más avanzado antes de instalar UModel.

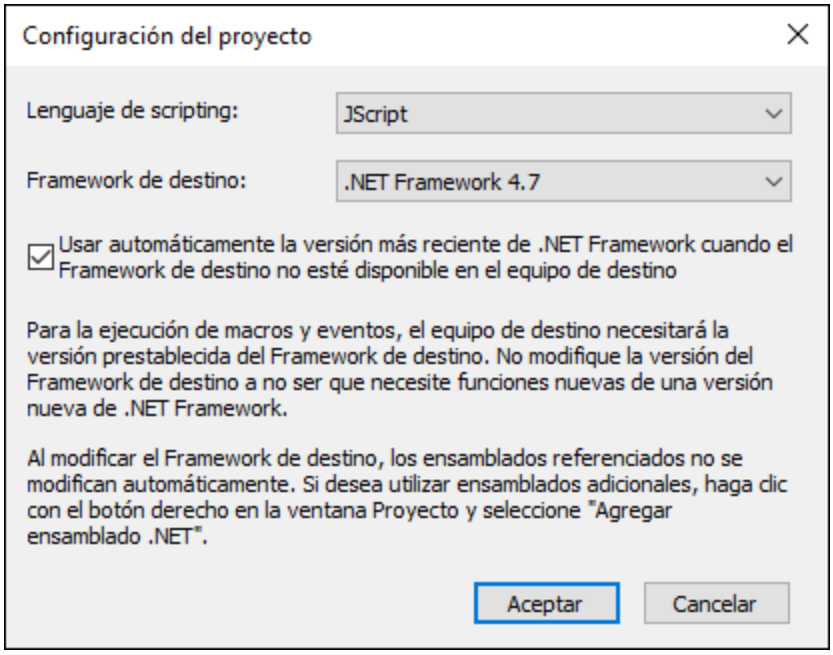
17.2.1 Crear un proyecto de scripting

Todos los scripts, así como la información de scripting que se crean en el Editor de scripts se guardan en los proyectos de scripting de Altova (archivos .asprj). Un proyecto de scripting puede contener macros, controladores de eventos de aplicaciones y formularios (que también pueden tener sus propios controladores de eventos). También puede añadir variables globales y funciones a un script "Declaraciones globales", que hace que esas variables y funciones sean accesibles en todo el proyecto.

Para empezar un proyecto nuevo ejecute el comando **Herramientas | Editor de scripts**.

Los lenguajes que se pueden usar en los proyectos de scripting son JScript y VBScript (no debe confundirse con Visual Basic, que no es compatible). Estos motores de scripting están disponibles por defecto en Windows y no precisan de requisitos especiales para ejecutarse. Para seleccionar el lenguaje de scripting que quiere usar:

1. Haga clic con el botón derecho en el elemento Proyecto, en el panel superior izquierdo y seleccione **Configuración del proyecto** en el menú contextual.
2. Seleccione un lenguaje (JScript o VBScript) y haga clic en **Aceptar**.



Desde el cuadro de diálogo "Configuración el proyecto", en la imagen anterior, también puede cambiar la versión de .NET Framework de destino. Esto suele ser necesario si su proyecto de scripting requiere alguna función que solamente exista en las versiones más recientes de .NET Framework. Tenga en cuenta que cualquier cliente que use su proyecto de scripting tendrá que tener instalada la misma versión de .NET Framework que usted (o una más reciente, siempre que sea compatible).

Por defecto, un proyecto de scripting hace referencia a varios ensamblados .NET, como `System`, `System.Data`, `System.Windows.Forms` entre otros. Si lo necesita puede importar más ensamblados .NET, incluidos ensamblados de caché de ensamblados global de .NET (GAC por sus siglas en inglés) o archivos .dll personalizados. Puede importar ensamblados:

1. De forma estática, añadiéndolos manualmente al proyecto. Haga clic con el botón derecho en el panel superior izquierdo y seleccione **Agregar ensamblado .NET** en el menú contextual.
2. De forma dinámica, en tiempo de ejecución, llamando al comando `CLR.LoadAssembly` desde el código.

Puede crear varios proyectos de scripting y guardar uno en disco para después volver a cargarlo más tarde en el Editor de scripts. Para ello use los botones estándar de Windows de la barra de herramientas: **Nuevo**, **Abrir**, **Guardar**, **Guardar como**. Una vez haya probado el proyecto y esté listo para implementarse puede cargarlo en UModel y ejecutar cualquiera de sus macros o controladores de eventos. Para más información consulte el apartado Habilitar scripts y macros.

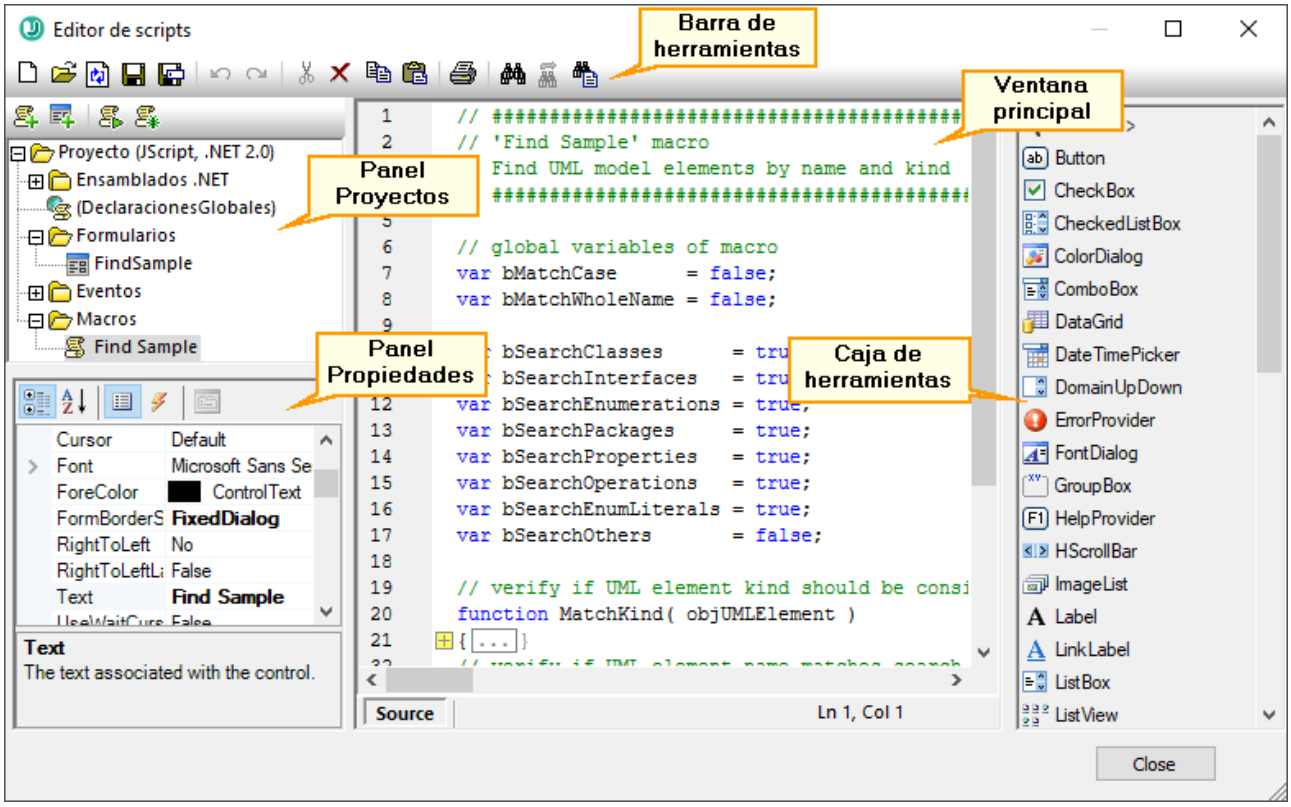
También puede encontrar un proyecto de scripting de ejemplo en: **C:\Usuarios\<user>\Documentos\Altova\UModel2023\UModelExamples\Scripting\ScriptSampleFind.asprj**.

Las secciones siguientes se centran en las partes de un proyecto de scripting: declaraciones globales, macros, formularios y eventos.

17.2.1.1 Información general

El Editor de scripts contiene estas partes:

- Barra de herramientas
- Panel Proyectos
- Panel Propiedades
- Ventana principal
- Cuadro de herramientas



Barra de herramientas

La barra de herramientas incluye comandos estándar de Windows para la gestión de archivos (**Nuevo**, **Abrir**, **Guardar**, **Guardar como**) y comandos de edición (**Copiar**, **Cortar**, **Eliminar**, **Pegar**). Al editar código fuente también se activan los comandos **Buscar** y **Reemplazar**, así como el comando **Imprimir**.

Panel Proyectos

En el panel Proyectos puede ver y gestionar la estructura del proyecto. Un proyecto de scripting consiste en varios componentes que pueden funcionar juntos y que no tienen que crearse en un orden determinado:

- *Un script "Declaraciones globales"*. Como el nombre indica, este script contiene información que se usa en todo el proyecto. En este script puede declarar cualquier variable o función que necesite tener disponible para todos los formularios, scripts de gestión de eventos y macros.
- *Formularios*. Los formularios suelen ser necesarios para recopilar información de los usuarios o para suministrar cuadros de diálogo informativos.Por ejemplo, imagine que crea un proyecto de scripting en el que aparece un formulario de entrada que permite al usuario introducir un elemento "nombre" y hacer clic en un botón **Eliminar**. Al hacer clic en ese botón todas las instancias del elemento introducido se eliminan del proyecto de UModel. Un formulario se invoca llamándolo sea con una función (en el script *Declaraciones globales*) o directamente con una macro.
- *Eventos*. La carpeta "Eventos" muestra eventos de la aplicación UModel suministrados por la API COM. Para escribir un script que se ejecute cuando ocurra un evento haga doble clic en cualquier evento y después teclee el código de manejo en el editor. No debe confundir los eventos de la

aplicación con los eventos de los formularios; estos últimos se manejan a nivel del formulario, como detallamos más adelante.

- *Macros*. Una macro es un script que puede invocarse a petición desde un menú contextual o ejecutarse automáticamente cuando empiece UModel. Las macros no tienen parámetros o valores de retorno. Una macro puede acceder a todas las variables y funciones declaradas en el script *Declaraciones globales* y también puede mostrar formularios.

Haga clic con el botón derecho en cualquiera de los componentes para ver los comandos del menú contextual y los atajos de teclado correspondientes. Haga doble clic en cualquier archivo (como un formulario o un script) para abrirlo en la ventana principal.

Con los botones de la barra de herramientas puede acceder a estos comandos rápidos:

	**Macro nueva**	Agrega una macro nueva al proyecto en el directorio **Macros**.
	**Formulario nuevo**	Agrega un formulario nuevo al proyecto en el directorio **Formularios**.
	**Ejecutar macro**	Ejecuta la macro seleccionada.
	**Depurar macro**	Ejecuta la macro seleccionada en modo depuración.

Panel Propiedades

El panel Propiedades es muy parecido al de Visual Studio y contiene:

- Propiedades del formulario, si se selecciona uno
- Propiedades del objeto, si se selecciona uno
- Eventos del formulario, si se selecciona uno
- Eventos del objeto, si se selecciona uno

Para alternar entre las propiedades y los eventos del componente seleccionado haga clic en los botones Propiedades o Eventos .

Los iconos **Por categorías**  y **Alfabético**  muestran las propiedades o los eventos organizados por categorías o por orden alfabético ascendente.

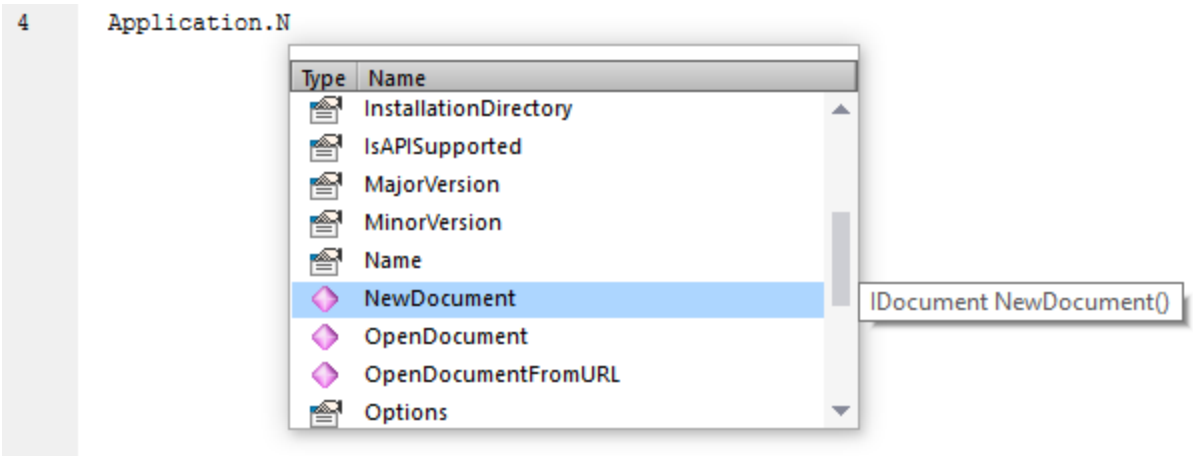
Cuando se selecciona una propiedad o un evento aparece una breve descripción de ese elemento en la parte inferior del panel Propiedades.

Ventana principal

La ventana principal es el área de trabajo donde puede introducir código fuente o modificar el diseño del formulario. Al editar formularios puede trabajar en dos pestañas: *Design* y *Source*. La pestaña *Design* muestra el diseño del formulario, mientras que la pestaña *Source* contiene el código fuente, como métodos de control de eventos.

El editor de código fuente cuenta con ayudas para la edición de código, como color de sintaxis, plegamiento de código, resaltado de los corchetes de inicio y final, zoom, sugerencias de finalización automática y marcadores.

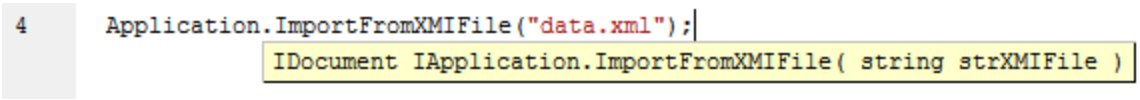
Sugerencias de finalización automática
JScript y VBScript son dos lenguajes sin tipo, por lo que la finalización automática se limita a los nombres de la API COM y a los comandos integrados de UModel. El método completo o la firma de la propiedad aparecen junto al ayudante de entrada de finalización automática.



Si los nombres empiezan por `objUMLxxx`, entonces se muestran los miembros de la interfaz `IUMLxxx` correspondiente. Por ejemplo, la API COM de UModel tiene una interfaz, `IUMLClass`. Si usa nombres como `objUMLClass`, `objUMLClass123` o `objUMLClassParent` se mostrarán los miembros de la `IUMLClass` correspondiente.

Si los nombres empiezan por `objApplication`, `objDocument` o `objDiagramWindow`, entonces se mostrarán los miembros de la interfaz correspondiente. Esto también se aplica al resto de interfaces definidas en la API de UModel.

Para ver la firma (y la documentación, si la hay) de un método o una propiedad conocidos, coloque el cursor del ratón sobre el elemento en cuestión, por ejemplo:



El ayudante de entrada de finalización automática suele aparecer automáticamente al editar, pero también puede activarlo pulsando **Ctrl+Barra espaciadora**.

Marcadores
- Para guardar o eliminar un marcador, haga clic dentro de una línea y después pulse **Ctrl+F2**
- Para ir al marcador siguiente pulse **F2**
- Para ir al marcador anterior pulse **Mayús+F2**
- Para eliminar todos los marcadores pulse **Ctrl+Mayús+F2**

Alejarse y acercarse con el zoom
- Para alejarse y acercarse con el zoom mantenga pulsada la tecla Ctrl y pulse las teclas "+" o "-", o gire la rueda del ratón.

Configurar la vista Texto
Para activar estas opciones haga clic con el botón derecho en el editor y seleccione **Configuración de la vista Texto** en el menú contextual.

Fuentes
Para cambiar la fuente haga clic con el botón derecho en el editor y seleccione **Fuentes de la vista Texto...** en el menú contextual.

Caja de herramientas

La Caja de herramientas contiene todos los objetos que hay disponibles para diseñar formularios, como son los botones, las cajas de texto, los cuadros combinados, etc.

Para añadir un elemento de la Caja de herramientas a un formulario:

1. Cree o abra un formulario y seleccione la pestaña **Diseño**.
2. Haga clic en el objeto de la caja de herramientas (por ejemplo, **Botón**) y después haga clic en el lugar del formulario donde lo quiere insertar. También puede arrastrar el objeto directamente hasta el formulario.

Algunos objetos, como el `Temporizador`, no se añaden al formulario, sino que se crean en una bandeja, en la parte inferior de la ventana principal. Puede seleccionar el objeto en la bandeja y configurar las propiedades y los controladores de eventos para el objeto del panel Propiedades. Para ver un ejemplo consulte el apartado Control de eventos de formularios.

También puede añadir controles ActiveX registrados al formulario. Para ello haga clic con el botón derecho en el área de la Caja de herramientas y seleccione **Agregar control ActiveX** en el menú contextual.

17.2.1.2 Declaraciones globales

El script "Declaraciones globales" existe por defecto en todos los proyectos de scripting, por lo que no lo tiene que crear explícitamente. Cualquier variable o función que quiera añadir a este script se considera como global a todo el proyecto. Esto significa que puede hacer referencia a esas variables y funciones desde cualquier macro o evento del proyecto. A continuación se muestra un extracto de ejemplo del script Declaraciones globales que importa el espacio de nombres `System.Windows.Forms` en el proyecto. Para ello el código invoca el comando `CLR.Import` que está integrado en el Editor de scripts.

```
// importa el espacio de nombres System.Windows.Forms para todas las macros, los
formularios y eventos:
CLR.Import( "System.Windows.Forms" );
```

Nota: cada vez que se ejecuta una macro o que se llama a un controlador de eventos las declaraciones globales se vuelven a inicializar.

17.2.1.3 Macros

Las macros son scripts que contienen declaraciones JScript (o VBScript, en función del lenguaje del proyecto), como declaraciones y funciones variables.

Puede añadir macros a su proyecto si quiere; para ello haga clic con el botón derecho en el panel Proyectos, seleccione **Agregar macro** en el menú contextual y después introduzca el código de la macro en el formulario principal. Ese código puede ser tan simple como una alerta, por ejemplo:

```
alert("Hola, soy una macro");
```

Una macro más avanzada podría contener variables y funciones locales. Las macros también pueden contener código que invoque formularios desde el proyecto. El extracto siguiente representa una macro de ejemplo que muestra un formulario. Se asume que el formulario ya se ha creado en la carpeta "Formularios" y que se le ha dado el nombre "FormularioEjemplo" (consulte también el apartado Formularios).

```
// muestra un formulario
ShowForm( "SampleForm" );
```

En el extracto anterior `ShowForm` es un comando integrado en el Editor de scripts. Para ver más comandos que puede usar para trabajar con formularios y objetos .NET consulte el apartado Comandos integrados.

Puede agregar varias macros al mismo proyecto y puede convertir cualquier macro en "automática". Al hacerlo, la macro se ejecuta automáticamente cuando se inicia UModel. Para definir una macro como automática haga clic con el botón derecho en ella y seleccione **Establecer como macro automática** en el menú contextual.

Solo se puede ejecutar una macro a la vez. Una vez se ha ejecutar la macro (o el evento) el script se cierra y las variables globales pierden sus valores.

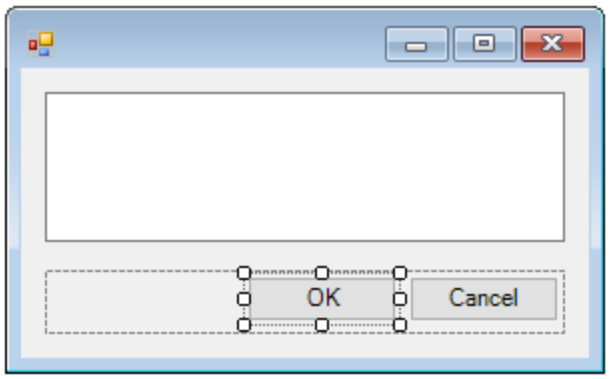
Para ejecutar una macro directamente en el Editor de scripts haga clic en Ejecutar macro  . Para depurar una macro con el depurador de Visual Studio haga clic en Depurar macro  . Para más información sobre cómo habilitar y ejecutar macros en UModel, consulte el apartado Habilitar scripts y macros.

17.2.1.4 Formularios

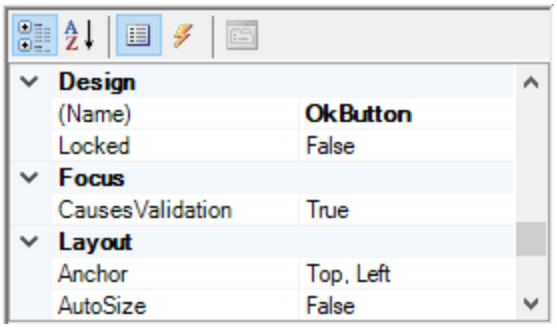
Los formularios son especialmente prácticos si necesita recolectar datos de entrada de usuarios o mostrar datos a usuarios. Para llevar a cabo esas acciones un formulario puede contener varios controles como botones, casillas de verificación, cuadros combinados, etc.

Para añadir un formulario haga clic con el botón derecho en el panel Proyecto y seleccione **Agregar formulario** en el menú contextual. Para añadir un control a un formulario arrástrelo desde la Caja de herramientas que está a la derecha del Editor de scripts y suéltela en el formulario.

Puede cambiar la posición y el tamaño de los controles directamente en el formulario usando las manijas que aparecen al hacer clic en un control, por ejemplo:



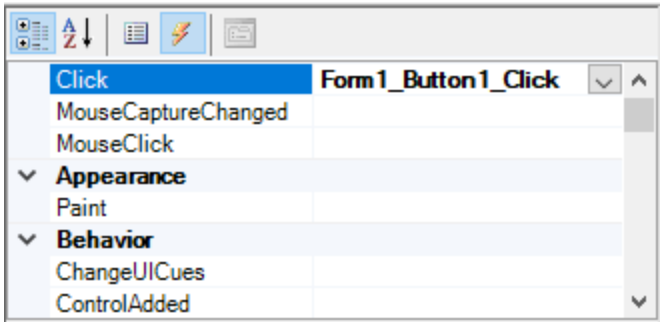
Todos los controles de los formularios tienen propiedades que puede ajustar en el panel Propiedades. Para ello, seleccione primero el control en el formulario y después edite las propiedades en cuestión en el panel Propiedades.



Control de eventos de formularios

Cada control del formulario expone también varios eventos a los que puede vincular el proyecto de scripting. Por ejemplo, puede que quiera invocar algún método de la API COM de UModel cuando se haga clic en un botón. Para crear una función que vincule a un evento de formulario siga estos pasos:

1. En el panel Propiedades haga clic en **Eventos** .
2. En la columna Acción haga doble clic en el evento donde necesita el método (por ejemplo, en la imagen siguiente, el evento de controlador es "Click").



También puede añadir métodos de controlador haciendo doble clic en un control del formulario. Por ejemplo, al hacer doble clic en un botón del diseño del formulario se genera un método de controlador para el evento "Click" de ese botón.

Una vez se ha generado el cuerpo del método del controlador puede teclear el código que controla este evento, por ejemplo:

```
//Ocurre cuando se hace clic en el componente.
function MyForm_ButtonClick( objSender, e_EventArgs )
{
   alert("A button was clicked");
}
```

Para mostrar un formulario sin terminar fuera del Editor de scripts haga clic con el botón derecho en el formulario y seleccione **Probar el formulario** en el menú contextual. Observe que el comando **Probar el formulario** solamente muestra el formulario; los eventos del formulario (como hacer clic en un botón) siguen estando deshabilitados. Para que el formulario reaccione a los eventos debe llamarlo desde una macro, por ejemplo:

```
// Instanciar y mostrar un formulario
ShowForm( "SampleForm" );
```

Acceder a los controles del formulario

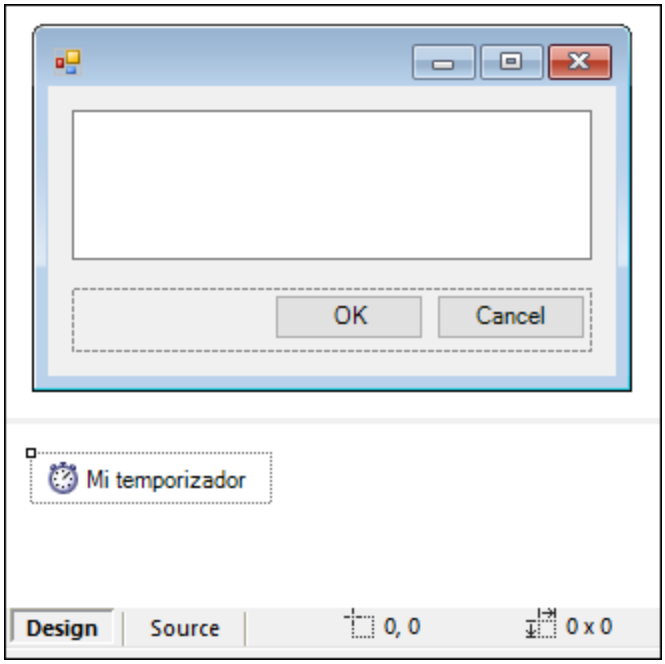
Puede acceder a cualquier componente de un formulario desde el código usando sintaxis de acceso a los campos. Por ejemplo, imagine que tiene un formulario con este diseño:

```
// MiFormulario
//   PanelBotón
//      BotónAceptar
//      BotónCancelar
//   EditorTexto
//      ReproductorMediaAx1
// ComponentesBandeja
//   MiTemporizador
```

El código siguiente muestra cómo instanciar el formulario, acceder a algunos de sus controles con la sintaxis de acceso a los campos y después mostrarlo:

```
// Instanciar el formulario
var objForm = CreateForm("MyForm");
// Deshabilitar el botón Aceptar
objForm.ButtonPanel.OkButton.Enabled = false;
// Cambiar el texto del Editor de texto
objForm.TextEditor.Text = "Hello";
// Mostrar el formulario
objForm.ShowDialog();
```

Al añadir al formulario ciertos controles, como los temporizadores, estos no aparecen en el formulario, sino que se muestran como componentes de la bandeja en la parte inferior del diseño del formulario, por ejemplo:



Para acceder a los controles de la bandeja puede usar el método `GetTrayComponent` en el objeto formulario y suministrar el nombre del control como argumento. En este ejemplo, para obtener una referencia a `MiTemporizador` y habilitarlo debe usar este código:

```
var objTimer = objForm.GetTrayComponent("MiTemporizador");
objTimer.Enabled = true;
```

En el caso de los controles ActiveX puede acceder al objeto COM subyacente con la propiedad OCX:

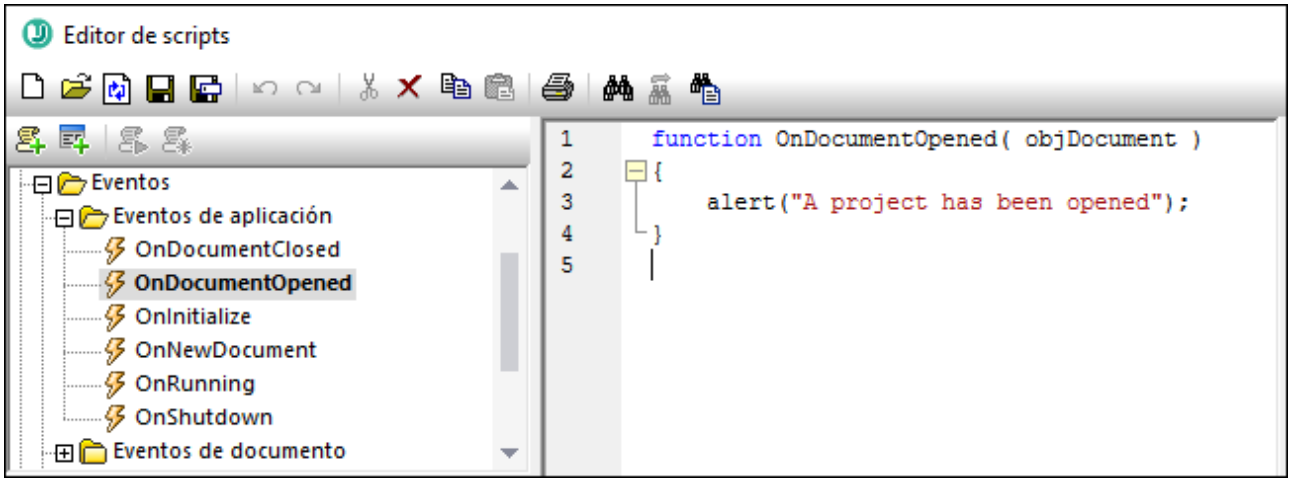
```
var ocx = lastform.AxMediaPlayer1.OCX; // obtener objeto COM subyacente
ocx.enableContextMenu = true;
ocx.URL = "mms://apasf.apa.at/fm4_live_worldwide";
```

17.2.1.5 Eventos

Un proyecto de scripting puede incluir scripts que manejen eventos de UModel como abrir, cerrar o guardar un documento, iniciar o cerrar UModel, añadir un elemento a un diagrama, etc. Estos eventos los suministra la API COM de UModel y los puede encontrar en la carpeta "Eventos" del proyecto de scripting. Tenga en cuenta que estos eventos son eventos específicos de UModel, en contraposición a los eventos de formulario. Los eventos están organizados en carpetas:

- Eventos de aplicación
- Eventos de documento
- Eventos de transacción
- Eventos de Datos UML
- Eventos orientados a Datos UML

Para crear un script de controlador de eventos haga clic con el botón derecho en un evento y seleccione **Abrir** en el menú contextual (o haga doble clic en el evento). El script del controlador de eventos aparece en la ventana principal, donde puede editarlo. Por ejemplo, el controlador de eventos de la imagen siguiente muestra una alerta cada vez que el proyecto se abre en UModel:



Tenga en cuenta que:

- El comando `alert` se puede aplicar a JScript. El equivalente en VBScript es `MsgBox`. Véase también alert.
- No debe cambiar el nombre de la función del controlador de eventos; de lo contrario no se llamará al script del controlador de eventos.
- Para que se procesen eventos debe marcar la casilla *Procesar eventos* al habilitar el proyecto de scripting en UModel. Para más información consulte el apartado Habilitar scripts y macros.

También puede definir variables locales y funciones de ayuda dentro del script del controlador, por ejemplo:

```
var local;

function OnInitialize( objApplication )
{
   local = "OnInitialize";
   Helper();
}

function Helper()
{
   alert("I'm a helper function for " + local);
}
```

17.2.1.6 Consejos para programar con JScript

A continuación ofrecemos algunos consejos para programar con JScript que le ayudarán a la hora de desarrollar un proyecto de scripting en el Editor de scripts de UModel.

Parámetros out

Los parámetros out de los métodos de NET Framework requieren variables especiales en JScript. Por ejemplo:

```
var dictionary =
CLR.Create("System.Collections.Generic.Dictionary<System.String,System.String>");
dictionary.Add("1", "A");
dictionary.Add("2", "B");

// use el método de JScript para acceder a los parámetros out
var strOut = new Array(1);
if ( dictionary.TryGetValue("1", strOut) ) // TryGetValue establece el parámetro out
  alert( strOut[0] ); // use out parameter
```

Números enteros como argumentos

Los métodos .NET que requieren números enteros como argumentos no se deben llamar directamente con los objetos numéricos de JScript, que son valores de puntos flotantes. Por ejemplo, en lugar de:

```
var objCustomColor = CLR.Static("System.Drawing.Color").FromArgb(128,128,128);
```

use:

```
var objCustomColor =
CLR.Static("System.Drawing.Color").FromArgb(Math.floor(128),Math.floor(128),Math.floor(12
8));
```

Iterar colecciones .NET

Para iterar colecciones .NET se pueden usar el Enumerador de JScript y el iterador .NET. Por ejemplo:

```
// iterar con el iterador de JScript
var itr = new Enumerator( coll );
for ( ; !itr.atEnd(); itr.moveNext() )
  alert( itr.item() );

// iterar con el iterador .NET
var itrNET = coll.GetEnumerator();
while( itrNET.MoveNext() )
  alert( itrNET.Current );
```

Plantillas .NET

Las plantillas .NET se pueden instanciar así:

```
var coll = CLR.Create( "System.Collections.Generic.List<System.String>" );
```

o así

```
CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary<String,Dictionary<String,String>>" );
```

Valores .NET de enumeración

Para acceder a los valores .NET de enumeración puede usar:

```
var enumValStretch = CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch;
```

Literales de enumeración

Para acceder a los literales de enumeración de la API de UModel (no necesita conocer su valor numérico) puede usar:

```
objExportXMIFileDlg.XMIType = eXMI21ForUML23;
```

17.2.1.7 Ejemplo: proyecto de scripting

UModel viene con un proyecto de scripting de ejemplo que puede encontrar aquí: **C:\Users\<user>\Documents\Altova\UModel2023\UModelExamples\Scripting\ScriptSampleFind.asprj**.

Este proyecto de scripting consiste en una macro y un formulario de Windows. En el formulario puede buscar paquetes, interfaces, operaciones u otros tipos de elementos UML dentro del proyecto de UModel que esté abierto en ese momento. Puede elegir el tipo de elemento que quiere buscar y también indicar si quiere que la búsqueda tenga en cuenta mayúsculas y minúsculas, o si quiere buscar solamente palabras completas.

Para cargar el proyecto de scripting en el Editor de scripts:

1. En el menú Herramientas haga clic en **Editor de scripts**.
2. Haga clic en Abrir y navegue hasta el archivo **ScriptSampleFind.asprj** desde la ruta indicada más arriba.

Observe que el proyecto contiene una macro llamada **Find Sample** en el directorio "Macros". Asimismo, en el directorio "Formularios" hay un formulario de búsqueda que contiene varios controladores de eventos de formulario.

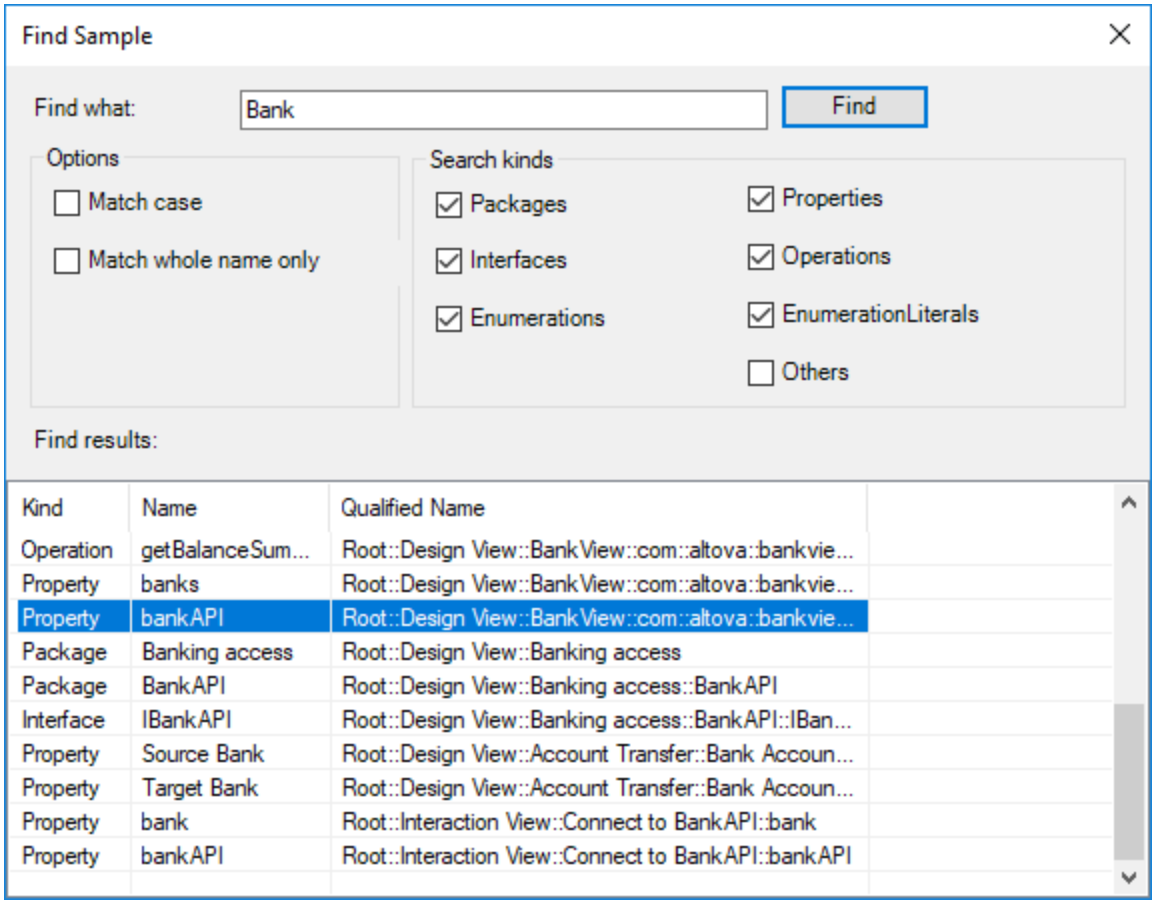
Para habilitar el proyecto de scripting como proyecto global de scripting de UModel:

1. En el menú **Herramientas** haga clic en **Opciones**.
2. Haga clic en la pestaña **Sctipt/Scripting**.
3. En *Archivo de proyecto de scripting global* haga clic en **Examinar** y seleccione el archivo **ScriptSampleFind.asprj** en la ruta indicada anteriormente.
4. Este proyecto de scripting no tiene macros automáticas o controladores de eventos; por lo tanto no necesita marcar las casillas **Ejecutar macros automáticas...** o **Procesar eventos**.
5. Haga clic en **Aplicar**.

En este punto en el menú **Herramientas | Macros** aparece un elemento de menú nuevo llamado **Find Sample**. Este elemento de menú nuevo llama a la macro del proyecto de scripting.

Para ejecutar la macro:

1. Abra un proyecto de UModel que contenga varios paquetes, operaciones, etc. (en este ejemplo, **C:\Users\<user>\Documents\Altova\UModel2023\UModelExamples\Bank_Java.ump**).
2. En el menú **Herramientas** haga clic en **Macros** y después en **Find Sample**.
3. Teclee el término de búsqueda y haga clic en **Buscar**.



En la imagen anterior se ve cómo aparecen todos los elementos del proyecto cuyo nombre contiene el término de búsqueda. Puede hacer clic en cualquiera de esos elementos para seleccionarlo en la ventana Proyecto.

17.2.2 Comandos integrados

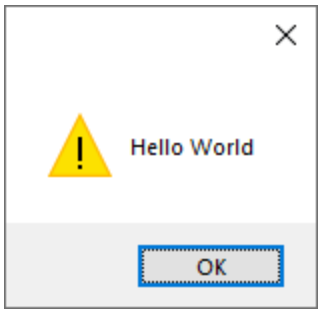
En este apartado se explican todos los comandos que puede usar el el Editor de scripts de UModel.

- alert
- confirm
- CLR.Create
- CLR.Import

- CLR.LoadAssembly
- CLR.ShowImports
- CLR.ShowLoadedAssemblies
- CLR.Static
- CreateForm
- doevents
- lastform
- prompt
- ShowForm
- watchdog

17.2.2.1 alert

Muestra un cuadro de mensaje en el que aparece un texto dado y el botón "Aceptar". Para continuar el usuario debe hacer clic en "Aceptar".



Firma

Para JScript la firma es:

```
alert(strMessage : String) -> void
```

Para VBScript la firma es:

```
MsgBox(strMessage : String) -> void
```

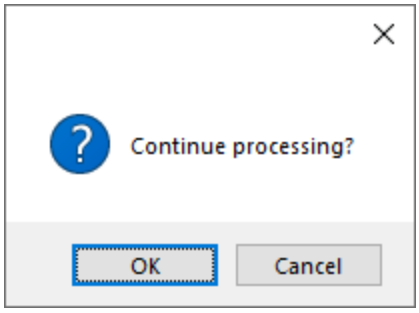
Ejemplo

Est código JScript muestra una caja de mensaje con el texto "Hello, World".

```
alert("Hello World");
```

17.2.2.2 confirm

Abre un cuadro de diálogo en el que aparecen un mensaje dado, un botón para confirmar y uno para cancelar. El usuario tiene que hacer clic en "Aceptar" o "Cancelar" para continuar. Devuelve un valor booleano que representa la respuesta del usuario. Si este hace clic en "Aceptar" la función devuelve **true**; si hace clic en "Cancelar" la función devuelve **false**.



Firma

```
confirm(strMessage : String) -> result : Boolean
```

Ejemplo (JScript)

```
if ( confirm( "¿Seguir procesando?" ) == false )
  alert("Ha cancelado esta acción");
```

Ejemplo (VBScript)

```
If ( confirm( "¿Seguir procesando?" ) = false ) Then
  MsgBox("Ha cancelado esta acción")
End If
```

17.2.2.3 CLR.Create

Crea una instancia de objeto .NET nueva con el nombre de clase dado como argumento. Si se pasa más de un argumento, los argumentos siguientes se interpretan como argumentos para el constructor del objeto .NET. El valor de retorno es una referencia al objeto .NET que se creó.

Firma

```
CLR.Create(strTypeNameCLR : String, constructor arguments ... ) -> object
```

Ejemplo

Este extracto de código de JScript ilustra cómo crear instancias de varias clases .NET.

```
// Crear una lista ArrayList
var objArray = CLR.Create("System.Collections.ArrayList");
// Crear una lista ListViewItem
var newItem = CLR.Create( "System.Windows.Forms.ListViewItem", "NewItemText" );
// Crear una lista List<string>
var coll = CLR.Create( "System.Collections.Generic.List<System.String>" );
// Importa los espacios de nombres necesarios y crea un objeto Diccionario
CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary< String, Dictionary< String, String > >" );
```

17.2.2.4 CLR.Import

Importa un espacio de nombres. Este es el equivalente en scripting de `using` en C# o `imports` en VB.Net. Al llamar a `CLR.Import` no se puede excluir la parte del espacio de nombres en llamadas posteriores, como `CLR.Create()` and `CLR.Static()`.

Nota: al importar un espacio de nombres no se añade ni se carga el ensamblado correspondiente en el proyecto de scripting. Puede añadir ensamblados al proyecto de scripting de forma dinámica (en el momento de la ejecución) en el código fuente llamando a CLR.LoadAssembly.

Firma

```
CLR.Import(strNamespaceCLR : String) -> void
```

Ejemplo

En lugar de tener que usar espacios de nombres completos, como aquí:

```
if ( ShowForm( "FormName" ) == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " + lastform.textboxLastName.Text;
    CLR.Static( "System.Windows.Forms.MessageBox" ).Show( "Hola " + sName );
}
```

Puede importar espacios de nombres primero y después usar la forma abreviada:

```
CLR.Import( "System.Windows.Forms" );

if ( ShowForm( "FormName" ) == CLR.Static( "DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " + lastform.textboxLastName.Text;
    CLR.Static( "MessageBox" ).Show( "Hola " + sName );
}
```

17.2.2.5 CLR.LoadAssembly

Carga el ensamblado .NET con el nombre de ensamblado largo dado o con la ruta de acceso al archivo. Devuelve el valor booleano **true** si el ensamblado se pudo cargar; en caso contrario devuelve **false**.

Firma

```
CLR.LoadAssembly(strAssemblyNameCLR : String, showLoadErrors : Boolean) -> result :
Boolean
```

Ejemplo

Este código JScript intenta establecer el texto del portapapeles cargando el ensamblado necesario de forma dinámica.

```
// establecer el texto del portapapeles (si se puede)
// System.Windows.Clipboard es parte del ensamblado PresentationCore, así que carga
primero este ensamblado:
if ( CLR.LoadAssembly( "PresentationCore, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35", true ) )
{
    var clipboard = CLR.Static( "System.Windows.Clipboard" );
    if ( clipboard != null )
        clipboard.SetText( "HelloClipboard" );
}
```

17.2.2.6 CLR.ShowImports

Abre un cuadro de mensaje que muestra el espacio de nombres importado en ese momento. El usuario tiene que hacer clic en "Aceptar" para continuar.

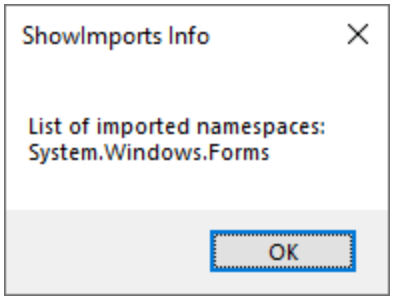
Firma

```
CLR.ShowImports() -> void
```

Ejemplo

Este código JScript primero importa un espacio de nombres y después muestra la lista de espacios de nombres importados:

```
CLR.Import( "System.Windows.Forms");
CLR.ShowImports();
```



17.2.2.7 CLR.ShowLoadedAssemblies

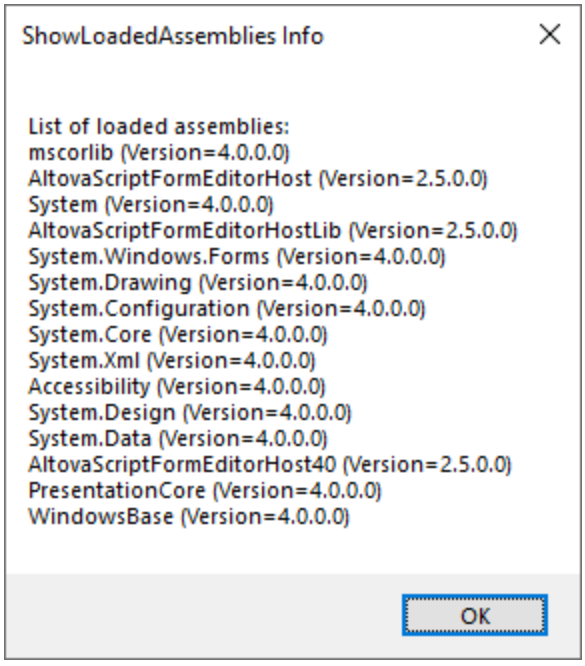
Abre un cuadro de mensaje que muestra los ensamblados cargados en ese momento. El usuario tiene que hacer clic en "Aceptar" para continuar.

Firma

```
CLR.ShowLoadedAssemblies() -> void
```

Ejemplo

```
CLR.ShowLoadedAssemblies();
```



17.2.2.8 CLR.Static

Devuelve una referencia a un objeto .NET estático. Puede usar esta función para acceder a tipos .NET que no tengan instancias y contengan solamente miembros estáticos.

Firma

```
CLR.Static(strTypeNameCLR : String) -> object
```

Ejemplo (JScript)

```
// Obtener el valor de una .NET Enum en una variable
var enumValStretch = CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch

// Establecer el valor del portapapeles de Windows
var clipboard = CLR.Static( "System.Windows.Clipboard" );
clipboard.SetText( "HelloClipboard" );

// Comprobar qué botón pulsó el usuario en un cuadro de diálogo
if ( ShowForm( "FormName" ) == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
    alert( "ok" );
else
    alert( "cancel" );
```

17.2.2.9 CreateForm

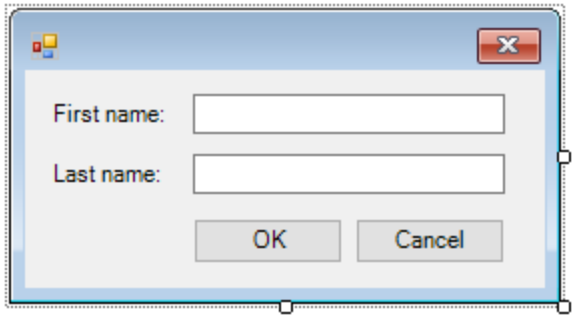
Instancia el objeto formulario `Form` identificado por el nombre dado como argumento. El formulario debe existir en la carpeta "Formularios" del proyecto de scripting. Devuelve el objeto formulario (`System.Windows.Forms.Form`) que corresponde al nombre dato o `null` si no existe ningún formulario con ese nombre.

Firma

```
CreateForm (strFormName : String) -> System.Windows.Forms.Form | null
```

Ejemplo

Imagine que en el proyecto de scripting existe un formulario llamado "FormName".



Este código JScript instancia el formulario con algunos valores predeterminados y lo muestra al usuario.

```
var myForm = CreateForm( "FormName" );
if ( myForm != null )
{
   myForm.textboxFirstName.Text = "Daniela";
   myForm.textboxLastName.Text = "Heidegger";
   var dialogResult = myForm.ShowDialog();
}
```

En consecuencia, `dialogResult` se puede seguir evaluando así:

```
if ( dialogResult == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
    alert( "ok" );
else
    alert( "cancel" );
```

Nota: el código de la imagen anterior solamente funciona si la propiedad `DialogResult` de los botones "Aceptar" y "Cancelar" está configurada correctamente en el panel *Propiedades* (por ejemplo, debe ser `OK` para el botón "Aceptar").

17.2.2.10 doevents

Procesa todos los mensajes de Windows que estén en ese momento en la cola de mensajes.

Firma

```
doevents() -> void
```

Ejemplo (JScript)

```
for ( i=0; i < nLongLastingProcess; ++i )
{
    // ejecutar procesos de larga duración
```

```
    doevents(); // procesar mensajes de Windows; permitir que la IGU se actualice
}
```

17.2.2.11 lastform

Este es un campo global que devuelve una referencia al último objeto formulario que se creó con `CreateForm()` o `ShowForm()`.

Firma

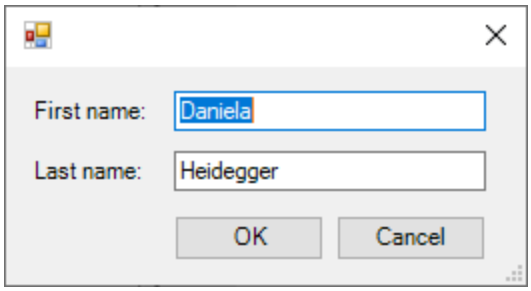
```
lastform -> formObj : System.Windows.Forms.Form
```

Ejemplo

Este código JScript muestra el formulario "FormName" como cuadro de diálogo.

```
CreateForm( "FormName" );
if ( lastform != null )
{
    lastform.textboxFirstName.Text = "Daniela";
    lastform.textboxLastName.Text = "Heidegger";
    var dialogResult = lastform.ShowDialog();
}
```

Los valores de los dos controles de campos de texto se inicializan con ayuda de `lastform`.



17.2.2.12 prompt

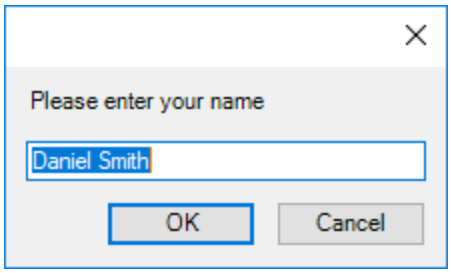
Abre un cuadro de diálogo en el que aparece un mensaje y un control de campo de texto con una respuesta predeterminada. Esto permite al usuario introducir un valor simple de cadena de texto. El valor de retorno es una cadena que contiene el valor del campo de texto o `null` si el usuario seleccionó "Cancelar".

Firma

```
prompt(strMessage : String, strDefault : String) -> val : String
```

Ejemplo

```
var name = prompt( "Please enter your name", "Daniel Smith" );
if ( name != null )
    alert( "¡Hola, " + name + "!" );
```



17.2.2.13 ShowForm

Instancia un objeto formulario nuevo a partir del nombre de formulario dado y lo muestra de inmediato como cuadro de diálogo. El valor de retorno es un número entero que representa el resultado generado: `DialogResult` (`System.Windows.Forms.DialogResult`). Para ver la lista de posibles valores consulte la documentación de `DialogResult` Enum (https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.dialogresult?view=netframework-4.8).

Firma

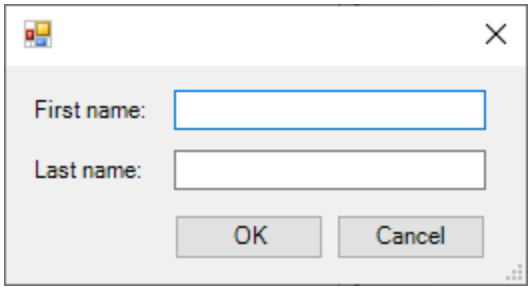
```
ShowForm(strFormName : String) -> result : Integer
```

Ejemplo

Este código JScript

```
var dialogResult = ShowForm( "FormName" );
```

muestra el formulario "FormName" como cuadro de diálogo:



Así, `DialogResult` se puede seguir evaluando, por ejemplo:

```
if ( dialogResult == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
    alert( "ok" );
else
    alert( "cancel" );
```

Nota: el código de la imagen anterior solamente funciona si la propiedad `DialogResult` de los botones "Aceptar" y "Cancelar" está configurada correctamente en el panel *Propiedades* (por ejemplo, debe ser `OK` para el botón "Aceptar").

17.2.2.14 watchdog

Los scripts largos que dan un uso intensivo a la CPU pueden preguntar al usuario si el script debe finalizar. El método `watchdog()` se usa para deshabilitar o habilitar la opción de perro guardián. Por defecto, esta opción está habilitada.

También se puede usar `watchdog(true)` para reiniciar la función de perro guardián. Esto puede ser útil antes de ejecutar tareas largas que hagan un uso intensivo de la CPU para asegurarse de que cuentan con la capacidad de procesamiento máxima permitida.

Firma

```
watchdog(bEnable : boolean) -> void
```

Ejemplo

```
watchdog( false );  // deshabilitar watchdog: la siguiente declaración hace un uso
intensivo de la CPU
doCPUIntensiveScript();
watchdog( true );   // volver a habilitar watchdog
```

17.2.3 Habilitar scripts y macros

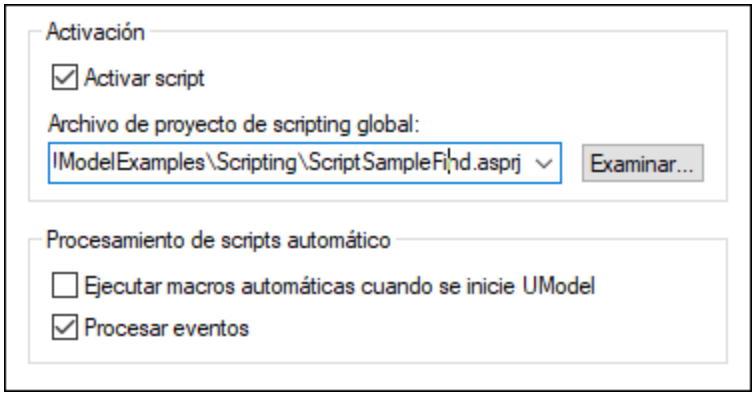
Una vez ha completado y probado un proyecto de scripting puede usarlo de varias maneras:

1. Como proyecto global de scripting para UModel. Esto significa que UModel puede usar todos los scripts y las macros del proyecto de scripting.
2. A nivel del proyecto. Esto significa que junto al proyecto de UModel se guarda también una referencia al archivo .asprj. Cuando se abre el proyecto de UModel también se puede llamar a los scripts y las macros asociados.

Para definir un proyecto de scripting como global:

1. En el menú **Herramientas** haga clic en **Opciones**.
2. Haga clic en la pestaña **Script/Scripting**.

3. Marque la casilla *Activar script* y navegue hasta el archivo `.asprj` para usarlo como proyecto global de scripting.



También puede habilitar estas otras opciones de procesamiento de scripts:

Ejecutar macros automáticas cuando se inicie UModel	Si marca esta casilla, cualquier macro que esté definida como "automática" en el proyecto se desencadena automáticamente cuando se inicia UModel.
Procesar eventos	Marque esta casilla si sus scripts están vinculados a algún evento de aplicación. Desmárquela para evitar que los scripts reaccionen con los eventos.

Para habilitar el proyecto de scripting a nivel del proyecto:

1. Abra el proyecto.
2. En el menú **Proyecto** haga clic en **Configuración del proyecto**.
3. Haga clic en la pestaña **Scripting**.
4. Marque la casilla **Activar scripts de proyecto** y navegue hasta el archivo .asprj.

La casilla *Ejecutar macros automáticas* funciona como hemos explicado en el punto anterior.

17.2.3.1 Ejecutar macros

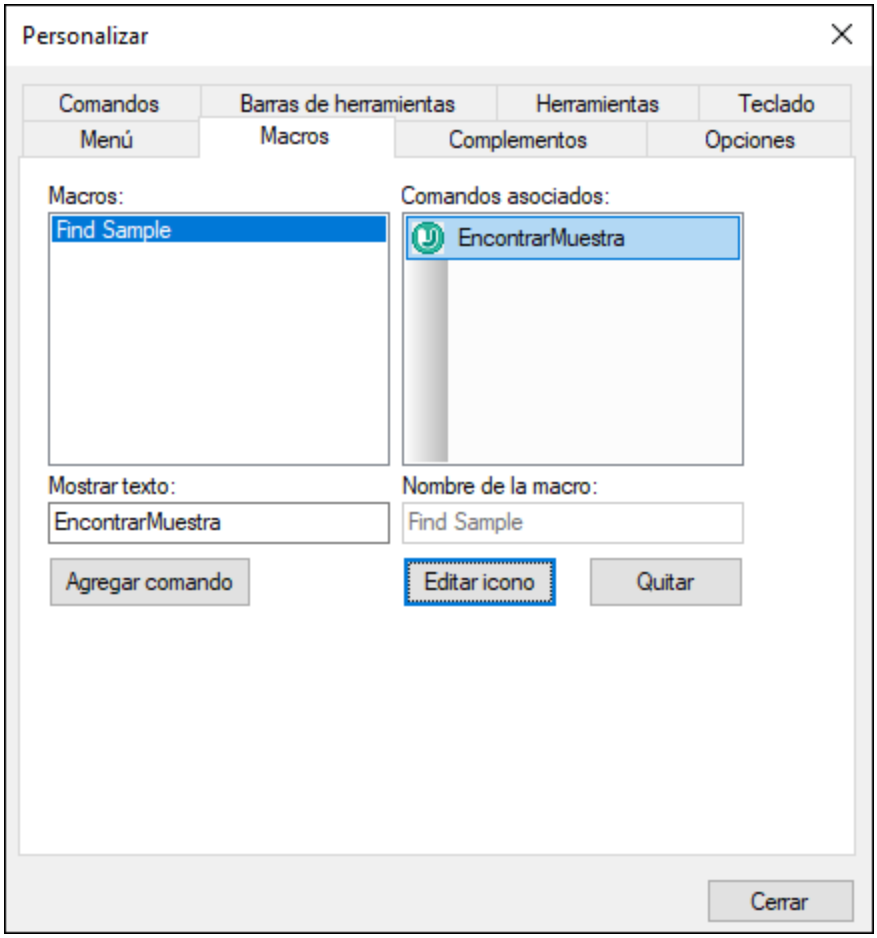
Cuando un proyecto de scripting está activo en UModel, cualquier macro que esté disponible en ese proyecto aparece en el menú **Herramientas | Macros**, por lo que puede ejecutar una macro en cualquier momento con el comando de menú correspondiente, como por ejemplo **Herramientas | Macros | <UnaMacro>**.

Las macros que se hayan configurado como automáticas se ejecutan automáticamente al iniciarse UModel, siempre que se haya habilitado este comportamiento en las opciones, tal y como se describe en el apartado Habilitar scripts y macros.

Para poder acceder más fácilmente a las macros puede crear botones para la barra de herramientas que activen esas macros:

1. En el menú **Herramientas** haga clic en **Personalizar**.

2. Haga clic en la pestaña *Macros*. En la lista aparecen todas las macros disponibles a nivel de la aplicación (en el proyecto global de scripting).
3. Haga clic en **Agregar comando**.



4. También puede hacer clic en **Editar icono** y seleccionar un icono nuevo para esa macro. Para asignar un atajo de teclado para esa macro, vaya a la pestaña *Teclado*.
5. Arrastre la macro desde el panel *Comandos asociados* hasta la barra de herramientas en la que quiere que aparezca.

Para eliminar una macro de una barra de herramientas:

1. En el menú **Herramientas** haga clic en **Personalizar**.
2. Haga clic en la pestaña *Macros*.
3. Arrastre la macro desde la barra de herramientas en la que aparece de vuelva hasta el panel *Comandos asociados*.

17.3 Complementos para entornos IDE

Si quiere puede crear sus propios complementos como bibliotecas DLL e integrarlos en UModel. Los complementos IDE (entorno de desarrollo integrado) de UModel pueden utilizarse para:

- Configurar UModel con nuevos comandos, menús, iconos, botones, etc.
- Reaccionar ante determinados eventos desde UModel.
- Ejecutar código dentro de UModel con total acceso a la API de la aplicación.
- Integrar controles ActiveX propios en UModel.

Los complementos se pueden escribir bien como aplicaciones COM (en C++) o en un lenguaje .NET adecuado para la interoperabilidad COM, como C#. Cualquier complemento de UModel debe implementar la interfaz IUModelPlugIn. También existen otros requisitos previos relacionados con la interoperabilidad .NET COM que se explican más adelante.

Encontrará varias soluciones de Visual Studio que muestran cómo acceder a las funciones de UModel con complementos personalizados aquí: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\Implementar IUModelPlugIn**.

Limitaciones

Al desarrollar un complemento para entornos IDE para UModel, evite definir la propiedad `VisualStyleState` del objeto **System.Windows.Forms.Application**, por ejemplo:

```
System.Windows.Forms.Application.VisualStyleState = VisualStyleState.NoneEnabled;
```

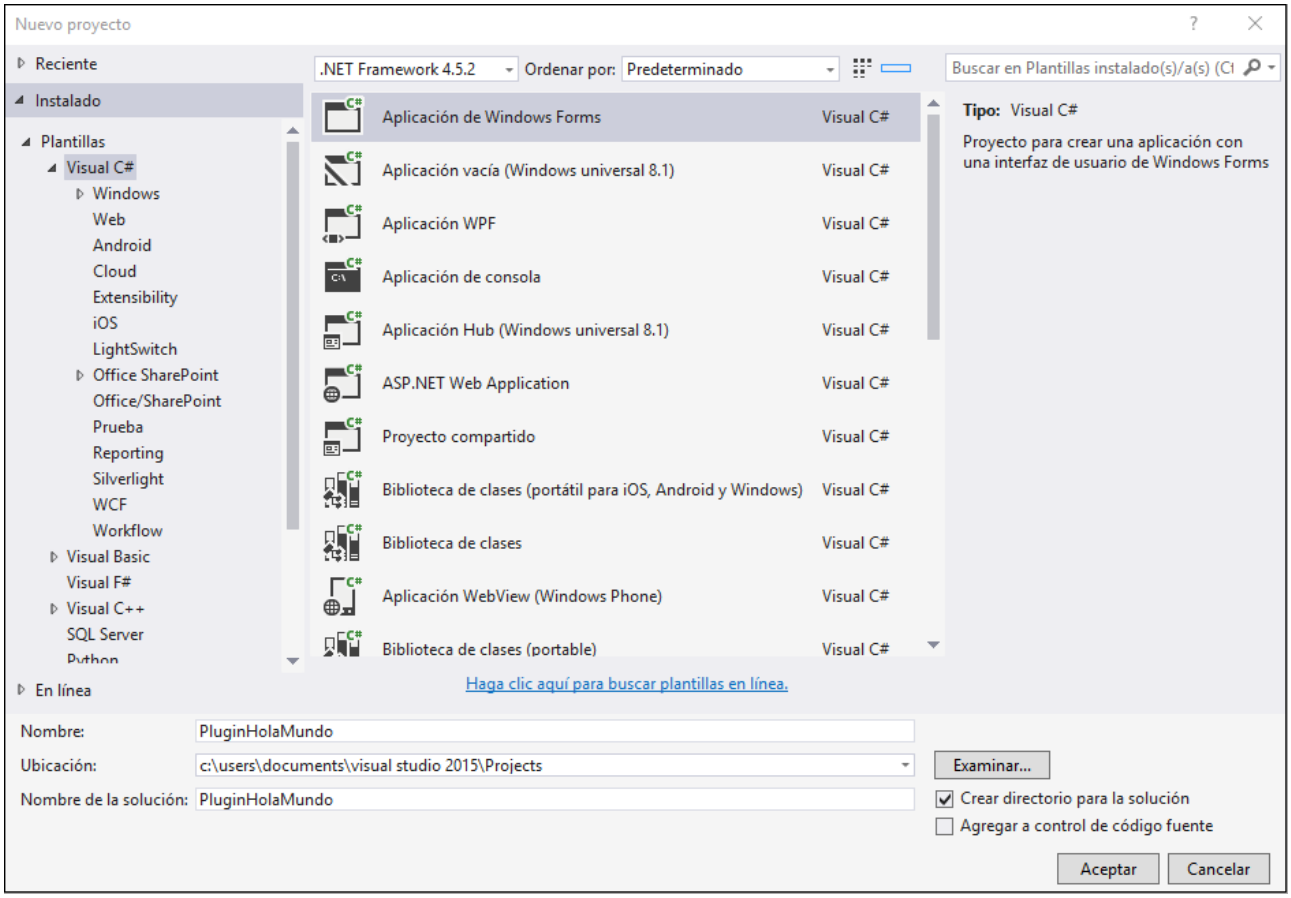
La configuración anterior evita que se cree la clase COM, por lo que bloquea los comandos de menú **Archivo | Abrir** y **Archivo | Guardar como** en UModel cuando se cargan los complementos.

17.3.1 Cómo crear un complemento IDE para UModel

Este apartado explica cómo usar C# y Visual Studio para crear un DLL para UModel.

Nota: necesita tener instalados en su equipo UModel Enterprise Edition, UModel Professional Edition, Visual Studio y Microsoft .NET Framework.

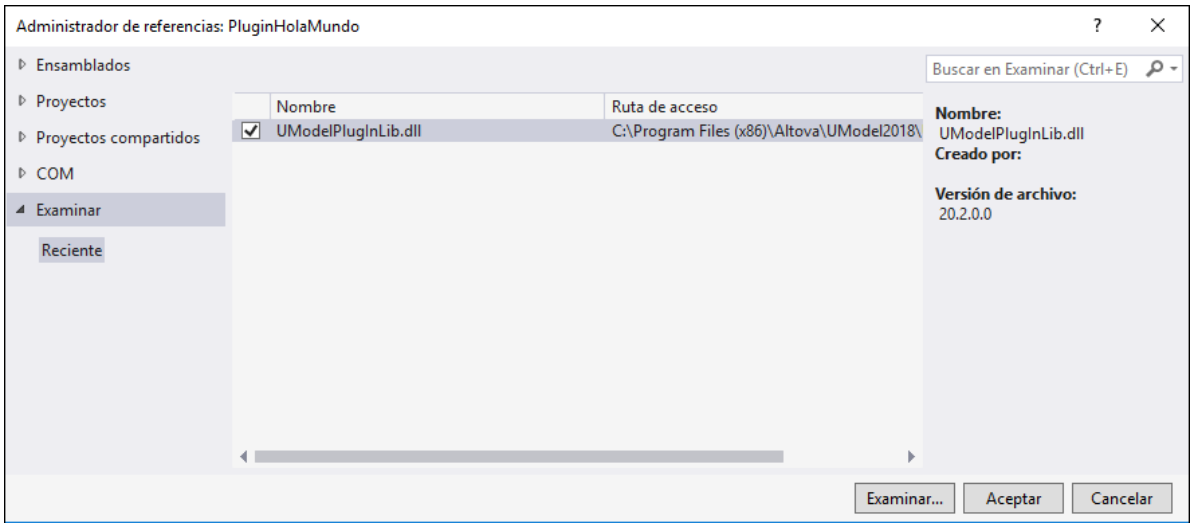
Para empezar, inicie Visual Studio y cree un nuevo proyecto de tipo "biblioteca de clases (.dll)".



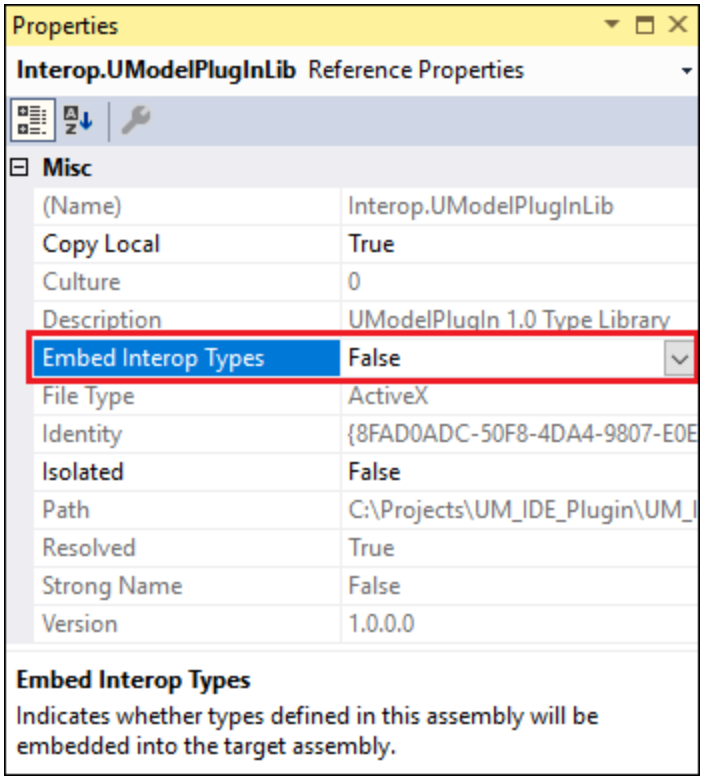
17.3.1.1 Agregar una referencia a la biblioteca del complemento de UModel

Si se añade un DLL a UModel en forma de complemento, es necesario que el DLL implemente la interfaz IUModelPlugIn. Para hacer esto posible debe añadirse primero en Visual Studio una referencia a `UModelPlugInLib.dll` como sigue:

1. Haga clic con el botón derecho en **References** en el Explorador de soluciones y seleccione el comando **Agregar referencia...** en el menú contextual.
2. En el cuadro de diálogo seleccione la pestaña *Examinar* y seleccione el archivo `UModelPlugInLib.dll` del directorio de instalación de UModel (por ejemplo, **C:\Program Files (x86)\Altova\UModel2023**).



3. En el Explorador de soluciones haga clic en la biblioteca de referencia (**UModelPlugInLib**). Encuentre la propiedad `Embed Interop Types` en la ventana Propiedades y asegúrese de que esta propiedad está definida como **`False`**.



`UModelPlugInLib.dll` es un ensamblado .NET creado a partir de `IUModelPlugIn.tlb` con Microsoft .NET Framework.

Si quiere instalar su complemento en un .NET Framework anterior a la versión 2.0 (p. ej. 1.1), debe generar su propio `UModelPlugInLib.dll` en la versión respectiva de .NET Framework.

Puede crear su propio ensamblado `UModelPlugInLib.dll` usando el importador de la biblioteca de tipos que quiera. En .NET esto se puede hacer con el importador de la biblioteca de tipos (**tlbimp.exe**) del kit de desarrollo Microsoft .NET Framework SDK:

```
tlbimp.exe IUmodelPlugIn.tlb
```

También puede crear el ensamblado con un par de claves con nombre seguro y una versión determinada:

```
tlbimp.exe IUmodelPlugIn.tlb /keyfile:UModelPlugIn.snk /asmversion:1.0.0.0
```

donde `UModelPlugIn.snk` es un archivo de clave creado en el .NET Framework SDK con sn.exe (p. ej. "`sn.exe -k UModelPlugIn.snk`") usando un comando como:

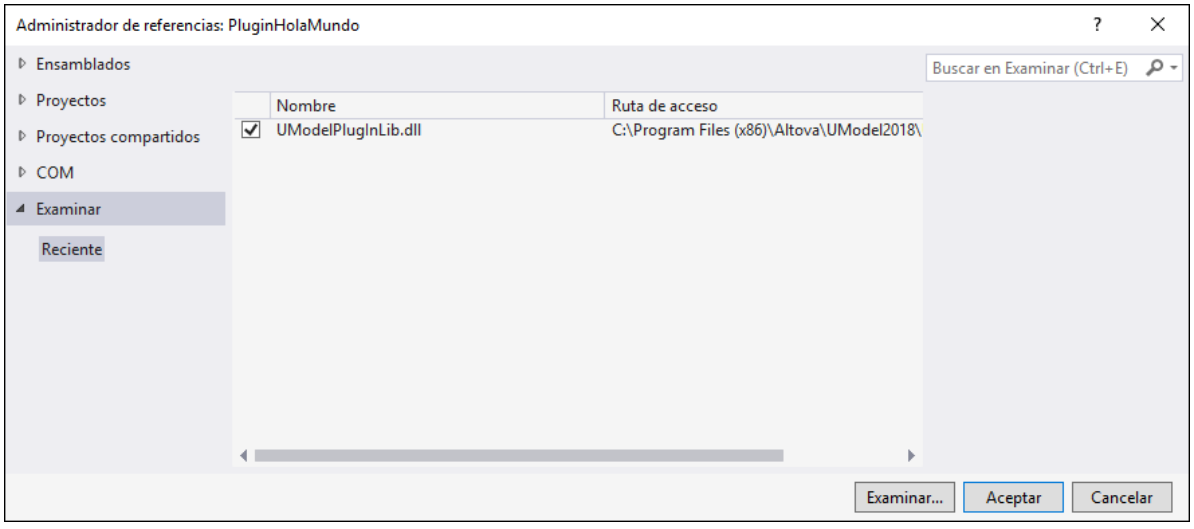
```
sn.exe -k UModelPlugIn.snk
```

Para más información sobre las herramientas incluidas en el .NET Framework, consulte la documentación de Microsoft https://docs.microsoft.com/en-us/dotnet/framework/tools/.

17.3.1.2 Agregar una referencia a la biblioteca de tipos de UModel

Para acceder a las funciones de la API de UModel desde su proyecto de Visual Studio, agregue una referencia a la biblioteca de tipos de UModel en Visual Studio:

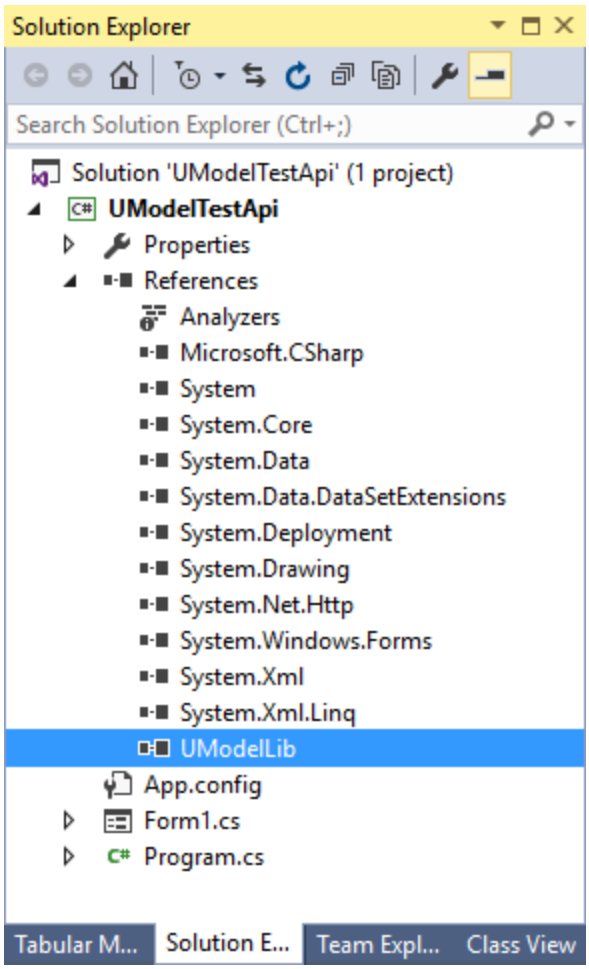
1. Cree un proyecto nuevo en Visual Studio o abra uno que ya existe.
2. En el menú Proyecto haga clic en **Agregar referencias**.
3. En la sección COM, seleccione **Biblioteca de tipos de UModel** de la lista. Si esta entrada no está disponible en la sección COM, haga clic en **Examinar** y seleccione el archivo `UModel.tlb` en la carpeta de aplicaciones de UModel.



Nota: no confunda la **Biblioteca de tipos de UModel** con la **Biblioteca de tipos del complemento de**

UModel. Esta última se puede usar para crear complementos personalizados e integrarlos en UModel (véase Agregar una referencia a la biblioteca del complemento de UModel).

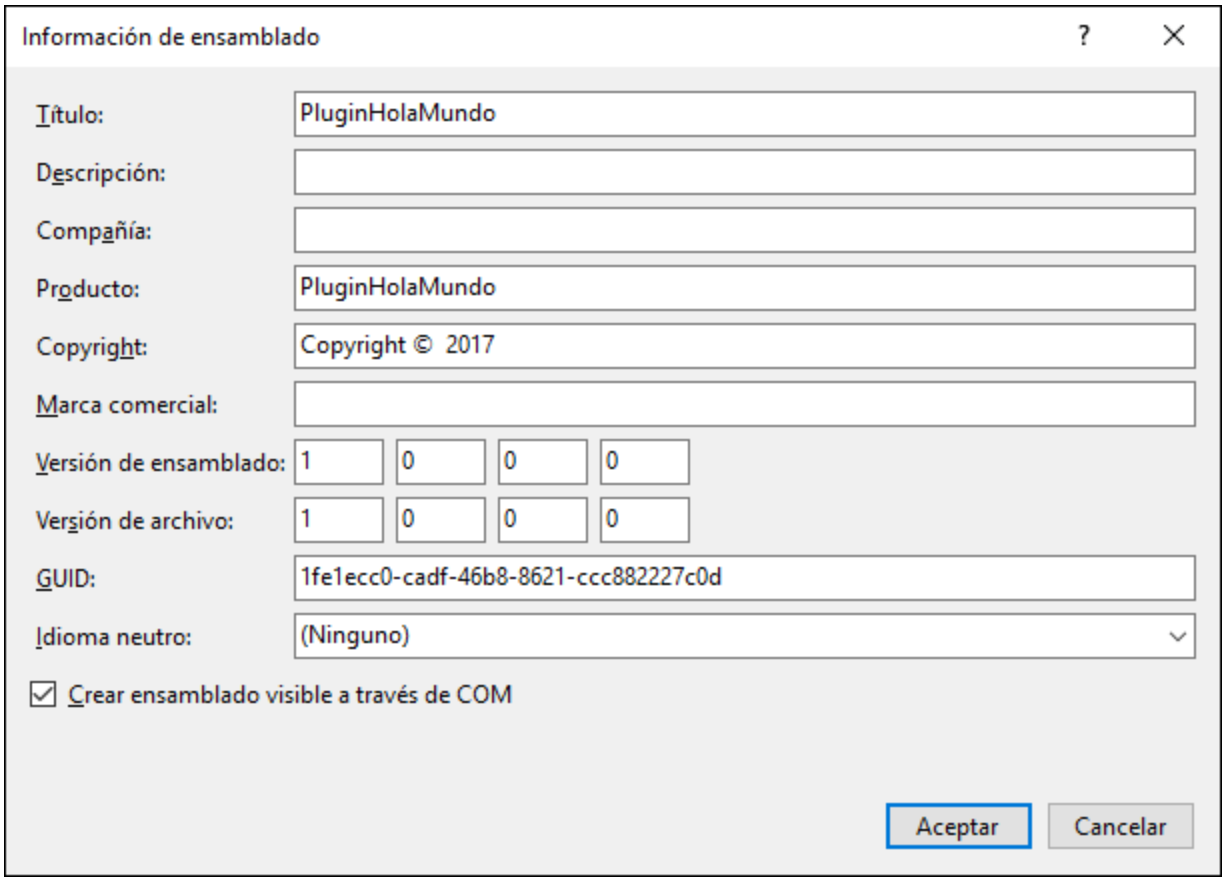
Tras seguir estos pasos, la biblioteca de tipos de UModel debería estar disponible en la lista de referencias de su solución de Visual Studio:



17.3.1.3 Crear ensamblado visible a través de COM

Para que sea posible acceder a su código desde COM debe cambiar la configuración de su compilador.

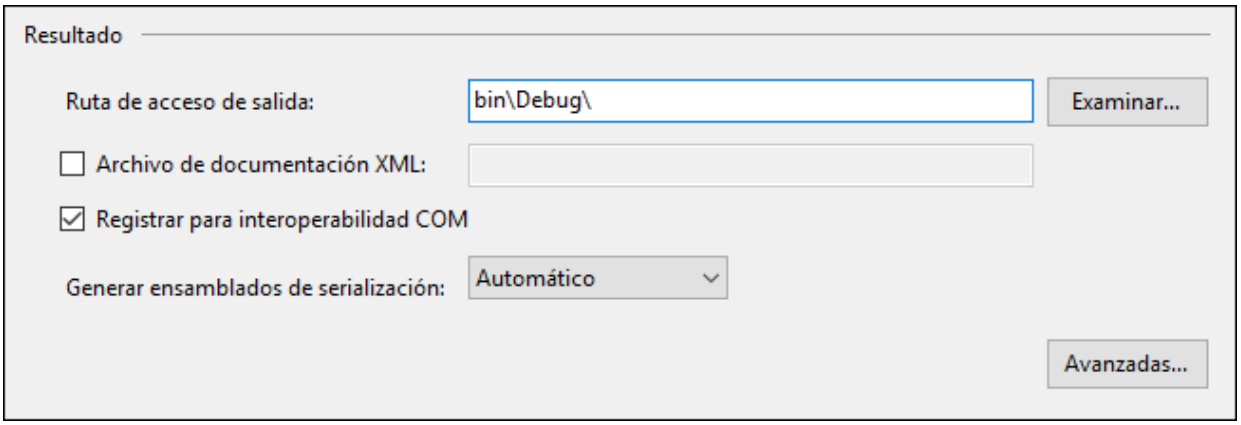
1. Haga clic con el botón derecho en su proyecto de C# y seleccione el comando **Propiedades**.
2. En la pestaña *Aplicación* haga clic en el botón **Información de ensamblado** y marque la casilla *Crear ensamblado visible a través de COM* situada en la parte inferior del cuadro de diálogo.



17.3.1.4 Exponer contenedor COM

Para exponer un contenedor al que se puede llamar desde COM que pueda interactuar con objetos COM:

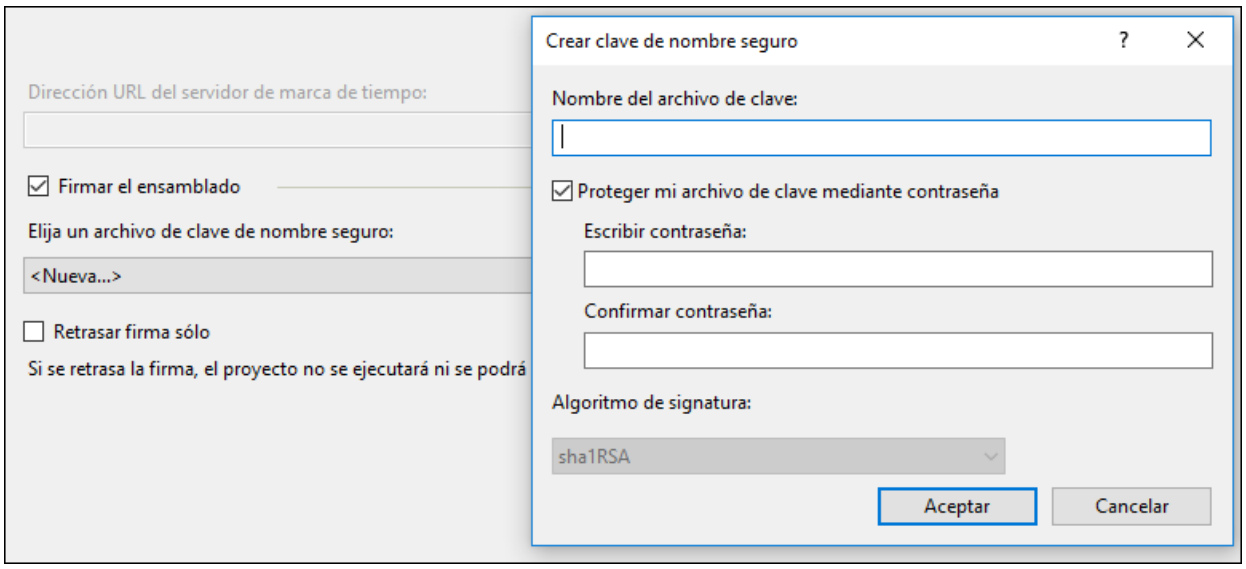
1. Haga clic con el botón derecho en su proyecto de C# y seleccione el comando **Propiedades**.
2. En la pestaña *Generar* marque la casilla *Registrar para interoperabilidad COM*.



17.3.1.5 Firmar el componente con un nombre seguro (opcional)

Para firmar el ensamblado con un par de claves con nombre seguro (p. ej. para la implementación):

1. Haga clic con el botón derecho en su proyecto de C# y seleccione el comando **Propiedades**.
2. En la pestaña *Firma* marque la casilla *Firmar el ensamblado*.
3. Elija un archivo de clave (bien con la opción **Examinar** para buscar un archivo de clave ya disponible, bien con la opción **Nuevo** para crear uno nuevo).



17.3.1.6 Implementar IUModelPlugIn

Los complementos deben implementar la interfaz IUModelPlugIn.

El código que aparece más abajo muestra una sencilla implementación de esta interfaz. Añade un elemento de menú y un separador al menú **Edición** de UModel. Al hacer clic en el elemento de menú aparecerá un cuadro de mensaje con el texto Hello, World!

Nota: como este ejemplo muestra un cuadro de mensaje, compruebe que su proyecto de C# también hace referencia a `System.Windows.Forms`.

```
using System;
using System.Collections.Generic;
using System.Text;
using UModelPlugInLib;

namespace HelloWorldPlugIn
{
    public class MyHelloWorldUModelPlugIn : IUModelPlugIn
    {
        #region IUModelPlugIn Members
```

```
        public string GetDescription()
        {
            return "HelloWorldPlugIn;HelloWorldPlugIn demonstrates a simple
implementation of an IDE plug-in for UModel";
        }

        public string GetUIModifications()
        {
            return  "<ConfigurationData>"                          +
                        "<Modifications>"                          +
                            // add "Hello World..." to Edit menu
                            "<Modification>"                       +
                                "<Action>Add</Action>"             +
                                "<UIElement type=\"MenuItem\">"    +
                                    "<ID>1</ID>"                   +
                                    "<Name>Hello world...</Name>"  +
                                    "<Info>My hello world</Info>"  +
                                    "<Place>0</Place>"             +
                                    "<MenuID>101</MenuID>"         +
                                    "<Parent>:Edit</Parent>"       +
                                "</UIElement>"                     +
                            "</Modification>"                      +
                        // add Separator to Edit menu
                            "<Modification>"                       +
                                "<Action>Add</Action>"             +
                                "<UIElement type=\"MenuItem\">"    +
                                    "<ID>0</ID>"                   +
                                    "<Place>1</Place>"             +
                                    "<MenuID>101</MenuID>"         +
                                    "<Parent>:Edit</Parent>"       +
                                "</UIElement>"                     +
                            "</Modification>"                      +
                       // finish modification description
                       "</Modifications>"                          +
                   "</ConfigurationData>";
        }

        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized
        }

        public void OnShutdown(object pUModel)
        {
            // application will shutdown; release all unused objects
        }

        public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
        {
            if (nID == 1)
                return UModelUpdateAction.UModelUpdateAction_Enable;

            return UModelUpdateAction.UModelUpdateAction_Disable;
```

```
        }

        public void OnCommand(int nID, object pUModel)
        {
            System.Windows.Forms.MessageBox.Show("Hello world!");
        }

        #endregion
    }
}
```

17.3.1.7 Crear y ejecutar el complemento de UModel

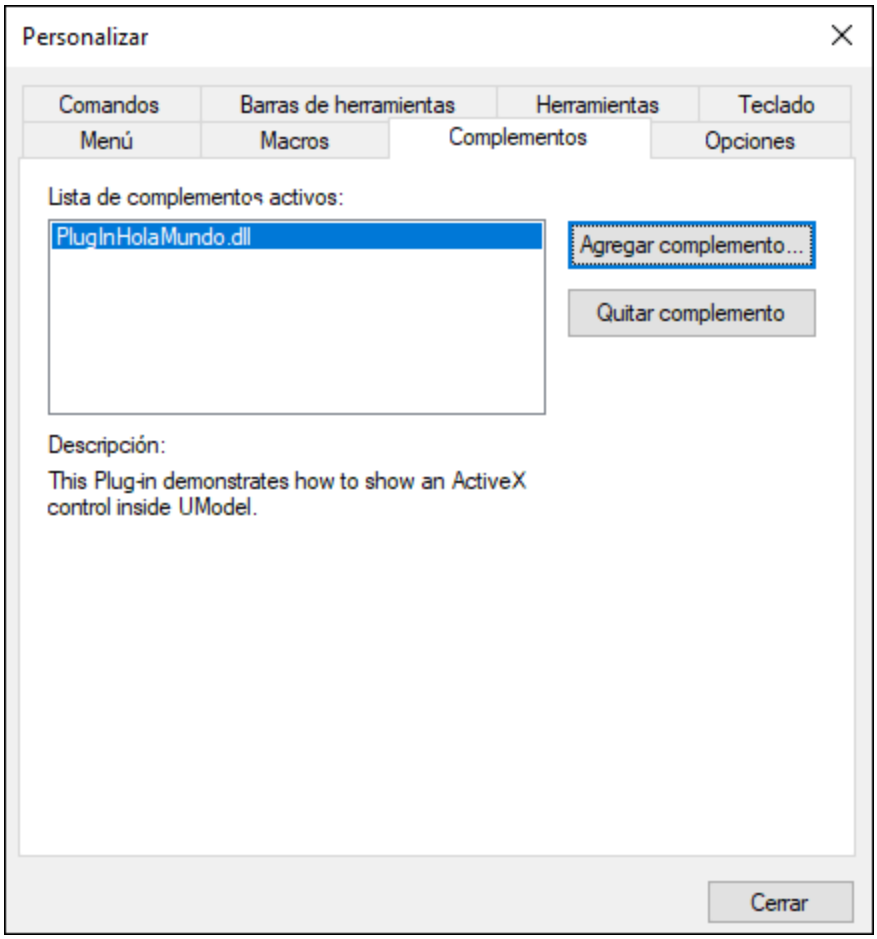
Después de seguir los pasos anteriores, cree la solución con Visual Studio (en el menú **Generar** haga clic en **Generar solución**).

Importante:

- Para compilar el complemento necesita tener acceso al registro, por lo que deberá ejecutar Visual Studio como administrador.
- Si tiene un sistema operativo de 64 bits pero está usando una versión de UModel para 32 bits, agregue la plataforma x86 al **gestor de configuraciones** de la solución y compile la muestra usando esta configuración. Para acceder al gestor de configuraciones, ejecute el comando de menú **Compilar | Administrador de configuración**.
- En el Explorador de soluciones haga clic en la biblioteca referenciada (`UModelPlugInLib`). Busque la propiedad `Embed Interop Types` en la ventana Propiedades y establézcala en `false`.

Una vez creado el proyecto de C#, puede añadir y probar el complemento de UModel:

1. Inicie UModel (si ya estaba iniciado puede reiniciarlo para que el registro lea correctamente la información del complemento).
2. Seleccione el comando **Herramientas | Personalizar**.
3. En la pestaña *Complementos* haga clic en el botón **Agregar complemento** y elija el archivo `.dll` que implementa su complemento (`PluginHelloWorld.dll`):



Nota: si encuentra un error con un texto parecido a "No se puede encontrar una implementación de la interfaz del complemento de UModel en la biblioteca de tipos", asegúrese de que la propiedad `Embed Interop Types` está establecida en `False` para la biblioteca **UModelPlugInLib**, como se describe en Agregar una referencia a la biblioteca del complemento de UModel.

El menú **Edición** de UModelo ahora contiene un nuevo comando de menú llamado **Hello world**. Ejecute este comando para mostrar un cuadro de diálogo con el mensaje "¡Hola, mundo!".

17.3.2 Implementación de complementos para IDE en UModel

En un PC de desarrollo el registro con COM se lleva a cabo al crear el complemento con Visual Studio. Normalmente no se requiere un registro manual. Si quiere implementar un complemento IDE de UModel en un sistema de destino cliente, el equipo de destino debe cumplir los siguientes requisitos:

- UModel Professional y Enterprise Edition y
- Microsoft .NET Framework (su el complemento está escrito en .NET).

En un equipo de implementación, el complemento se puede registrar manualmente o al implementarlo. Para ver un ejemplo de un proyecto de implementación, vaya a "Set Styles" Sample..

Para registrar manualmente un complemento IDE de UModel:

1. En el menú **Herramientas** de UModel haga clic en **Personalizar**.
2. Haga clic en la pestaña *Complementos*.
3. Haga clic en **Agregar complemento** y busque el archivo `.dll` del complemento.

Puede comprobar si un complemento de UModel está registrado ejecutando `regedit.exe` en la línea de comando. UModel tiene la siguiente clave de registro para todos los complementos registrados:

```
HKEY_CURRENT_USER\Software\Altova\UModel\PlugIns
```

Todos los valores de esta clave se tratan como referencias a los complementos registrados y deben tener el siguiente formato:

Nombre del valor:	`ProgID` of the plug-in
Tipo de valor:	must be `REG_SZ`
Datos del valor:	`CLSID` of the component

Cada vez que de inicia UModel se escanean los valores de la clave "Complementos" y se cargan los complementos registrados. Si surge algún problema, compruebe que el CLSID de su complemento está correctamente registrado en la clave "Complementos". Si no es el caso, es probable que deba cambiar el nombre de su complemento DLL.

Nota: al implementar su complemento IDE de UModel en versiones de .NET Framework previas a la 2.0, el archivo `.dll` del complemento debe estar instalado en el mismo directorio que UModel.exe o estar firmado con una clave de nombre seguro y registrado en el caché global de ensamblados.

Si necesita ejecutar varias tareas relacionadas con ensamblados manualmente, tenga en cuenta que las siguientes herramientas están incluidas en el .NET Framework SDK:

- Herramienta de registro de ensamblados (`regasm.exe`). Puede usar esta herramienta para registrar o eliminar del registro ensamblados de COM. Por ejemplo, para registrar manualmente **UModelPlugLib.dll**, use:

  ```
  regasm.exe UModelPlugInLib.dll /codebase
  ```

- Herramienta de nombre seguro (`sn.exe`). Puede usar esta herramienta para firmar un ensamblado con una clave segura, por ejemplo:

  ```
  sn.exe -k MyKeyFile.snk
  ```

  La clave también puede generarse desde Visual Studio (véase Firmar el componente con un nombre seguro (opcional)).

- Herramienta Caché global de ensamblados (`gacutil.exe`). Puede usar esta herramienta para agregar o eliminar ensamblados en el Caché global de ensamblados. Por ejemplo, apara agregar `MyPlugin.dll` al Caché global de ensamblados, use:

  ```
  gacutil.exe /i MyPlugin.dll
  ```

Para más información acerca de las herramientas incluidas en .NET Framework, consulte la documentación de Microsoft: https://docs.microsoft.com/en-us/dotnet/framework/tools/.

17.3.3 XML de configuración

Con el complemento puede cambiar la interfaz del usuario (IU) de UModel. Para ello es necesario describir cada modificación con una secuencia de datos XML. Los datos XML de configuración se pasan a UModel con el método `GetUIModifications` de la Interfaz IUModelPlugIn.

El archivo XML que contiene las modificaciones de la IU para el complemento debe seguir esta estructura:

```
<ConfigurationData>
   <ImageFile>path To image file</ImageFile>
   <Modifications>
      <Modification>
         ...
      </Modification>
      ...
   </Modifications>
</ConfigurationData>
```

Puede definir iconos o botones de barra de herramientas para los nuevos elementos de menú que el complemento añade a la IU de UModel. La ruta de acceso del archivo que contiene las imágenes se establece con el elemento `ImageFile`. Las imágenes deben tener cada una un tamaño de 16 x 16 píxels. Las referencias de imagen deben hacerse de izquierda a derecha en una sola línea (<ImageFile>...). El valor de índice de la imagen más a la derecha es cero.

El elemento `Modifications` puede tener un número ilimitado de secundarios `Modification`. Cada elemento `Modification` define un cambio concreto en la IU estándar de UModel. También se pueden eliminar elementos de la IU de UModel.

Estructura de los elementos Modification

Todos los elementos `Modification` están formados por estos dos elementos secundarios:

```
<Modification>
   <Action>Type of action</Action>
   <UIElement Type="type of UI element">
   </UIElement>
</Modification>
```

Estos son los valores válidos para el elemento `Action`:

Add - para agregar el elemento de interfaz gráfica en UModel.
Hide - para ocultar el elemento de interfaz gráfica en UModel.
Remove - para quitar el elemento de interfaz gráfica del cuadro de lista *Comandos* del cuadro de diálogo "Personalizar" de UModel.

Estos valores del elemento `Action` se pueden combinar (p. ej. "Hide Remove").

El elemento `UIElement` describe elementos nuevos o que ya existen de la IU de UModel. Estos son los elementos posibles: barras de herramientas, botones, menús y elementos de menú. El atributo `type` define qué elemento de la IU describe el elemento XML.

Elementos secundarios de UIElement

Los elementos `ID` y `Name` son válidos para todos los tipos de fragmentos de `UIElement`. Sin embargo, se puede ignorar uno de los valores para un tipo de `UIElement` determinado (p. ej. Name se ignora en los separadores).

```
<ID></ID>
<Name></Name>
```

Si `UIElement` describe un elemento que ya existe de la IU, el valor de `ID` viene ya predefinido por UModel. Lo normal es que estos valores no sean públicos. Si, por el contrario, el fragmento XML describe una parte nueva de la IU, entonces el valor de `ID` es aleatorio y el valor debe ser menor que 1000.

El elemento `Name` establece el valor textual. Los elementos que ya existen de la IU se pueden identificar por el nombre (p. ej. menús y elementos de menú con submenús asociados). Para elementos nuevos de la IU, el elemento `Name` establece el título (p. ej. el título de una barra de herramientas o el texto para un elemento de menú).

Barras de herramientas y menús

Para poder definir una barra de herramientas es necesario especificar el identificador o el nombre de la barra de herramientas. Puede referirse a barras de herramientas que ya existen con su nombre (o con su identificador, si lo conoce). Para crear una barra de herramientas **nueva**, deben definirse ambos valores. El atributo **type** debe ser igual a "ToolBar".

```
<UIElement Type="ToolBar">
   <ID>1</ID>
   <Name>Styles</Name>
</UIElement>
```

Para referirse a un menú de UModel es necesario usar dos parámetros:

- El identificador de la barra de menú que incluye el menú. (El identificador de la barra de menú principal de UModel es 101).
- El nombre del menú. Los menús no tienen asociado ningún identificador. Por ejemplo, a continuación puede ver cómo se define el menú **Edición** de la barra de menú:

```
<UIElement Type="Menu">
   <ID>101</ID>
   <Name>Edit</Name>
</UIElement>
```

Si quiere crear un elemento nuevo, debe añadir otro elemento: el elemento `Place`, que define la posición del nuevo menú en la barra de menú:

```
<UIElement Type="Menu">
   <ID>101</ID>
   <Name>PlugIn Menu</Name>
   <Place>12</Place>
</UIElement>
```

Si asigna el valor -1 al elemento `Place`, el nuevo botón o elemento de menú se coloca al final del menú o de la barra de herramientas.

Comandos

Si añade un comando nuevo (mediante un botón o un elemento de menú nuevo), el fragmento `UIElement` puede incluir estos subelementos:

```
<Info></Info>
<ImageID></ImageID>
```

El elemento `Info` contiene una breve cadena de texto que aparece en la barra de estado cuando se pasa el puntero del ratón por encima del comando (del botón o del elemento de menú). El elemento `ImageID` define el índice del icono en el archivo de imagen externo. Recuerde que todos los iconos se almacenan en un solo archivo de imagen.

Para definir un botón de barra de herramientas debe crear un fragmento `UIElement` con esta estructura:

```
<UIElement Type="ToolBarItem">
   <!--don't reuse local IDs even the commands do the same-->
   <ID>6</ID>
   <Name>Fill red</Name>
   <!--Set Place To -1 If this is the first button to be inserted-->
   <Place>-1</Place>
   <ImageID>0</ImageID>
   <ToolBarID>1</ToolBarID>
   <!--instead of the toolbar ID the toolbar name could be used-->
   <ToolBarName>Styles</ToolBarName>
</UIElement>
```

También puede declarar un botón de barra de herramientas con estos elementos: `Place`, `ToolBarID` y `ToolBarName`. Los elementos `ToolBarID` y `ToolBarName` se utilizan para identificar la barra de herramientas que contiene el botón nuevo o que ya existe. El valor textual del elemento `ToolBarName` distingue entre mayúsculas y minúsculas. El atributo **type** debe ser igual a `ToolBarItem`.

Para definir un elemento de menú, además de los elementos estándar utilizados para declarar un comando, están disponibles los elementos `MenuID`, `Place` y `Parent`. El elemento `MenuID` debe ser 101. Para más información consulte el subapartado anterior sobre barras de herramientas y menús.

El elemento `Parent` sirve para identificar el **menú** donde debe insertarse la entrada de menú nueva. Como los elementos de los submenús no tienen un identificador de Windows único, hay que encontrar otra manera de identificar al submenú.

El valor del elemento `Parent` es una ruta de acceso al elemento de menú.

El valor textual del elemento `Parent` debe equivaler al **nombre del menú primario** del submenú (y el nombre del submenú debe estar separado por dos puntos). Si el menú no tiene un primario (porque no es un submenú) añada dos puntos (:) al principio del nombre. El atributo **type** debe tener el valor `MenuItem`.

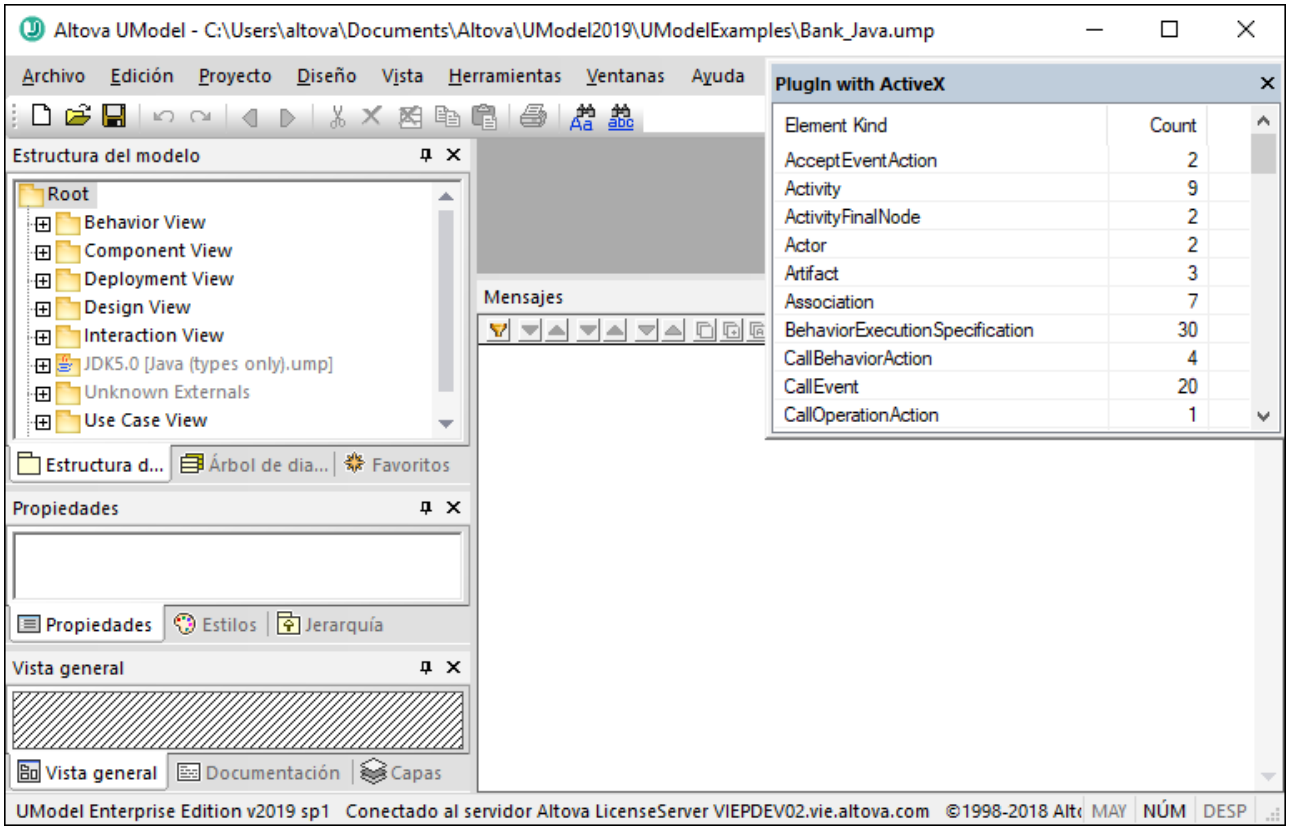
Por ejemplo, este fragmento `UIElement` define un elemento de menú:

```
<UIElement Type="MenuItem">
   <!--the following element is a Local command ID-->
   <ID>3</ID>
   <Name>Fill red</Name>
   <Place>-1</Place>
   <MenuID>101</MenuID>
   <Parent>:PlugIn Menu1</Parent>
   <ImageID>0</ImageID>
</UIElement>
```

UModel permite añadir separadores de barra de herramientas y menús si el valor del elemento `ID` es 0.

17.3.4 Complementos como controles ActiveX

Para que funcione como un control ActiveX, el complemento debe implementar la interfaz `IOleControl` (C++) o derivarse de `System.Windows.Forms.UserControl` (C#, VB.NET). Estos controles aparecen como una ventana nueva en la interfaz gráfica del usuario y también como comando de menú en el menú **Vista**.



El código fuente del complemento de la imagen anterior está disponible en **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\IDEPlugIn\StatisticsActiveX\StatisticsActiveX.cs** (*véase el ejemplo "Estadísticas"*).

17.3.5 Interfaz IUModelPlugIn

Si se añade un archivo DLL como complemento en UModel, para que el DLL se reconozca como complemento es necesario que registre un componente de COM que responda a una interfaz `IUModelPlugIn`. La interfaz `IUModelPlugIn` contiene los siguientes métodos, que deben estar implementados por un complemento cliente.

- `OnInitalize`
- `OnRunning`
- `OnShutdown`
- `GetUIModifications`
- `GetDescription`
- `OnCommand`
- `OnUpdateCommand`

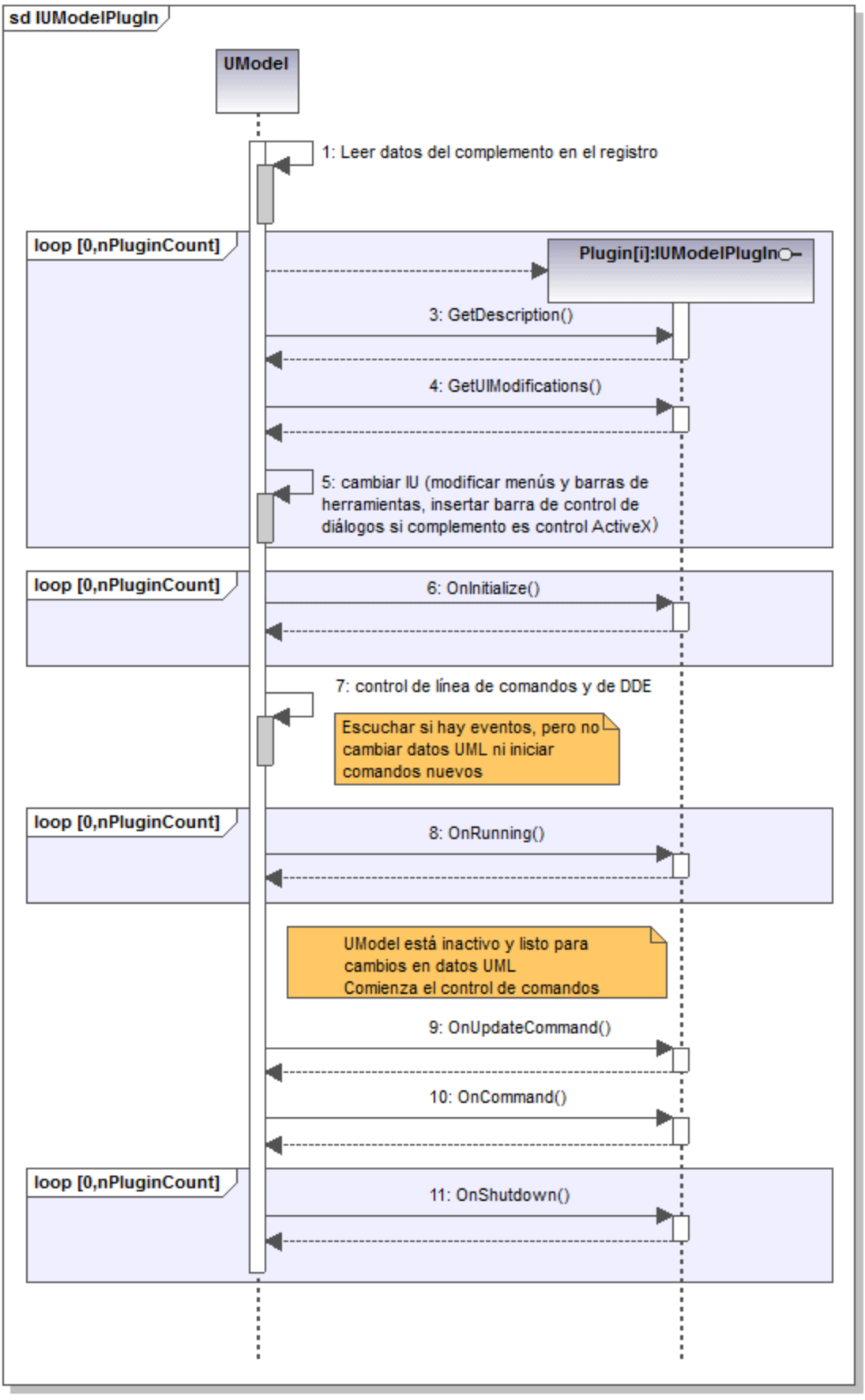
Declaración del método	Uso
`OnInitialize(pUModel as IDispatch)`	Al método `OnInitialize` de la implementación de interfaz se le llama cuando se inicializa el complemento y **antes** de que se

Declaración del método	Uso
	procesen los comandos DDE o de procesamiento por lotes. Puede adjuntar notificadores y escuchar si hay eventos de UModel, pero no debería iniciar comandos / cambios nuevos hasta que se invoque el método `OnRunning`. `pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel.
`OnRunning(pUModel as IDispatch)`	Al método `OnRunning` de la implementación de interfaz se le llama cuando se inicializa el complemento y **después** de que se procesen los comandos DDE o de procesamiento por lotes. Ahora la aplicación está completamente inicializada y puede iniciar nuevos comandos / cambios y modificar los datos UML. `pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel.
`OnShutdown(pUModel as IDispatch)`	Al método `OnShutdown` de la implementación de interfaz se le llama **inmediatamente antes** de que se descargue el complemento (por ejemplo cuando se va a cerrar la aplicación). `pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel.
`GetUIModifications() as String`	Al método `GetUIModifications()` se le llama durante la inicialización del complemento para obtener los datos XML de configuración que definen los cambios en la interfaz de usuario de UModel. Al método se le llama cuando se carga el complemento por primera vez y cada vez que se inicia UModel. Para más información sobre cómo cambiar la interfaz de usuario consulte el apartado XML de configuración.
`GetDescription() as String`	`GetDescription()` se utiliza para definir la cadena de descripción para las entradas del complemento que son visibles en el cuadro de diálogo "Personalizar".
`OnCommand(nID as long, pUModel as IDispatch)`	Al método `OnCommand()` de la implementación de interfaz se le llama cada vez que se procesa un comando añadido por el complemento (ya sea un comando de menú o un botón de una barra de herramientas). `nID` almacena el identificador del comando definido por el elemento `ID` del correspondiente `UIElement`.

Declaración del método	Uso
	`pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel.
`OnUpdateCommand(nID as long, pUModel as IDispatch) as UModelUpdateAction`	Al comando `OnUpdateCommand()` se le llama cada vez que se necesita establecer el estado visible de un botón u opción de menú. `nID` almacena el identificador del comando definido por el elemento `ID` del correspondiente `UIElement`. `pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel. Los valores devueltos posibles (según lo definido en UModelUpdateAction) para establecer el estado de actualización son: `UModelUpdateAction_Enable` = 1 `UModelUpdateAction_Disable` = 2 `UModelUpdateAction_Check` = 4 `UModelUpdateAction_Uncheck` = 8 Puede combinar varios valores usando el operador OR bit a bit (p. ej. `UModelUpdateAction_Enable | UModelUpdateAction_Check`).

Para ver un ejemplo muy sencillo de implementación de interfaz, consulte Implement IUModelPlugIn Interface. Encontrará más ejemplos de implementación (como soluciones de Visual Studio) en: **C:\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamples\IDEPlugIn**.

El diagrama de secuencia que aparece a continuación explica cómo interactúa UModel con IUModelPlugIn:


sd IUModelPlugIn
UModel
1: Leer datos del complemento en el registro
loop [0,nPluginCount]
Plugin[i]:IUModelPlugIn
3: GetDescription()
4: GetUIModifications()
5: cambiar IU (modificar menús y barras de herramientas, insertar barra de control de diálogos si complemento es control ActiveX)
loop [0,nPluginCount]
6: OnInitialize()
7: control de línea de comandos y de DDE
Escuchar si hay eventos, pero no cambiar datos UML ni iniciar comandos nuevos
loop [0,nPluginCount]
8: OnRunning()
UModel está inactivo y listo para cambios en datos UML
Comienza el control de comandos
9: OnUpdateCommand()
10: OnCommand()
loop [0,nPluginCount]
11: OnShutdown()

17.4 The UModel API

The COM-based API of UModel enables clients to access the functionality of UModel from a custom code or application and automate a wide range of tasks.

The UModel API follows the common specifications for automation servers as set out by Microsoft. UModel is automatically registered as a COM server object during installation. Once the COM server object is registered, you can invoke it from within applications and scripting languages that have programming support for COM calls. This makes it possible to access the UModel API not only from development environments using .NET, C++ and Visual Basic, but also from scripting languages like JScript and VBScript. In Java, the UModel API is available through Java-COM bridge libraries.

Note: If you use the UModel API to create an application that you intend to distribute to other clients, UModel must be installed on each client computer. Also, your custom integration code must be deployed to (or your application installed on) each client computer.

17.4.1 Accessing the API

To access the COM API, a new instance of the `Application` object must be created in your application (or script). Once you have created the application object, you can start using the functionality of UModel. You will generally either open an existing document, create a new one, or access the active document (`IDocument`). `IDocument` corresponds to a UModel project and can be used to include sub-projects, generate documentation, synchronize model and code, while also giving access to the main UMLData objects, see also Object Model.

Note: When implementing a UModel IDE plugin, there is no need to create an instance of the application object, because UModel is already running and the current instance of the application object is provided by `IApplication` as parameter for all important methods of `IUModelPlugIn`.

Prerequisites

To make the UModel COM object available in your Visual Studio project, add a reference to the UModel type library (.tlb) file, see How to Reference the UModel Type Library. A sample UModel API client in C# is available at: **C:\Users\<username>\Documents\Altova\UModel2023\UModelExamples\API\C#**.

In Java, the UModel API is available through Java-COM bridge libraries. These libraries are available in the UModel installation folder: **C:\Program Files (x86)\Altova\UModel2023\JavaAPI** (note this path is valid when 32-bit UModel runs on 64-bit Windows, otherwise adjust the path accordingly).

- `AltovaAutomation.dll`: a JNI wrapper for Altova automation servers
- `AltovaAutomation.jar`: Java classes to access Altova automation servers
- `UModelAPI.jar`: Java classes that wrap the UModel automation interface
- `UModelAPI_JavaDoc.zip`: a `Javadoc` file containing help documentation for the Java API

Note: In order to use the Java API, the .dll and .jar files must be on the Java `classpath`.

A sample UModel API client in Java is available at: **C:\Users\<username>\Documents\Altova\UModel2023\UModelExamples\API\Java**.

In scripting languages such as JScript or VBScript, the UModel COM object is accessible through the Microsoft Windows Script Host (see https://msdn.microsoft.com/en-us/library/9bbdkx3k.aspx). Such scripts

can be written with a text editor, and do not need compilation, since they are executed by the Windows Script Host packaged with Windows. (To check that the Windows Script Host is running, type `wscript.exe /?` at the command prompt). Several JScript example files that call the UModel API are available at: **C:\Users\<username>\Documents\Altova\UModel2023\UModelExamples\API\JScript**.

Nota: en la versión de 32 bits de UModel, el nombre registrado o identificador programático (ProgId) del objeto COM es `UModel.Application`. Para la versión de 64 bits de UModel, el nombre es `UModel_64.Application`. Sin embargo, debe tener en cuenta que el programa que realiza las llamadas accede a las entradas de registro CLASSES de su propio subárbol o grupo (de 32 o 64 bits). Por tanto, si ejecuta scripts usando la línea de comandos estándar y el explorador de Windows en una instancia Windows de 64 bits, el programa accederá a las entradas de registro de 64 bits, que apuntan a la versión de 64 bits de UModel. Por eso, si tiene instaladas tanto la versión de 32 bits como la versión de 64 bits de UModel necesitará dar un rodeo para llamar a la versión de 32 bits. Por ejemplo, si el programa que realiza las llamadas es Windows Scripting Host, haga lo siguiente:

```
C:\Users\...\Documents\Altova\UModel2023\UModelExamples\API\JScript\Start.js
```

Guidelines

The following guidelines should be considered in your client code:

- Do not hold references to objects in memory longer than you need them. If a user interacts between two calls of your client, then there is no guarantee that these references are still valid.
- Be aware that if your client code crashes, instances of UModel may still remain in the system. For details on how to handle error messages, see Error handling.
- Free references explicitly, if using languages such as C or C++. In C# and Visual Basic, `GC.Collect()` can be used to force garbage collection.
- UModel API collections are zero-based. For example, the statement `myPackage.InsertPackagedElementAt(0, "Interface");` will insert a new interface as first child of the package.

17.4.2 Object Model

The starting point for every application which uses the UModel API is the `IApplication` interface. The application object consists of the following main parts (each indentation level indicates a child–parent relationship with the level directly above):

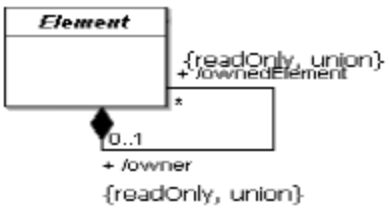
```
IApplication
   IDocument
      IDiagramWindows
         IDiagramWindow
      IFocusedUMLDataNotifier
      ITransactionNotifier
      IUMLData (and all other derived UML data interfaces)
         IUMLDataList
   IDialogs
      IExportXMIFileDlg
      IGenerateDocumentationDlg
         IKindSelectionList
            IKindSelection
```

IGenerateSequenceDiagramDlg
IGenerateStateMachineCodeDlg
IImportBinaryTypesDlg
IImportBinaryTypeEntries
IImportBinaryTypeEntry
IImportDatabaseDlg
IImportSourceDirectoryDlg
IImportSourceProjectDlg
IImportXMLSchemaDirectoryDlg
IImportXMLSchemaFileDlg
IIncludeSubprojectDlg
IModelTransformationDlg
IModelTransformationTypeMappings
IModelTransformationTypeMapping
IProjectSettingsDlg
ISaveAllDiagramsAsImagesDlg
ISynchronizationSettingsDlg
IMatchRenamedDlg
IMatchRenamedEntries
IMatchRenamedEntry
IURLDlg
ILocalOptions
ILocalOptionsCodeEngineering
ILocalOptionsDiagramEditing
ICollectionTemplates
ICollectionTemplate
ILocalOptionsEditing
ILocalOptionsFile
ILocalOptionsView

In addition, several Enumerations and Events are part of the model.

17.4.2.1 Object Model UMLData

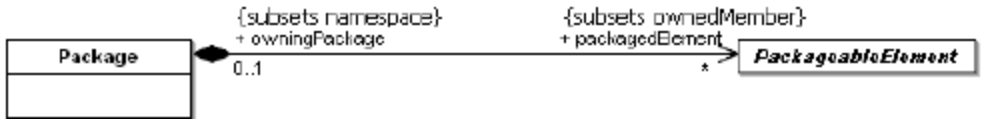
The starting point to access UML elements is the root package (IUMLPackage), which is a property of the IDocument interface. All children of the root package are a subtype of IUMLElement and are stored as defined by the OMG in the UML Superstructure Specification (http://www.uml.org). Specifically, the UML Superstructure Specification defines the following relationship for UML Element:



Which means that every UML element can have a list of owned elements, and every UML element (apart from the root-package) has an owner.

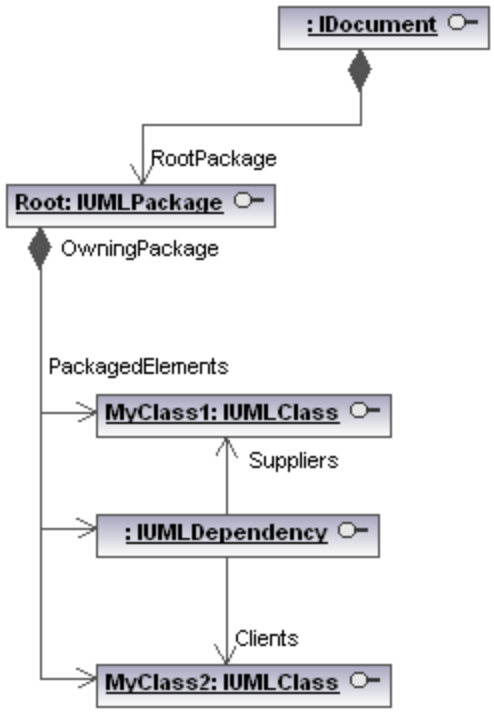
In the UModel API, an UML element is mapped to IUMLElement having the properties `OwnedElement` and `Owner`. Since these relationships are "read only" in the UML specification, both properties cannot be modified in the UModel API.

The UML Superstructure Specification also defines the following relationship between `Package` and `PackageableElement`:



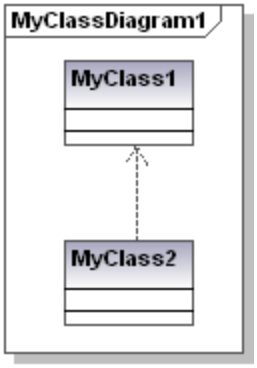
This is mapped to IUMLPackageableElement having a property `OwningPackage` and an IUMLPackage, which not only has a property `PackagedElements`, but also a method `InsertPackagedElementAt` to insert new IUMLPackageableElements (at the specified position). The method `EraseFromModel` deletes any IUMLElement (and all its children) from the model.

The sample below shows the mapping of a project which consists of two classes (IUMLClass) with a dependency (IUMLDependency) between them:



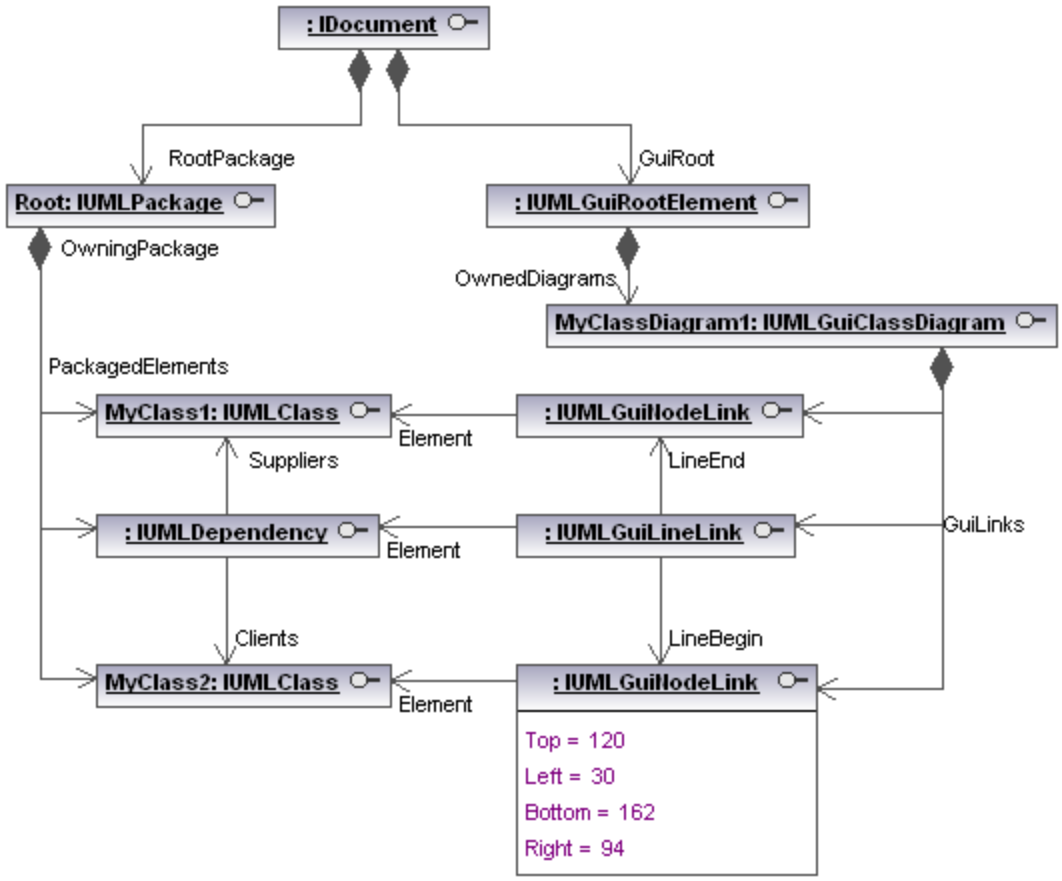
This structure is independent of whether these elements are shown on any diagram or not.

The representation of graphical objects on diagrams (as shown in the image below) is stored in a second structure with elements of kind IUMLGuiElement (also see Graphical Objects).



The starting point to access UML GUI elements is the GuiRoot (IUMLGuiRootElement), which is a property of the IDocument interface.

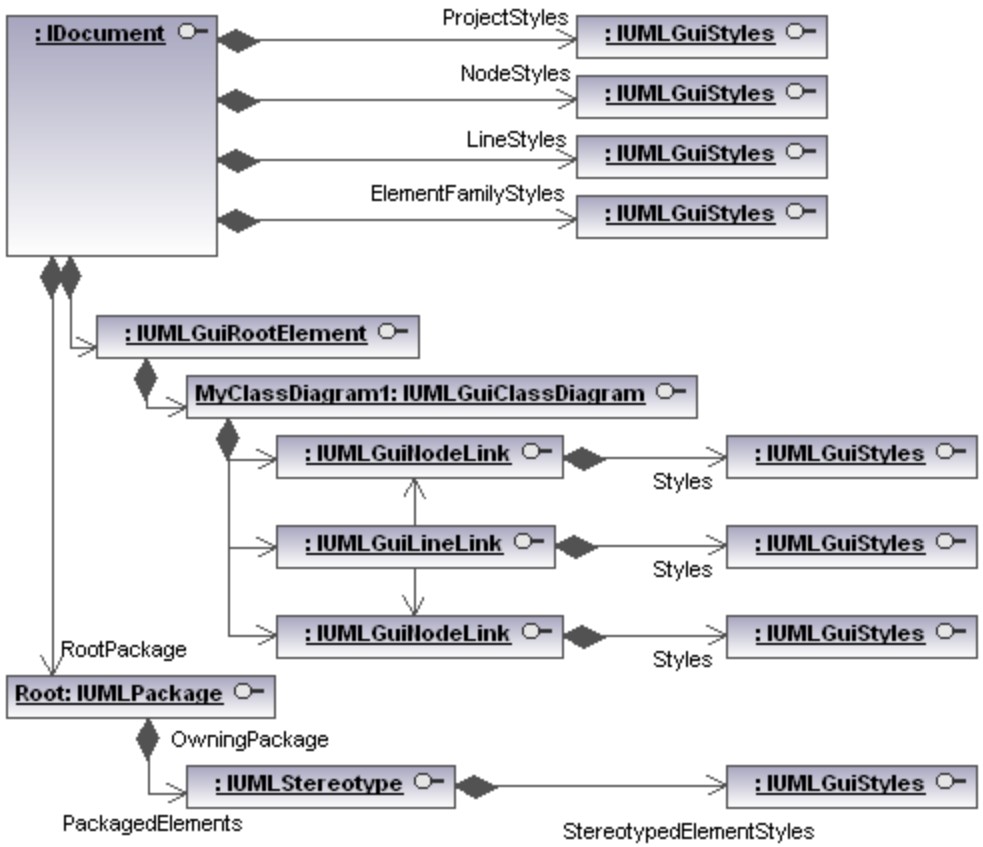
Lines are handled by IUMLGuiLineLinks, most other objects (like classes, interfaces, packages,...) by IUMLGuiNodeLinks.



17.4.2.2 Object Model UMLData Styles

UModel has various styles allowing you to adapt the diagram appearance (i.e. font size, weight, color, visibility options,...).

The following picture shows how the different styles (IUMLGuiStyles) can be accessed using the UModel API:



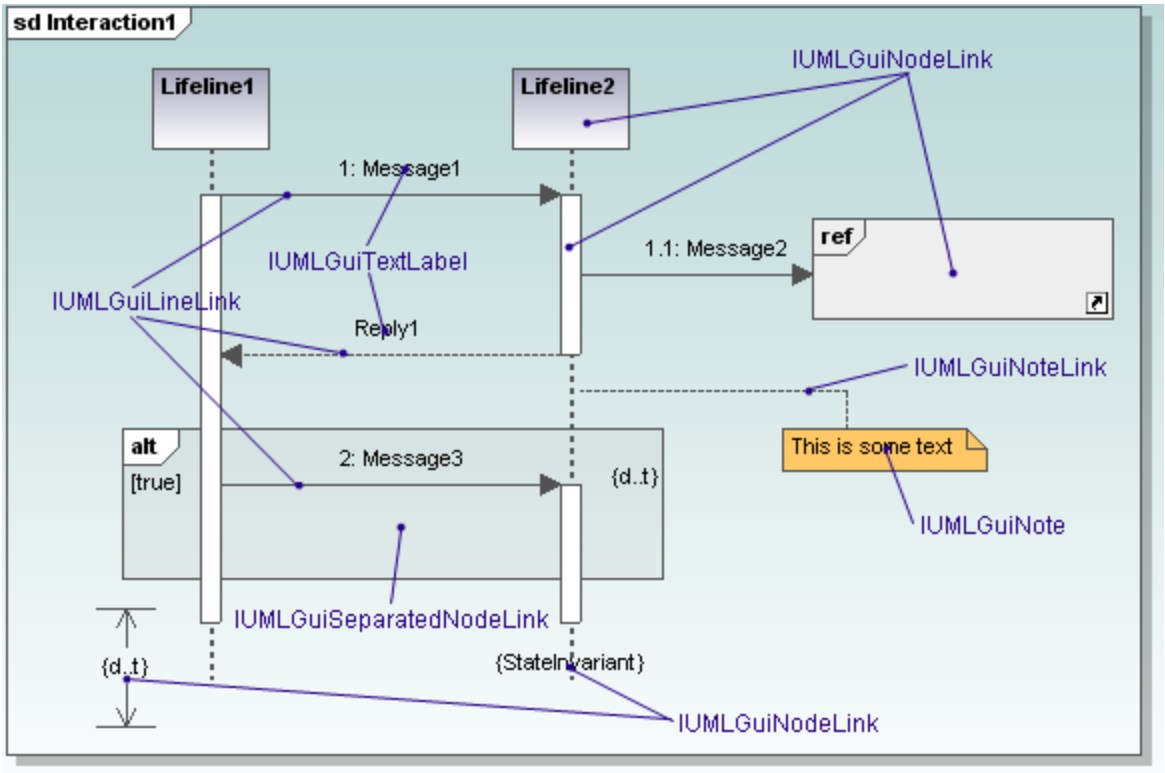
The different styles can be identified by ENUMUMLGuiStyleKind.

17.4.2.3 Graphical Objects

In the UModel API, graphical objects on diagrams are represented by objects derived from the IUMLGuiElement interface. Most of them can be accessed using the IUMLGuiDiagram property 'GuiLinks'.

For most diagrams, most objects which are lines are instances of IUMLGuiLineLink and most other, solid objects or 'nodes' are instances of IUMLGuiNodeLink. These interfaces have properties and methods for manipulating the basic properties of these graphical objects, such as position, color and style.

There are of course more specialized interfaces derived from these general interfaces which provide access to special properties. The following image shows a sequence diagram and the interface representing each graphical object on it:



17.4.3 How to...

17.4.3.1 How to Create Sequence Diagrams

There are two ways to create sequence diagrams programmatically using the UModel API:

- Generating a sequence diagram from existing source code when there is code available that you want to be reverse engineered and displayed as UML diagram
- Manually create a sequence diagram from scratch using `IUMLGuiElements` directly

17.4.3.1.1 How to Generate Sequence Diagrams from Code

Sequence diagrams in UModel can be generated programmatically from an IUMLOperation element. The operation needs to exist in the model and have some source code associated to it.

The operation could possibly have been previously "read" by the reverse engineering functionality of UModel. Creating new Sequence Diagrams programmatically by reverse engineering source code using the UModel API involves two short steps:

- Setting up the options for diagram generation
- Invoking the diagram generation function

The following C# code shows how to set up the options and start the generation of the sequence diagram:

```
// starts the sequence diagram generation process based on an operation given as
parameter
public static void reverseEngineerAndCreateSequenceDiagram(IApplication application,
IUMLOperation operation)
{
    GenerateSequenceDiagramDlg dialog = application.Dialogs.GenerateSequenceDiagramDlg;

    // set some options
    dialog.ShowEmptyCombinedFragments = false;
    dialog.UseDedicatedLineForStaticCalls = true;
    dialog.ShowCodeOfMessagesDisplayedDirectlyBelow = true;
    dialog.ShowCodeInNotes = true;

    dialog.ShowDialog = true; // set this to true if you want the dialog to be displayed

    // generated the sequence diagram now
    application.ActiveDocument.GenerateSequenceDiagram(dialog, operation);
}
```

17.4.3.1.2 How to Create Sequence Diagrams Manually

Creating new Sequence Diagrams programmatically from scratch using the UModel API is basically nothing more than placing interaction fragments, such as Lifelines on a diagram and connecting them with messages.

Messages can easily be created using the `AddUMLLineElement()` method of `IUMLGuiLineLink`, which removes the necessity of creating multiple underlying UML Elements such as MessageEnds, ExecutionOccurrences and similar manually.

To make it simple to create Messages between two interaction fragments such as Lifelines, create a small helper function which calls `AddUMLLineElement()` and layouts the created line:

```
// Creates a message between two interaction fragments (i.e. lifelines, interaction uses,
// combined fragments or gates) and attaches all necessary elements like events and
activation bars.
// Possible values for 'kind': "Message", "Reply", "Create", "Destruct"
protected static IUMLMessage addMessage(int ypos, string kind,
                                IUMLGuiNodeLink from, IUMLGuiNodeLink to,
                                DiagramWindow wnd)
{
    // add message
    IUMLGuiLineLink line = wnd.Diagram.AddUMLLineElement(kind, from, to);

    if (line == null)
        return null;

    // set position of the line where we want it to show up
    wnd.UpdateWindow();

    if (from == to && line.Waypoints.Count > 3)
    {
```

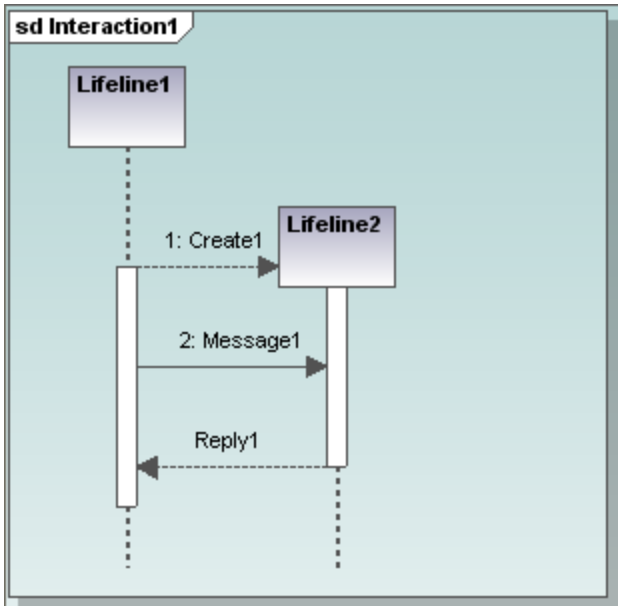
```
        // self-message
        ((IUMLGuiWaypoint)line.Waypoints[1]).SetPos(0, ypos);
        ((IUMLGuiWaypoint)line.Waypoints[4]).SetPos(0, ypos + 25);
    }
    else
    if (line.Waypoints.Count > 1)
    {
        // normal message
        ((IUMLGuiWaypoint)line.Waypoints[1]).SetPos(0, ypos);
        ((IUMLGuiWaypoint)line.Waypoints[2]).SetPos(0, ypos);
    }

    return (IUMLMessage)line.Element;
}
```

As you can see, IUMLDiagram.AddUMLLineElement() accepts as a parameter not only the string "Message", to create a Message Line; but also "Reply", "Create" and "Destruct", for Reply Messages, Creation Messages and Destruction Messages.

In order to create a simple diagram it is only necessary to create a Sequence Diagram in the GuiRoot object, open the diagram, add a handful of lifelines and connect them with messages using this helper function:

```
IDocument document = theapplication.ActiveDocument;

// create diagram and open it
IUMLGuiSequenceDiagram sequenceDiagram =
   (IUMLGuiSequenceDiagram)document.GuiRoot.InsertOwnedDiagramAt(0, document.RootPackage,
 "SequenceDiagram");

DiagramWindow wnd = document.OpenDiagram(sequenceDiagram);

// create two lifelines
IUMLGuiNodeLink lifeline1 = sequenceDiagram.AddUMLElement("Lifeline", 0, 0);
IUMLGuiNodeLink lifeline2 = sequenceDiagram.AddUMLElement("Lifeline", 100, 70);

// connect these lifelines using some messages
addMessage(100, "Create",  lifeline1, lifeline2, wnd);
addMessage(150, "Message", lifeline1, lifeline2, wnd);
addMessage(200, "Reply",   lifeline2, lifeline1, wnd);
```

The resulting created Diagram will look like this:



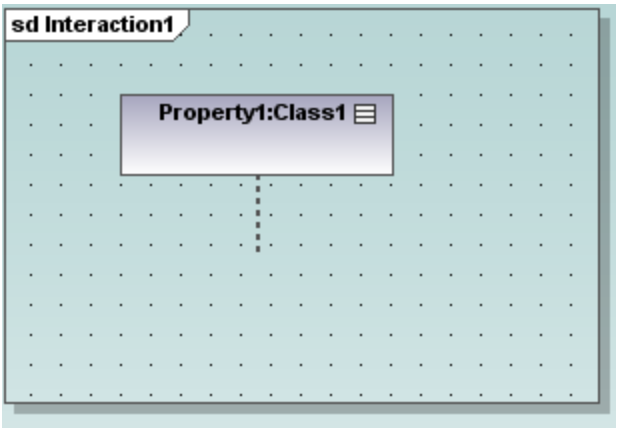
Setting the Type of a Lifeline

To display the Type represented by a Lifeline, be it be a Class, Interface, DataType or similar, this is done by using the `IUMLLifeline.Represents` property which references a `IUMLProperty`. If the Type of this property is set, the Type will show up on the diagram as well.

The following code creates a Lifeline which references a class:

```
// create a class to be referenced by the lifeline
IUMLClass someclass = (IUMLClass)document.RootPackage.InsertPackagedElementAt(0,
"Class");

// create a lifeline and a property with the class as type in the interaction
// of the sequence diagram to reference this class
IUMLInteraction interaction = (IUMLInteraction)sequenceDiagram.LinkedOwner;

IUMLProperty prop = interaction.InsertOwnedAttributeAt(0);
prop.Type = someclass;

UModelLib.IUMLLifeline lifeline = interaction.InsertLifelineAt(0);
lifeline.Represents = (IUMLConnectableElement)prop;

// show the lifeline on the diagram
sequenceDiagram.AddUMLGuiNodeLink(lifeline, 200, 0);
```

The resulting lifeline would then look like this:



Setting the Operation of a Message

Messages usually represent the invocation of an operation of an object. Note: based on the type of the Message (normal Message, Creation, Deletion or Reply) and the existence, or absence of underlying UML elements, such as MessageOccurenceSpecifications or CallEvents, it is not always possible for a Message to represent an Operation, and getting the cprrect UML element to point to the Operation is not that trivial.

This is why the `IUMLMessage` interface in the UModel API, offers the method `SetOperation()` with makes it possible to let a Message refer an Operation if it is able to do so:

```
// create a message, an operation in a class and let the message refer this operation
IUMLMessage msg = addMessage(250, "Message", lifeline1, lifeline2, wnd);

UModelLib.IUMLOperation someoperation = someclass.InsertOwnedOperationAt(0);
someoperation.Name = "SomeOperation";

msg.SetOperation(someoperation);
```

17.4.3.2 Undo / Redo and UMLData Transaction Handling

When modifying the UML data structure using the UModel API, there is no need to take care of Undo/Redo or transactions.

The following code makes three modifications:

```
        public void ChangeClass( IUMLClass iClass )
        {
            iClass.SetName("NewName");
            iClass.Visibility = ENUMUMLVisibilityKind.eVisibility_Public;
            iClass.IsAbstract = true;
        }
```

and for every modification, a new undo-step is created, in other words: the user will have to press the "Undo" button three times in UModel to undo these three changes.

This is not always the required behavior so the UModel API supports "transaction-handling" making it possible to execute multiple modifications in one step.

IDocument has the functionality to define when a group of modifications starts ("BeginModification") and when it ends ("EndModification"):

```
        public void ChangeClass(IUMLClass iClass, IDocument iDoc)
        {
            try
            {
                // make all modifications within one UndoStep; start modification here
                if (!iDoc.BeginModification())
                    return;

                iClass.SetName("NewName");
                iClass.Visibility = ENUMUMLVisibilityKind.eVisibility_Public;
                iClass.IsAbstract = true;

                // do not forget to end modification and finish UndoStep
                iDoc.EndModification();
            }
            catch (System.Exception)
            {
                // rollback made changes
                iDoc.AbortModification();

                // add error handling
            }
        }
```

This kind of transaction handling may only be used for UML data modifications. Other functions, such as e.g. 'synchronize model from code', will create one single Undo step anyway.

17.4.3.3 How to Use Predefined UModel Elements

UModel defines several important elements as "predefined". This includes several internal elements (Root, Component View and Unknown Externals package) as well as the elements of all profiles installed with UModel (e.g. the C#, VB and Java profile).

Predefined elements can be uniquely identified by using ENUMUMLPredefinedElement, which allows direct and easy access to these elements for several functionalities, for example:

- Find a predefined element:

```
// get the CSharp profile
IUMLProfile iCSharpProfile = (IUMLProfile)
iDoc.RootPackage.FindPredefinedOwnedElement(ENUMUMLPredefinedElement.ePredefined_CSharp_P
rofile, false);
```

- Apply a predefined stereotype:

```
// set the CSharp 'delegate' stereotype
```

```
iClass.ApplyPredefinedStereotype(
 ENUMUMLPredefinedElement.ePredefined_CSharp_delegateStereotypeOfClass );
```

- Check if a predefined stereotype is applied:

```
// check if package is a CSharp - namespace (if 'namespace' stereotype is applied)
bool bIsCSharpNamespace =
iPackage.IsPredefinedStereotypeApplied(
 ENUMUMLPredefinedElement.ePredefined_CSharp_namespaceStereotypeOfPackage );
```

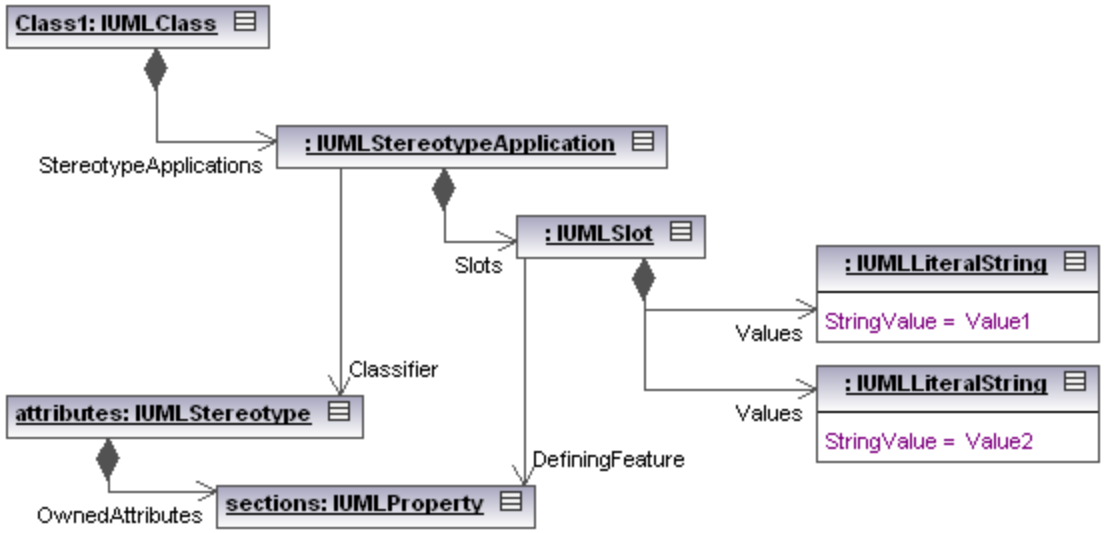
- Set the tagged value of a predefined tag definition:

```
// set attribute-section "STAThread"
// ...
iStereotypeApp.SetPredefinedTaggedValueAt(-1,
ENUMUMLPredefinedElement.ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty,
 iSTAThread.Name);
```

17.4.3.4 How to Work with Stereotypes and Tagged Values

Stereotypes and tagged values are quite complex as defined in the UML Superstructure Specification. UModel has simplified their handling and treats them similar to IUMLInstanceSpecifications and IUMLSlots in UML. In the following sample, the stereotype "attributes" is applied to "Class1", and the tag definition "sections" has the tagged values "Value1" and "Value2":

UModel API introduces IUMLStereotypeApplications and maps the sample above to the following UMLData structure:



Applying stereotypes and setting tagged values using the UModel API is quite simple:

```
IUMLStereotype iStereotypeAttributes = ...;
IUMLProperty iTagDefSections = ...;
IUMLClass iClass = ...;

IUMLStereotypeApplication iStereotypeApp = iClass.ApplyStereotype(iStereotypeAttributes);
iStereotypeApp.SetTaggedValueAt(-1, iTagDefSections, "Value1");
iStereotypeApp.SetTaggedValueAt(-1, iTagDefSections, "Value2");
```

See also the section Predefined UModel elements for information about dealing with predefined stereotypes, tag definitions and tagged values.

17.4.3.5 How to Use UMLData Events and Event Filters

Event receivers must implement the `IUMLDataEvents` interface in order to receive one or more of following possible events from `IUMLData` objects:

`OnBeforeErase`	Sent immediately before the UML data is erased from the model. If multiple data are erased, this event is sent for every `IUMLData` (not only for the topmost one).
`OnAfterAddChild`	Sent when the UML data is added to the model tree. If multiple data are added in one step (e.g. a class with multiple attributes is added to a package) only the topmost `IUMLData` event is sent.
`OnChanged`	Sent when the UML data has been modified (e.g. when a class name is changed).

OnMoveData	Sent when the UML data has been moved to a new parent (e.g. when a class is moved to another package in the ModelTree). This event always occurs twice: once when detaching from the old parent, and once when the UML data is attached to the new parent.

Eventfilter can be set with (combinations of) ENUMUMLDataEventFilter in order to specify which events should be sent by the UModel API. To keep performance high and the overhead as low as possible, event receivers should only register for events they need.

For example, the following code registers OnAfterAddChild events when specifically the root-package gets a new child (no event will arrive if a child of the root-package gets a new child):

```
// ensure we get informed when m_RootPackage (and only itself; we do not care about its
children) gets a new child
m_RootPackage.EventFilter = (int)ENUMUMLDataEventFilter.eUMLDataEvent_AddChild;
```

UMLData events work hierarchically, so the event filter can be set to receive events from the attached IUMLData only, or from the attached IUMLData and any of its children (grandchildren,...).

```
// ensure we get "OnBeforeErase" events also for *any* erased child (grandchild,...) of
the rootpackage
m_RootPackage.EventFilter |= (int)ENUMUMLDataEventFilter.eUMLDataEvent_EraseDataOrChild;
```

UMLData events are also sent when UML data is modified by Undo / Redo, but beware that no UML data modification may be made during Undo / Redo:

```
public void OnAfterAddChild(IUMLData ipUMLParent, IUMLData ipUMLChild)
{
   // check if child was added by undo/redo
      // (we are not allowed to modify anything during Undo/Redo !!)
      IDocument iDoc = (IDocument)ipUMLChild.Parent;
      if (!iDoc.IsInUndoRedo)
      {
         // ...
      }
}
```

17.4.3.6 How to Create and Use Hyperlinks

UModel allows hyperlinks between most modeling elements (except for lines) and:

- any diagram in the current ump project
- any element on a diagram
- any element in the Model Tree
- external documents, e.g. PDF, Excel or Word documents
- web pages

See also: Hyperlinking modeling elements.

Hyperlinks are not part of the UML specification and the UModel API introduces the following interfaces for hyperlinks on

IUMLNamedElements:

- IUMLHyperlink is the common base interface and can be used to open links as well as to retrieve the default- and user-defined link name
- IUMLHyperlink2File to handle external documents and web pages
- IUMLHyperlink2GuiElement to handle any diagram in the current ump project or any element on a diagram
- IUMLHyperlink2Model for hyperlinks to model elements (in the Model Tree)

Examples

Insert a hyperlink to the Altova homepage:

```
IUMLHyperlink2File iHyperlink = iMyClass.InsertOwnedHyperlink2FileAt(-1,
"http://www.altova.com");
```

Insert a hyperlink to a diagram of the current ump project:

```
IUMLGuiDiagram iDiagram = ...;
IUMLHyperlink2GuiElement iHyperlink = iMyClass.InsertOwnedHyperlink2GuiElementAt(-1,
iDiagram, null);
```

Insert a hyperlink to the representation of a class on a diagram:

```
IUMLGuiNodeLink iNodeLink = ...;
IUMLHyperlink2GuiElement iHyperlink = iMyClass.InsertOwnedHyperlink2GuiElementAt(-1,
iNodeLink, null);
```

Insert a hyperlink to an attribute of a class on a diagram:

```
IUMLGuiNodeLink iNodeLink = ...;
IUMLProperty iAttribute = ...;
IUMLHyperlink2GuiElement iHyperlink = iMyClass.InsertOwnedHyperlink2GuiElementAt(-1,
iNodeLink, iAttribute);
```

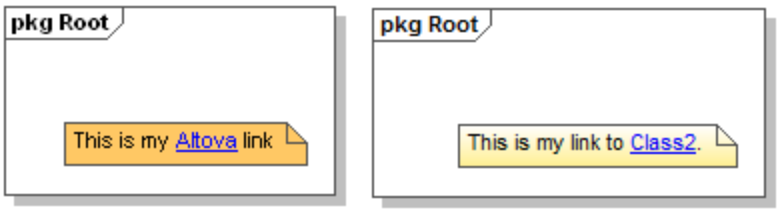
Insert a hyperlink to the same attribute (from above) in the Model Tree:

```
IUMLHyperlink2Model iHyperlink = iMyClass.InsertOwnedHyperlink2ModelAt(-1, iAttribute);
```

Open all hyperlinks of an IUMLNamedElement:

```
foreach (IUMLHyperlink iHyperlink in iMyClass.OwnedHyperlinks)
   iHyperlink.OpenLink();
```

UModel also allows hyperlinks in notes (IUMLGuiNote) and comments (IUMLComment):



These are handled by IUMLGuiTextHyperlinks (respectively IUMLCommentTextHyperlinks) and the start- and end-character position of the hyperlink must be specified, e.g:

```
IUMLGuiDiagram iDiagram = ...;
IUMLGuiNote iNote = iDiagram.AddUMLGuiNote(200, 100);

iNote.NoteText = "This is my Altova link";
int nStart = iNote.NoteText.IndexOf("Altova");
int nEnd = nStart + "Altova".Length;

IUMLGuiTextHyperlink iHyperlink = iNote.InsertOwnedGuiTextHyperlinkAt(nStart, nEnd,
"http://www.altova.com");
```

Similar for hyperlinks in comments:

```
IUMLComment iComment = ...;
IUMLClass iClass2 = ...;

iComment.Body = "This is my link to Class2";
int nStart = iComment.Body.IndexOf("Class2");
int nEnd = nStart + "Class2".Length;

IUMLCommentTextHyperlink iHyperlink = iComment.InsertOwnedCommentTextHyperlinkAt(nStart,
nEnd, "");
iHyperlink.SetHyperlinkModelElementAddress( iClass2 );
```

17.4.3.7 Handle Errors

The UModel API returns errors in two different ways. Every API method returns an `HRESULT`. This return value informs the caller about any errors during the execution of the method. If the call was successful, the return value is equal to `S_OK`. C/C++ programmers generally use `HRESULT` to detect errors.

VisualBasic, scripting languages, and other high-level development environments do not give the programmer access to the returning `HRESULT` of a COM call. They use the second error-raising mechanism supported by the UModel API, the `IErrorInfo` interface. If an error occurs, the API creates a new object that implements the `IErrorInfo` interface. The development environment takes this interface and fills its own error-handling mechanism with the provided information.

The example code listings below show how to deal with errors raised from the UModel API in different development environments.

Visual Basic

A common way to handle errors in VisualBasic is to define an error handler. This error handler can be set with the `On Error` statement. Usually the handler displays an error message and performs cleanup functions to avoid spare references and any kind of resource leaks.

VisualBasic fills its own `Err` object with the information from the `IErrorInfo` interface.

```
Sub Validate()
  'place variable declarations here

  'set error handler
  On Error GoTo ErrorHandler

  'if DoSomeWork fails, program execution continues at ErrorHandler:
  objUModel.ActiveDocument.DoSomeWork()

  'additional code comes here

  'exit
  Exit Sub

  ErrorHandler:
  MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) &
     "Description: " & Err.Description)
End Sub
```

JavaScript

The Microsoft implementation of JavaScript (JScript) provides a try-catch mechanism to deal with errors raised from COM calls. It is very similar to the Visual Basic approach, in that you also declare an error object containing the necessary information.

```
   function Generate()
   {
      // please insert variable declarations here

      try
      {
         objUModel.ActiveDocument.DoSomeWork();
      }
      catch(Error)
      {
         sError = Error.description;
         nErrorCode = Error.number & 0xffff;
         return false;
      }

      return true;
   }
```

C/C++

C/C++ gives you easy access to the `HRESULT` of the COM call and to the `IErrorInterface`.

```
HRESULT hr;

// Call DoSomeWork() from the UModel API
if(FAILED(hr = ipDocument->DoSomeWork()))
{
   IErrorInfo *ipErrorInfo = Null;

   if(SUCCEEDED(::GetErrorInfo(0, &ipErrorInfo)))
   {
      BSTR   bstrDescr;
      ipErrorInfo->GetDescription(&bstrDescr);

      // handle Error information
      wprintf(L"Error message:\t%s\n",bstrDescr);
      ::SysFreeString(bstrDescr);

      // release Error info
      ipErrorInfo->Release();
   }
}
```

17.4.4 C# API Examples

To help you get started, your UModel package contains an example C# project, which is located at **C:\Users\<username>\Documents\Altova\UModel2023\UModelExamples\API**.

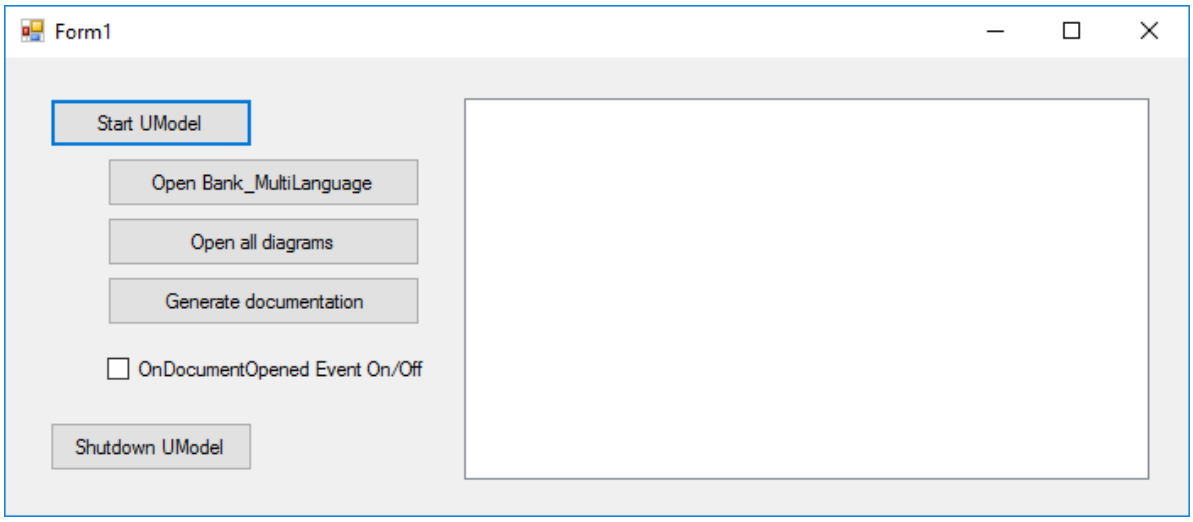
Importantly, this example project includes a reference to the UModel Type Library, see How to Reference the UModel Type Library. A reference to the UModel Type Library is required in each project where you need the UModel API. This makes it possible to instantiate the main application object from your code as follows:

```
UModelLib.Application um = new UModelLib.Application();
MessageBox.Show(String.Format("Hello from UModel API version {0}.{1}",
um.APIMajorVersion, um.APIMinorVersion));
```

Si tiene un sistema operativo de 64 bits pero está usando una versión de UModel para 32 bits, agregue la plataforma x86 al **gestor de configuraciones** de la solución y compile la muestra usando esta configuración. Para acceder al gestor de configuraciones, ejecute el comando de menú **Compilar | Administrador de configuración**.

The example application displays a Windows form with buttons that invoke basic UModel operations:

- Start UModel
- Open **Bank_MultiLanguage.ump**
- Open All Diagrams
- Generate documentation for the currently active document
- Shows how to listen to UModel events (`OnDocumentOpened` Event On/Off)
- Shutdown UModel

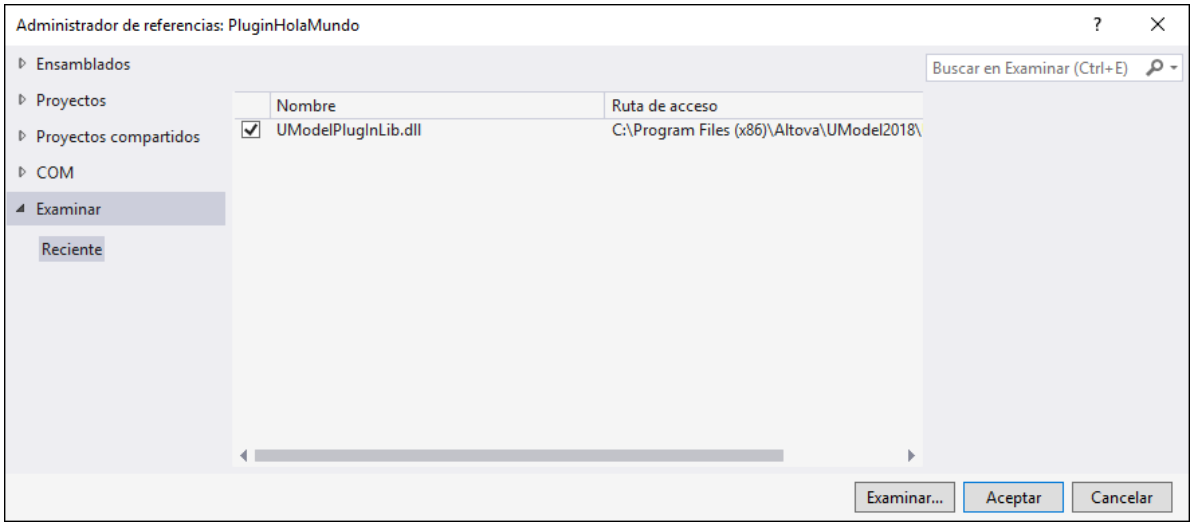


The code essentially consists of a series of handlers for the buttons in the user interface shown above. Note that you may need to adjust the path to the UModel examples folder which is referenced from the code.

17.4.4.1 How to Reference the UModel Type Library

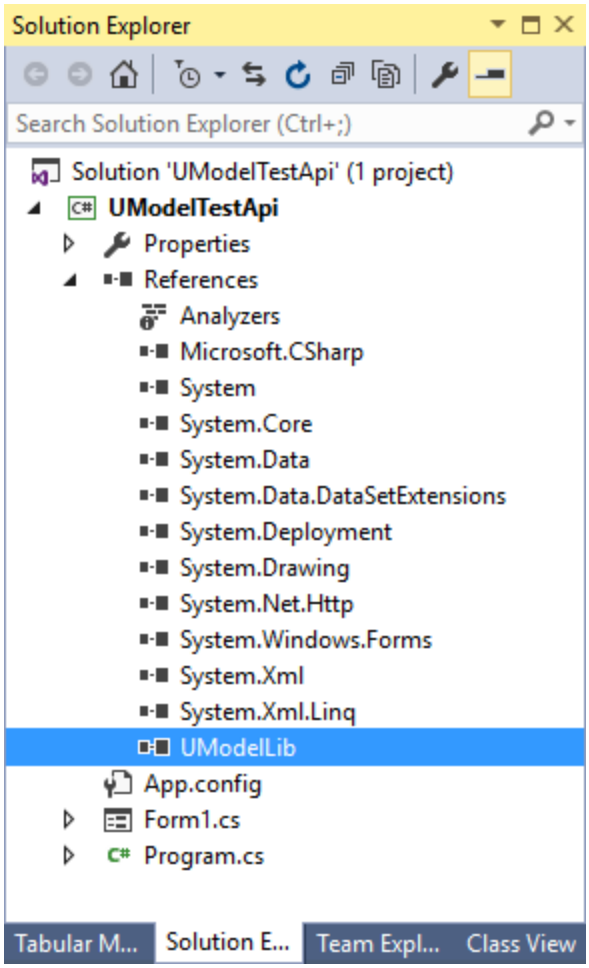
Para acceder a las funciones de la API de UModel desde su proyecto de Visual Studio, agregue una referencia a la biblioteca de tipos de UModel en Visual Studio:

1. Cree un proyecto nuevo en Visual Studio o abra uno que ya existe.
2. En el menú Proyecto haga clic en **Agregar referencias**.
3. En la sección COM, seleccione **Biblioteca de tipos de UModel** de la lista. Si esta entrada no está disponible en la sección COM, haga clic en **Examinar** y seleccione el archivo `UModel.tlb` en la carpeta de aplicaciones de UModel.



Nota: no confunda la **Biblioteca de tipos de UModel** con la **Biblioteca de tipos del complemento de UModel**. Esta última se puede usar para crear complementos personalizados e integrarlos en UModel (véase Agregar una referencia a la biblioteca del complemento de UModel).

Tras seguir estos pasos, la biblioteca de tipos de UModel debería estar disponible en la lista de referencias de su solución de Visual Studio:



17.4.4.2 Importing Binary Types Programmatically

With UModel, you can import binary types from .NET .dll or Java .jar files, either from the graphical user interface, or programmatically using the UModel API. This example illustrates how to import binary types from a NET .dll file into UModel using the UModel API. For information about importing binary types from the graphical user interface, see Importing Java, C# and VB.NET Binaries.

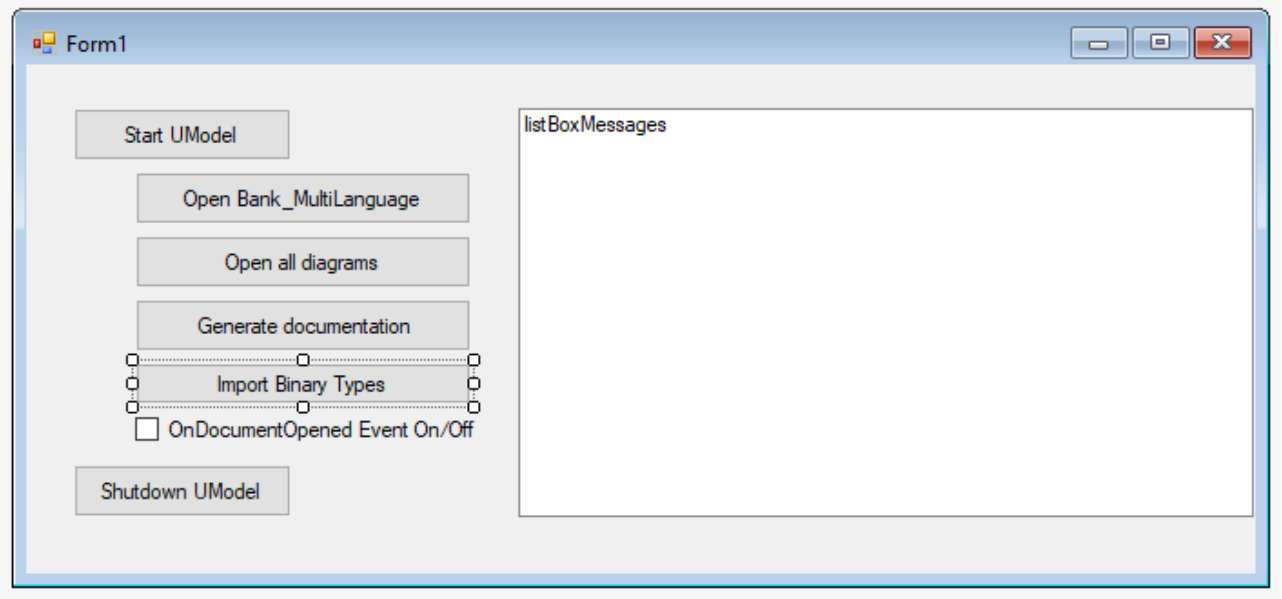
This example uses Microsoft Visual Studio 2015 and C#. The instructions below (except for the code listing) are similar for VB.NET. To complete this example, you also need a .dll that contains some types (such as classes or interfaces) that you would like to import into UModel.

To accomplish the task, we will use an existing C# demo application that already integrates into the UModel API, rather than creating a new project from scratch. Namely, we will add to this demo application a new button. When clicked, the button will create a new UModel project and import into it types from a .dll file. To begin, run Visual Studio and open the following solution: **C:**

\Usuarios\<usuario>\Documentos\Altova\UModel2023\UModelExamplesAPI\C#\AutomateUModel_VS2010.sln.

Note: The demo application already includes a reference to the UModel Type Library so it is not necessary to add a reference explicitly. However, if you are creating a new Visual Studio project, make sure to reference the UModel Type Library from your project, see How to Reference the UModel Type Library.

Next, open the `Form1.cs` in the Design Editor and add a new button. Let's call it **Import Binary Types**.



Double-click the new button and paste the following code into the body of the handler method. Make sure that the path to the .dll file is correct and that the .dll qualifies for import of binary types (that is, it must not be obfuscated).

```
try
{
    // Create a new document
    UModelDocument = UModel.NewDocument();
    // Instantiate the Import Binary Types dialog
    UModelLib.ImportBinaryTypesDlg dlg = UModel.Dialogs.ImportBinaryTypesDlg;
    // Set the .NET runtime version according to your environment (must be greater than
v2.0)
    dlg.Runtime = "v2.0.50727";
    // Set the import language (C# 6.0, in this case)
    dlg.Language = UModelLib.ENUMCodeLangVersion.eCodeLang_CSharp_6_0;
    // No need to show the dialog since we want to do this programmatically
    dlg.ShowDialog = false;
    // Add a new binary type entry to be imported
    UModelLib.IBinaryTypeEntry entry = dlg.CSharp_BinaryTypes.AddItem();
    // Specify the .dll to import (make sure to adjust the path)
    entry.Entry = "C:\\Path\\To\\My.dll";
    // All types shall be imported from this .dll
    entry.ImportTypes = true;
```



```
    // The .dll is an executable
    entry.Executeable = true;
    // Perform the actual import
    UModelDocument.ImportBinaryTypes(dlg);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

Importing all types

The code above essentially creates a new UModel project, sets the import options in the "Import Binary Types" dialog box, and performs the actual import of binary types.

Build and run the solution. Click **Start UModel**, and be patient while the application loads. Only after the application has finished loading, click **Import Binary Types**, and observe the outcome in the Messages window of UModel.

If you would like to import only specific types, set the `ImportTypes` property is **false**, and supply the types to be imported as arguments to the `TypesToImport` method. The list of distinct types can be separated by comma, semi-colon, or space characters, as illustrated in the code listing below.

```
try
{
    UModelDocument = UModel.NewDocument();
    UModelLib.ImportBinaryTypesDlg dlg = UModel.Dialogs.ImportBinaryTypesDlg;
    dlg.ShowDialog = false;
    dlg.CSharp_BinaryTypes.RemoveAllItems();
    UModelLib.IBinaryTypeEntry entry = dlg.CSharp_BinaryTypes.AddItem();
    entry.Entry = "C:\\Path\\To\\My.dll";
    entry.ImportTypes = false;
    entry.Executeable = true;
    // import only specific types:
    entry.TypesToImport = "MyNamespace.Class1; MyNamespace.Class2";
    UModelDocument.ImportBinaryTypes(dlg);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

Importing distinct types

17.4.4.3 "Set Styles" Sample

The following sample sets multiple styles for selected diagram elements (if style is available and not already set). The sample uses both the UModel API and the UModel IDE Plug-In library and is available in the following file: **..\UModelExamples\IDEPlugIn\Styles\Styles.cs**.

The solution also includes two setup projects (in .vdproj format, for 32-bit and 64-bit platforms). The setup installs all necessary files, and registers the IDE plug-in for COM and UModel on your target system, so that the plug-in is automatically loaded when UModel is started the next time.

Notes:

- To build and run the sample, the same requirements as for other UModel IDE Plug-ins apply, see Build and Run the Plug-In.
- Visual Studio setup projects are not supported starting with Visual Studio 2012 and require a separate extension to be opened. See the information messages displayed by the Visual Studio migration wizard for more details.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using UModelLib;
using UModelPlugInLib;

/*
 * Styles sample
 * set following styles for selected diagram elements
 *      Fill Color
 *      Header Gradient Begin Color
 *      Header Gradient End Color
 * if style is available and not already set
 */

namespace Styles
{
    public class UModelStyles : UModelPlugInLib.IUModelPlugIn
    {
        bool m_bPlugInVersionOK = true; // verify if UModel-API has been changed in a way
that a recompile of this plug-in is recommended

        #region helpers

        protected string GetPlugInPath()
        {
            string sDLLPath = System.Reflection.Assembly.GetExecutingAssembly().Location;
            return System.IO.Path.GetDirectoryName(sDLLPath);
        }

        #endregion

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "Styles sample Plug-in for UModel;This Plug-in demonstrates how to
change several styles of the selected diagram elements.";
        }

        public string GetUIModifications()
        {
            try
            {
```

```
                string sPath = GetPlugInPath();

                System.IO.StreamReader myFile = new System.IO.StreamReader(sPath + "\
\config.xml");
                string sRet = myFile.ReadToEnd();
                myFile.Close();

                // this replaces the token "**path**" from the XML file with
                // the actual installation path of the plug-in to get the image file
                return sRet.Replace("**path**", sPath);
            }
            catch (System.Exception ex)
            {
                MessageBox.Show("Error in GetUIModifications:" + ex.Message);
                throw ex;
            }
        }

        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized

            // verify if UModel-API has been changed in a way that a recompile of this
plug-in is recommended:
            IApplication iApp = (IApplication)pUModel;
            if (iApp == null || iApp.APIMajorVersion != 5) // this plug-in was compiled
for API major version '5'!
            {
                MessageBox.Show("'Styles': This Plug-in has been made with a previous
version of the UModel-API and should be recompiled.\nDisabled Plug-in commands in the
meantime.");
                m_bPlugInVersionOK = false;
            }
        }

        public void OnShutdown(object pUModel)
        {
            // application will shutdown; release all unused objects
            GC.Collect();
        }

        public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
        {
            UModelUpdateAction action = UModelUpdateAction.UModelUpdateAction_Disable;
            if (!m_bPlugInVersionOK)
                return action;

            // check for "fill red"
            if (nID == 3 || nID == 6)
                action = OnUpdateSetStyles((IApplication)pUModel);

            // check for "fill green"
            if (nID == 4 || nID == 7)
                action = OnUpdateSetStyles((IApplication)pUModel);
```

```
            // release unused objects
            GC.Collect();

            return action;
        }

        public void OnCommand(int nID, object pUModel)
        {
            if (!m_bPlugInVersionOK)
                return;

            // fill red
            if (nID == 3 || nID == 6)
                OnSetStyles((IApplication)pUModel, "red");

            // fill green
            if (nID == 4 || nID == 7)
                OnSetStyles((IApplication)pUModel, "green");

            // release unused objects
            GC.Collect();
        }

        #endregion

        #region SetStyles // set styles of selected diagram elements

        UModelUpdateAction OnUpdateSetStyles(IApplication pUModel)
        {
            if (pUModel == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the active document of the application
            IDocument iDoc = pUModel.ActiveDocument;
            if (iDoc == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the active diagram window
            IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
            if ( iActiveDiagram == null )
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the selected elements on the active diagram
            IUMLDataList iSelection = iActiveDiagram.SelectedGuiElements;
            if ( iSelection == null )
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // search all selected elements, if at least one has one of the styles to
change
            foreach ( IUMLGuiElement iSelGuiElement in iSelection )
            {
                // verify if it is a GuiVisibleElement (with Styles) and if it may be
modified
                if ( iSelGuiElement is IUMLGuiVisibleElement &&
iSelGuiElement.IsEditable )
                {
                    IUMLGuiVisibleElement iVisGuiElement = (IUMLGuiVisibleElement)
iSelGuiElement;
```

```
                    if
( iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_FillColor) != null ||
                        iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_
HeaderGradientBeginColor) != null ||
                        iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_
HeaderGradientEndColor) != null )
                    {
                        return UModelUpdateAction.UModelUpdateAction_Enable;
                    }
                }
            }

            // nothing found => disable command
            return UModelUpdateAction.UModelUpdateAction_Disable;
        }

        public void OnSetStyles(IApplication pUModel, string sColor)
        {
            if (pUModel == null)
                return;

            // get the active document of the application
            IDocument iDoc = pUModel.ActiveDocument;
            if (iDoc == null)
                return;

            // get the active diagram window
            IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
            if (iActiveDiagram == null)
                return;

            // get the selected elements on the active diagram
            IUMLDataList iSelection = iActiveDiagram.SelectedGuiElements;
            if (iSelection == null)
                return;

            try
            {
                // make all modifications within one UndoStep; start modification here
                if (!iDoc.BeginModification())
                    return;

                // search all selected elements, and change the style if the wanted value
is not already used (directly applied or through style-chain)
                foreach (IUMLGuiElement iSelGuiElement in iSelection)
                {
                    // verify if it is a GuiVisibleElement (with Styles) and if it may be
modified
                     if (iSelGuiElement is IUMLGuiVisibleElement &&
iSelGuiElement.IsEditable )
                    {
                        IUMLGuiVisibleElement iVisGuiElement = (IUMLGuiVisibleElement)
iSelGuiElement;

                        // set Fill Color if possible and not already set
                        IUMLGuiStyle iStyle =
iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_FillColor);
                        if (iStyle != null && iStyle.UsedValue != sColor)
```

```
                            iStyle.Value = sColor;

                        // set Header Gradient Begin Color if possible and not already
set
                        iStyle =
iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_HeaderGradientBeginColor)
;
                        if (iStyle != null && iStyle.UsedValue != sColor)
                            iStyle.Value = sColor;

                        // set Header Gradient End Color if possible and not already set
                        iStyle =
iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_HeaderGradientEndColor);
                        if (iStyle != null && iStyle.UsedValue != sColor)
                            iStyle.Value = sColor;
                    }
                }

                // do not forget to end modification and finish UndoStep
                iDoc.EndModification();
            }
            catch( System.Exception )
            {
                // rollback made changes
                iDoc.AbortModification();

                // add error handling
            }
        }

        #endregion
    }
}
```

17.4.4.4 "C# Delegate" Sample

The following sample inserts a new C# delegate at the top/left corner of the active diagram window (if this diagram is inside a C# namespace root). The sample uses both the UModel API and the UModel IDE Plug-In library and is available in the following file: **..\UModelExamples\IDEPlugIn\CSharpDelegate\UModelCSharpDelegate.cs**.

To build and run the sample, the same requirements as for other UModel IDE Plug-ins apply, see Build and Run the Plug-In.

```
using System;
using System.Collections.Generic;
using System.Text;
using UModelLib;
using UModelPlugInLib;

/*
 * CSharp delegate sample
 * add a new CSharp delegate on the top/left corner of the active class diagram if
```

```
possible
 * (i.e. when diagram is inside a C# root namespace)
 */

namespace CSharpDelegate
{
    public class UModelCSharpDelegate : UModelPlugInLib.IUModelPlugIn
    {
        #region helpers

        protected string GetPlugInPath()
        {
            string sDLLPath = System.Reflection.Assembly.GetExecutingAssembly().Location;
            return System.IO.Path.GetDirectoryName(sDLLPath);
        }

        #endregion

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "CSharpDelegate sample Plug-in for UModel;This Plug-in demonstrates
how to create a new CSharp delegate on a class diagram.";
        }

        public string GetUIModifications()
        {
            try
            {
                string sPath = GetPlugInPath();

                System.IO.StreamReader myFile = new System.IO.StreamReader(sPath + "\
\config.xml");
                string sRet = myFile.ReadToEnd();
                myFile.Close();

                // this replaces the token "**path**" from the XML file with
                // the actual installation path of the plug-in to get the image file
                return sRet.Replace("**path**", sPath);
            }
            catch (System.Exception ex)
            {
                System.Windows.Forms.MessageBox.Show("Error in GetUIModifications:" +
ex.Message);
                throw ex;
            }
        }


        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized
        }
```

```
public void OnShutdown(object pUModel)
{
    // application will shutdown; release all unused objects
    GC.Collect();
}

public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
{
    UModelUpdateAction action = UModelUpdateAction.UModelUpdateAction_Disable;

    // check if we can add a new CSharpDelegate on the active diagram
    if (nID == 3 || nID == 4)
        action = OnUpdateAddNewCSharpDelegate((IApplication)pUModel);

    // release unused objects
    GC.Collect();

    return action;
}

public void OnCommand(int nID, object pUModel)
{
    // add a new CSharpDelegate on the active diagram
    if (nID == 3 || nID == 4)
        OnAddNewCSharpDelegate((IApplication)pUModel);

    // release unused objects
    GC.Collect();
}

#endregion

#region AddNewCSharpDelegate // add new CSharp delegate on active diagram

UModelUpdateAction OnUpdateAddNewCSharpDelegate(IApplication pUModel)
{
    if (pUModel == null)
        return UModelUpdateAction.UModelUpdateAction_Disable;

    // get the active document of the application
    IDocument iDoc = pUModel.ActiveDocument;
    if (iDoc == null)
        return UModelUpdateAction.UModelUpdateAction_Disable;

    // get the active diagram window
    IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
    if ( iActiveDiagram == null )
        return UModelUpdateAction.UModelUpdateAction_Disable;

    // get the UML diagram of the diagram window
    IUMLGuiDiagram iUMLDiagram = iActiveDiagram.Diagram;

    // check if it is a class diagram
    if ( !( iUMLDiagram is IUMLGuiClassDiagram) )
        return UModelUpdateAction.UModelUpdateAction_Disable;

    // verify if the diagram may be modified
    if ( !iUMLDiagram.IsEditable )
```

```
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the UML element, which "owns" the class diagram
            IUMLElement iDiagramOwner = iUMLDiagram.LinkedOwner;
            if (iDiagramOwner == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // verify if we are inside a CSharp namespace root (otherwise adding a CSharp
delegate makes no sense)
            IUMLElement iFindNamespaceRoot = iDiagramOwner;
            while( iFindNamespaceRoot != null)
            {
                if ( iFindNamespaceRoot is IUMLPackage)
                {
                    IUMLPackage iPackage = (IUMLPackage) iFindNamespaceRoot;
                    if
( iPackage.IsCodeLangNamespaceRoot( ENUMCodeLang.eCodeLang_CSharp ) )
                        return UModelUpdateAction.UModelUpdateAction_Enable;
                }

                iFindNamespaceRoot = iFindNamespaceRoot.Owner;
            }

            // nothing found => disable command
            return UModelUpdateAction.UModelUpdateAction_Disable;
        }

        public void OnAddNewCSharpDelegate(IApplication pUModel)
        {
            if (pUModel == null)
                return;

            // get the active document of the application
            IDocument iDoc = pUModel.ActiveDocument;
            if (iDoc == null)
                return;

            // get the active diagram window
            IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
            if (iActiveDiagram == null)
                return;

            // get the UML diagram of the diagram window
            IUMLGuiDiagram iUMLDiagram = iActiveDiagram.Diagram;

            // get the CSharp profile
            IUMLProfile iCSharpProfile = (IUMLProfile)
iDoc.RootPackage.FindPredefinedOwnedElement(ENUMUMLPredefinedElement.ePredefined_CSharp_P
rofile, false);
            if ( iCSharpProfile == null)
                return;

            try
            {
                // make all modifications within one UndoStep; start modification here
                if (!iDoc.BeginModification())
                    return;

                // get top left corner of the visible diagram area
```

```
                int nInsertPosX = iActiveDiagram.ScrollPosX;
                int nInsertPosY = iActiveDiagram.ScrollPosY;

                // add new class on diagram
                IUMLGuiNodeLink iClassNode = iUMLDiagram.AddUMLElement("Class",
nInsertPosX + 100, nInsertPosY + 100);

                IUMLClass iClass = (IUMLClass) iClassNode.Element;
                // use SetName (instead of Name) that UModel automatically generates a
valid, unique name starting with "NewDelegate"
                iClass.SetName("NewDelegate");

                // set the CSharp 'delegate' stereotype
                iClass.ApplyPredefinedStereotype(
 ENUMUMLPredefinedElement.ePredefined_CSharp_delegateStereotypeOfClass );

                // set attribute-section "STAThread"
                IUMLStereotypeApplication iStereotypeApp =
iClass.ApplyPredefinedStereotype(ENUMUMLPredefinedElement.ePredefined_CSharp_attributesSt
ereotypeOfClass);
                IUMLEnumerationLiteral iSTAThread = (IUMLEnumerationLiteral)
iCSharpProfile.FindPredefinedOwnedElement(ENUMUMLPredefinedElement.ePredefined_CSharp_Att
ributePresetsEnumeration_STAThreadEnumerationLiteral, true);
                iStereotypeApp.SetPredefinedTaggedValueAt(-1,
ENUMUMLPredefinedElement.ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty,
 iSTAThread.Name);

                // add delegate operation:
                IUMLOperation iOperation = iClass.InsertOwnedOperationAt(-1);
                iOperation.SetName( "delegate");

                // per default set operation-return type "void"
                IUMLPrimitiveType iTypeVoid = (IUMLPrimitiveType)
iCSharpProfile.FindPredefinedOwnedElement(ENUMUMLPredefinedElement.ePredefined_CSharp_voi
dPrimitiveType, true);
                iOperation.Type = iTypeVoid;

                // do not forget to end modification and finish UndoStep
                iDoc.EndModification();

                // at last focus newly inserted delegate on the diagram:
                iActiveDiagram.SelectGuiElement(iClassNode, true);
            }
            catch( System.Exception )
            {
                // rollback made changes
                iDoc.AbortModification();

                // add error handling
            }
        }

        #endregion
    }
}
```

17.4.4.5 "Set Prefix" Sample

The following sample automatically sets a prefix when new attributes or enumeration literals are added to your UModel project. The sample uses both the UModel API and the UModel IDE Plug-In library and is available in the following file: **..\UModelExamples\IDEPlugIn\DefaultPrefix\DefaultPrefix.cs**.

To build and run the sample, the same requirements as for other UModel IDE Plug-ins apply, see Build and Run the Plug-In.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.Runtime.InteropServices.ComTypes;
using UModelLib;
using UModelPlugInLib;

/*
 * DefaultPrefix sample
 * listen for newly added UML data and
 * set the prefix of properties ('m_') and EnumerationLiterals ('k_')
 * if the corresponding option is turned on
 */

namespace DefaultPrefix
{
    /* UModelDefaultPrefix is the main class of this plugin and implements
UModelPlugInLib.IUModelPlugIn
     * it is also responsible for attaching/detaching UModelApplicationEvents to/from
UModels IApplication interface
     * and implements the handling of turning on/off the whole "SetPrefix" functionality
     */
    public class UModelDefaultPrefix : UModelPlugInLib.IUModelPlugIn
    {
        // variable which defines whether "SetPrefix" functionality is turned on or off
        bool m_bSetPrefix = true;

        // reference to UModelApplicationEvents; is only used when "SetPrefix"
functionality is turned on (to reduce overhead in the other case)
        UModelApplicationEvents m_AppEvents = null;

        #region helpers

        protected string GetPlugInPath()
        {
            string sDLLPath = System.Reflection.Assembly.GetExecutingAssembly().Location;
            return System.IO.Path.GetDirectoryName(sDLLPath);
        }

        #endregion

        // create UModelApplicationEvents and attach it to IApplication
        protected void AttachAppEvents( IApplication iUModelApp )
        {
            if (m_AppEvents == null && iUModelApp != null)
```

```
            {
                m_AppEvents = new UModelApplicationEvents();
                m_AppEvents.Attach(iUModelApp);
            }
        }

        // detach UModelApplicationEvents;
        protected void DetachAppEvents()
        {
            if (m_AppEvents != null)
            {
                m_AppEvents.Detach();
                m_AppEvents = null;
            }
        }

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "DefaultPrefix sample Plug-in for UModel;This Plug-in demonstrates how
to attach to several callback interfaces and how to add a prefix to newly inserted
elements.";
        }

        public string GetUIModifications()
        {
            try
            {
                string sPath = GetPlugInPath();

                System.IO.StreamReader myFile = new System.IO.StreamReader(sPath + "\
\config.xml");
                string sRet = myFile.ReadToEnd();
                myFile.Close();

                // this replaces the token "**path**" from the XML file with
                // the actual installation path of the plug-in to get the image file
                return sRet.Replace("**path**", sPath);
            }
            catch (System.Exception ex)
            {
                System.Windows.Forms.MessageBox.Show("Error in GetUIModifications:" +
ex.Message);
                throw ex;
            }
        }

        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized
            // and we can attach UModelApplicationEvents
            AttachAppEvents( (IApplication)pUModel );
        }
```

```
        public void OnShutdown(object pUModel)
        {
            // detach UModelApplicationEvents; stop receiving events
            DetachAppEvents();

            // application will shutdown; release all unused objects
            GC.Collect();
        }

        public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
        {
            UModelUpdateAction action = UModelUpdateAction.UModelUpdateAction_Disable;

            // check if automatically setting the prefix is turned on:
            if (nID == 3 || nID == 4)
            {
                action = UModelUpdateAction.UModelUpdateAction_Enable;

                if (m_bSetPrefix)
                    action |= UModelUpdateAction.UModelUpdateAction_Check;
            }

            // release unused objects
            //GC.Collect(); not necessary since we do not access objects here

            return action;
        }

        public void OnCommand(int nID, object pUModel)
        {
            // toggle automatically setting the prefix:
            if (nID == 3 || nID == 4)
                m_bSetPrefix = !m_bSetPrefix;

            // attach UModelApplicationEvents when "SetPrefix" functionality is turned
on; detach otherwise
            if (m_bSetPrefix)
                AttachAppEvents((IApplication)pUModel);
            else
                DetachAppEvents();

            // release unused objects
            GC.Collect();
        }

        #endregion
    }


    /* UModelApplicationEvents is an eventhandler to receive _IApplicationEvents
     * that we know when UModel documents are opened or closed
     * and that we can Attach/Detach UModelDataEvents
     * We are interested in all _IApplicationEvents and use a connectionpoint to connect
to all these events
     */
    public class UModelApplicationEvents : UModelLib._IApplicationEvents
    {
        // connection point to _IApplicationEvents
```

```
        System.Runtime.InteropServices.ComTypes.IConnectionPoint m_cpApplicationEvents =
null;
        // connection cookie
        int m_nApplicationtEventsCookie = 0;
        // we always hold a reference to UModelDataEvents
        UModelDataEvents m_UMLDataEvents = new UModelDataEvents();

        public void Attach(IApplication iApp)
        {
            if (m_cpApplicationEvents == null && iApp != null)
            {
                // find connection point of _IApplicationEvents
                IConnectionPointContainer icpc = (IConnectionPointContainer)iApp;
                Guid IID = typeof(UModelLib._IApplicationEvents).GUID;
                icpc.FindConnectionPoint(ref IID, out m_cpApplicationEvents);

                // advise UModelApplicationEvents as sink for _IApplicationEvents
                m_cpApplicationEvents.Advise(this, out m_nApplicationtEventsCookie);

                // also attach UModelDataEvents to the current document and start
receiving events there
                m_UMLDataEvents.Attach(iApp.ActiveDocument);
            }
        }
        public void Detach()
        {
            if (m_cpApplicationEvents != null)
            {
                // also detach UModelDataEvents and stop receiving events there
                m_UMLDataEvents.Detach();

                // terminate established connection to _IApplicationEvents
                m_cpApplicationEvents.Unadvise(m_nApplicationtEventsCookie);
                m_cpApplicationEvents = null;
            }
        }

        #region _IApplicationEvents Members
        public void OnNewDocument(Document ipDocument)
        {
            Debug.WriteLine("UModelApplicationEvents.OnNewDocument " + ipDocument.Name);
            // a new document has been created in UModel => (re-)connect UModelDataEvents
            m_UMLDataEvents.Attach(ipDocument);
        }

        public void OnDocumentOpened(Document ipDocument)
        {
            Debug.WriteLine("UModelApplicationEvents.OnDocumentOpened " +
ipDocument.Name);
            // a document has been opened in UModel => (re-)connect UModelDataEvents
            m_UMLDataEvents.Attach(ipDocument);
        }

        public void OnDocumentClosed(Document ipDocument)
        {
            Debug.WriteLine("UModelApplicationEvents.OnDocumentClosed " +
ipDocument.Name);
            // document has been closed in UModel => disconnect UModelDataEvents
            m_UMLDataEvents.Detach();
```

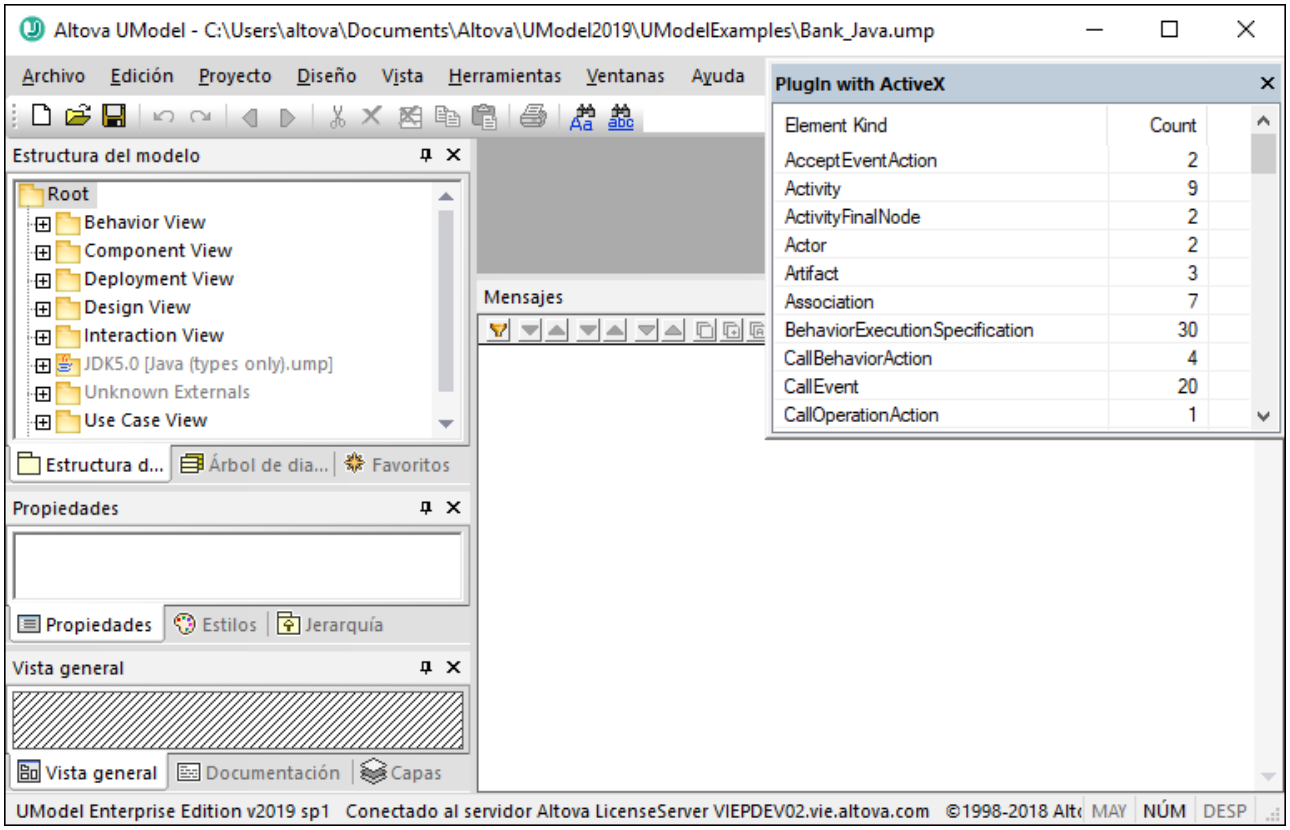
```
        }

        public void OnShutdown()
        {
            Debug.WriteLine("UModelApplicationEvents.OnShutdown");
        }

        #endregion
    }


    /* UModelDataEvents is an eventhandler to receive _IUMLDataEvents
     * from the root-package and all its children.
     * We are only interested in 'OnAfterAddChild' events, so we use a delegate to
connect to this event.
     */
    public class UModelDataEvents : UModelLib._IUMLDataEvents
    {
        // hold a reference to the current UML Root package; this is safe as long as we
listen to when it is deleted
        protected UMLData m_RootPackage = null;

        // attach this eventhandler to the root-package of the (current) document
        public void Attach(IDocument iDoc)
        {
            if (m_RootPackage == null && iDoc != null && iDoc.RootPackage != null)
            {
                // hold a reference to the current UML Root package
                m_RootPackage = (UMLData)iDoc.RootPackage;

                // ensure we get 'OnAfterAddChild' events for *any* added child of the
rootpackage
                // (added to the root-package or one of its children)
                m_RootPackage.EventFilter = (int)
ENUMUMLDataEventFilter.eUMLDataEvent_AddChildOrGrandChild;
                // ensure we get informed when m_RootPackage (and only itself; we do not
care about its chidlren) is deleted
                m_RootPackage.EventFilter |= (int)
ENUMUMLDataEventFilter.eUMLDataEvent_EraseData;

                // we are only interested in 'OnAfterAddChild' and 'OnBeforeErase' events
so use and connect the delegates
                m_RootPackage.OnAfterAddChild += new
_IUMLDataEvents_OnAfterAddChildEventHandler(OnAfterAddChild);
                m_RootPackage.OnBeforeErase += new
_IUMLDataEvents_OnBeforeEraseEventHandler(OnBeforeErase);
            }
        }

        // detach eventhandler from the current UML Root package
        public void Detach()
        {
            if (m_RootPackage != null)
            {
                m_RootPackage.OnAfterAddChild -= OnAfterAddChild;
                m_RootPackage.OnBeforeErase -= OnBeforeErase;
                m_RootPackage = null;

                // release unused objects
```

```
                GC.Collect();
            }
        }

        #region _IUMLDataEvents Members

        public void OnAfterAddChild(IUMLData ipUMLParent, IUMLData ipUMLChild)
        {
            if (ipUMLParent == null || ipUMLChild == null)
                return;

            Debug.WriteLine("UModelDataEvents.OnAfterAddChild " + GetName(ipUMLChild) + "
to " + GetName(ipUMLParent));

           // verify if newly added child is of interesting kind:
            bool bIsEnumerationLiteral = (ipUMLChild is IUMLEnumerationLiteral);
            bool bIsProperty = (ipUMLChild is IUMLProperty);

            if (bIsProperty || bIsEnumerationLiteral)
            {
                try
                {
                    // check if child was added by undo/redo
                    // (we are not allowed to modify anything during Undo/Redo !!)
                    IDocument iDoc = (IDocument)ipUMLChild.Parent;
                    if (!iDoc.IsInUndoRedo)
                    {
                        // we only make one single modification here
                        // no need to use iDoc.BeginModification / iDoc.EndModification
in this case

                        // get the wanted prefix for the element kind
                        string sPrefix = null;

                        if (bIsProperty)
                            sPrefix = "m_";
                        if (bIsEnumerationLiteral)
                            sPrefix = "k_";

                        IUMLNamedElement iNamedChild = (IUMLNamedElement)ipUMLChild;

                        // set prefix only if not already set:
                        if (sPrefix != null && !iNamedChild.Name.StartsWith(sPrefix))
                        {
                            // use SetName (instead of Name) that UModel automatically
generates a valid, unique name starting with 'sPrefix + iNamedChild.Name'
                            iNamedChild.SetName(sPrefix + iNamedChild.Name);
                        }
                    }
                }
                catch (System.Exception e)
                {
                    Debug.WriteLine("EXCEPTION: " + e.Message);
                }
            }

            // release unused objects
            GC.Collect();
        }
```

```
        public void OnBeforeErase(IUMLData ipUMLData)
        {
            if (ipUMLData != null && m_RootPackage != null &&
ipUMLData.IsSameUMLData((IUMLData)m_RootPackage)) // should always be
            {
                // Detach ourself, since the UML data of m_RootPackage has been deleted
in UModel and we may not access it anymore
                Detach();
            }
        }

        public void OnChanged(IUMLData ipUMLData, string strHint)
        {
            // unused
        }

        public void OnMoveData(IUMLData ipUMLParent, IUMLData ipUMLChild, bool bAttach)
        {
            // unused
        }

        #endregion

        protected string GetName(IUMLData iUMLData)
        {
            if (iUMLData is IUMLNamedElement)
                return ((IUMLNamedElement)iUMLData).Name;

            return "";
        }
    }
}
```

17.4.4.6 "Statistics" Sample

The "Statistics" sample listens for data modifications and counts elements of different element kinds. The sample uses both the UModel API and the UModel IDE Plug-In library. Since the plug-in derives from `System.Windows.Forms.UserControl`, it also acts as an ActiveX control and the results can be shown in a custom window inside UModel:



This code is available in the following file: **..\UModelExamples\IDEPlugIn\StatisticsActiveX\StatisticsActiveX.cs**.

To build and run the sample, the same requirements as for other UModel IDE Plug-ins apply, see Build and Run the Plug-In.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Runtime.InteropServices.ComTypes;
using System.Windows.Forms;
using UModelLib;
using UModelPlugInLib;

/*
 * StatisticsActiveX sample
 * listen for data modifications and count the elements of the different element kinds
 * show the result in a listview of an ActiveX control
 */
namespace StatisticsActiveX
{
    /* StatisticsActiveX is the main class of this plugin and implements
```

```
UModelPlugInLib.IUModelPlugIn
     * it is also responsible for attaching/detaching _IApplicationEvents and
_ITransactionEvents
     */
    public partial class StatisticsActiveX : UserControl,
                                             IUModelPlugIn,
                                            _IApplicationEvents,
                                            _ITransactionEvents
    {
        // a sorted dictionary to count the different element kinds
        private Statistics m_Statistics;
        // reference to the transaction notifier of a UModel document
        private TransactionNotifier m_TransactionNotifier;
        // connection point to _IApplicationEvents
        private IConnectionPoint m_cpApplicationEvents = null;
        // connection cookie
        int m_nApplicationtEventsCookie = 0;

        public StatisticsActiveX()
        {
            InitializeComponent();
        }

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "PlugIn with ActiveX;This Plug-in demonstrates how to show an ActiveX
control inside UModel.";
        }

        public string GetUIModifications()
        {
            // We don't add any menu or toolbar modifications.
            return "<ConfigurationData><Modifications/></ConfigurationData>";
        }

        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized
            // and we can attach to get _IApplicationEvents

            IApplication iApp = (IApplication)pUModel;

            if (m_cpApplicationEvents == null && iApp != null)
            {
                // find connection point of _IApplicationEvents
                IConnectionPointContainer icpc = (IConnectionPointContainer)iApp;
                Guid IID = typeof(UModelLib._IApplicationEvents).GUID;
                icpc.FindConnectionPoint(ref IID, out m_cpApplicationEvents);

                // advise UModelApplicationEvents as sink for _IApplicationEvents
                m_cpApplicationEvents.Advise(this, out m_nApplicationtEventsCookie);
            }
```

```
            AttachTransactionEvents(iApp.ActiveDocument);
        }

        public void OnShutdown(object pUModel)
        {
            // detach application events; stop receiving events
            DetachTransactionEvents();

            if (m_cpApplicationEvents != null)
            {
                // terminate established connection to _IApplicationEvents
                m_cpApplicationEvents.Unadvise(m_nApplicationtEventsCookie);
                m_cpApplicationEvents = null;
            }

            // application will shutdown; release all unused objects
            GC.Collect();
        }

        public void OnCommand(int nID, object pUModel)
        {
            // unused; we did not add any menu- or toolbar-commands
        }

        public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
        {
            // unused; we did not add any menu- or toolbar-commands
            return UModelUpdateAction.UModelUpdateAction_Disable;
        }

        #endregion

        private void AttachTransactionEvents(IDocument iDoc)
        {
            if (iDoc != null)
            {
                m_TransactionNotifier = iDoc.TransactionNotifier;
                if (m_TransactionNotifier != null)
                {
                    // we are only interested in 'OnEndDataModification' events so use
and connect the delegate
                    m_TransactionNotifier.OnEndDataModification += new
_ITransactionEvents_OnEndDataModificationEventHandler(OnEndDataModification);
                }
            }

            UpdateStatistics(iDoc);
        }

        // detach eventhandler from the transaction notifier
        private void DetachTransactionEvents()
        {
            if (m_TransactionNotifier != null)
            {
                m_TransactionNotifier.OnEndDataModification -= OnEndDataModification;
                m_TransactionNotifier = null;
            }
            UpdateStatistics(null);
```

```
        }

        void UpdateStatistics(IDocument iDoc)
        {
            // count current elements
            Statistics statistics = new Statistics();

            if (iDoc != null && iDoc.RootPackage != null)
                CountElements(iDoc.RootPackage, ref statistics);

            // anything changed to last update ?
            if (!statistics.IsEqual(m_Statistics))
            {
                m_Statistics = statistics;
                PopulateListView(m_Statistics);
            }

            // release unused objects
            GC.Collect();
        }

        private void CountElements(IUMLElement iElem, ref Statistics statistics)
        {
            // we only count editable elements
            if (iElem == null || iElem.IsEditable == false)
                return;

            string sKindName = iElem.KindName;

            if (!statistics.ContainsKey(sKindName))
                statistics[sKindName] = 1;
            else
                statistics[sKindName]++;

            foreach (IUMLElement iChild in iElem.OwnedElements)
                CountElements(iChild, ref statistics);
        }

        private void PopulateListView(Statistics statistics)
        {
            listView1.BeginUpdate();

            listView1.Items.Clear();
            foreach (KeyValuePair<string, int> kvp in statistics)
            {
                ListViewItem item = new ListViewItem(kvp.Key);
                item.SubItems.Add(Convert.ToString(kvp.Value));

                listView1.Items.Add(item);
            }

            listView1.EndUpdate();
        }

        #region _ITransactionEvents Members

        public void OnBeginDataModification(Document ipDocument)
        {
            // begin of transaction
```

```
        }

        public void OnEndDataModification(Document ipDocument)
        {
            // end of transaction - update statistics
            if (ipDocument != null && ipDocument.TransactionNotifier ==
m_TransactionNotifier)
                UpdateStatistics(ipDocument);
        }

        #endregion

        #region _IApplicationEvents Members

        public void OnNewDocument(Document ipDocument)
        {
            // a new document has been created in UModel => (re-)connect transaction
events
            AttachTransactionEvents(ipDocument);
        }

        public void OnDocumentOpened(Document ipDocument)
        {
            // a document has been opened in UModel => (re-)connect transaction events
            AttachTransactionEvents(ipDocument);
        }

        public void OnDocumentClosed(Document ipDocument)
        {
            // document has been closed in UModel => disconnect transaction events
            if (ipDocument != null && ipDocument.TransactionNotifier ==
m_TransactionNotifier)
                DetachTransactionEvents();
        }

        public void OnShutdown()
        {
        }

        #endregion


        #region Statistics dictionary

        private class Statistics : SortedDictionary<string, int>
        {
            public bool IsEqual(Statistics other)
            {
                if (other == null)
                    return false;

                if (Count != other.Count)
                    return false;

                Enumerator e1 = GetEnumerator();
                Enumerator e2 = other.GetEnumerator();
                while (e1.MoveNext() && e2.MoveNext())
                {
                    if ((e1.Current.Key != e2.Current.Key) ||
```

```
                        (e1.Current.Value != e2.Current.Value))
                        return false;
                }

                return true;
            }
        }

        #endregion
    }
}
```

17.4.5 Java API Example

The UModel installation package contains an example Java project, located at **C:\Users\<username>\Documents\Altova\UModel2023\UModelExamples\API**. This folder contains Java examples for the UModel API. You can test it directly from the command line using the batch file `BuildAndRun.bat`, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

The Java examples folder contains all the files required to run the example project. These files are listed below:

AltovaAutomation.dll	Java-COM bridge: DLL part
AltovaAutomation.jar	Java-COM bridge: Java library part
UModelAPI.jar	Java classes of the UModel API
RunUModel.java	Java example source code
BuildAndRun.bat	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
.classpath	Eclipse project helper file
.project	Eclipse project file
UModelAPI_JavaDoc.zip	`Javadoc` file containing help documentation for the Java API
Readme.txt	This file

The example starts up UModel and performs a few operations, including opening and closing documents. When done, UModel stays open. You must close it manually.

Running the example from the command line

To run the example from the command line, open a command prompt window, go to the Java folder of the API Examples folder (*see above for location*), and then type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

The Java binary folder must be that of a JDK 1.7 or later installation on your computer.

Press the **Return** key. The Java source in `RunUModel.java` will be compiled and then executed.

Loading the example in Eclipse

Open Eclipse and use the **File | Import... | General | Existing Projects into Workspace** command to add the Eclipse project file (`.project`) located in the Java folder of the API Examples folder (*see above for location*). The project `RunUModel` will then appear in your Package Explorer or Navigator.

Select the project and then the command **Run as | Java Application** to execute the example.

Note: You can select a class name or method of the Java API and press F1 to get help for that class or method.

17.4.6 JScript Examples

This section contains listings of JScript code that demonstrate the following basic functionality:

- Start application
- Document Access
- Generate documentation
- Generate code
- Update Documentation

Example files

The code listings in this section are available in example files that you can test as is or modify to suit your needs. The JScript example files are located at **C:\Users\<username>\Documents\Altova\UModel2023\UModelExamples\API**.

The example files can be run in one of two ways:

- *From the command line:* Open a command prompt window, change the directory to the path above, and type the name of one of the example scripts (for example, `Start.js`).
- *From Windows Explorer:* In Windows Explorer, browse for the JScript file and double-click it.

The script is executed by Windows Script Host that is packaged with Windows operating system. For more information about Windows Script Host, refer to MSDN documentation (https://msdn.microsoft.com).

17.4.6.1 Start application

The JScript below starts the application and shuts it down. If an instance of the application is already running, the running instance will be returned.

Note: For 32-bit UModel, the registered name, or programmatic identifier (ProgId) of the COM object is `UModel.Application`. For 64-bit UModel, the name is `UModel_x64.Application`.

This code is available in the sample file `..\UModelExamples\API\JScript\Start.js` (see also Example Files).


```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, a the main COM object of an
already
// running application might be returned.
try {   objUModel = WScript.GetObject("", "UModel.Application");   }
catch(err) {}

if( typeof( objUModel ) == "undefined" )
{
   try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
   catch(err)
   {
      WScript.Echo( "Can't access or create UModel.Application" );
      WScript.Quit();
   }
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objUModel.Visible = true;

WScript.Echo(objUModel.Edition + " has successfully started. ");

objUModel.Visible = false; // will shutdown application if it has no more COM connections
//objUModel.Visible = true;   // will keep application running with UI visible
```

17.4.6.2 Document Access

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and close documents.

This code is available in the sample file `..\UModelExamples\API\JScript\DocumentAccess.js` (see also Example Files).

```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, a the main COM object of an
already
// running application might be returned.
try {   objUModel = WScript.GetObject("", "UModel.Application");   }
catch(err) {}

if( typeof( objUModel ) == "undefined" )
{
   try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
   catch(err)
   {
      WScript.Echo( "Can't access or create UModel.Application" );
      WScript.Quit();
   }
}
```

```
// if newly started, the application will start without its UI visible. Set it to
visible.
objUModel.Visible = true;

// ************************** code snippet for "Simple Document Access"
**********************

// Locate examples using property PersonalDataDirectory
objDoc = objUModel.OpenDocument(objUModel.PersonalDataDirectory + "\\UModelExamples\
\Bank_MultiLanguage.ump");
// open all diagrams
objDoc.OpenAllDiagrams();

// ************************** code snippet for "Simple Document Access"
**********************

// ************************** code snippet for "Iteration"
***********************************

objName = "";
count = 0;
// go through all open diagrams using a JScript Enumerator
for (var iterDiagrams = new Enumerator(objDoc.DiagramWindows); !iterDiagrams.atEnd();
iterDiagrams.moveNext())
{
   objName += "\t" + ++count + " " + iterDiagrams.item().Name + "\n";
}

WScript.Echo("Opened diagrams: \n" + objName);

// go through all open diagrams using index-based access to the document collection
for (i = objDoc.DiagramWindows.Count; i > 0; i--)
   objDoc.DiagramWindows.Item(i).Close();


// ************************** code snippet for "Iteration"
***********************************

//objUModel.Visible = false;      // will shutdown application if it has no more COM
connections
objUModel.Visible = true;   // will keep application running with UI visible
```

17.4.6.3 Generate Documentation

The JScript listing below shows how to generate documentation for the **Bank_MultiLanguage.ump** file in the UModelExamples folder.

This code is available in the sample file `..\UModelExamples\API\JScript\GenerateDoc.js` (see also Example Files).

```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, a the main COM object of an
```

```
already
// running application might be returned.
try {   objUModel = WScript.GetObject("", "UModel.Application");   }
catch(err) {}

if( typeof( objUModel ) == "undefined" )
{
   try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
   catch(err)
   {
      WScript.Echo( "Can't access or create UModel.Application" );
      WScript.Quit();
   }
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objUModel.Visible = true;

// Locate examples via USERPROFILE shell variable.
objWshShell = WScript.CreateObject("WScript.Shell");
majorVersionYear = objUModel.MajorVersion + 1998
strExamplesFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\Documents\
\Altova\\UModel" + majorVersionYear + "\\UModelExamples\\";

objDoc = objUModel.OpenDocument(strExamplesFolder + "Bank_MultiLanguage.ump");

// generate documentation
dlgs = objUModel.Dialogs;
docDlg = dlgs.GenerateDocumentationDlg;
docDlg.OutputFormat = 0; // ENUMDocumentationOutputFormat.eDocumentationOutputFormat_HTML

var myObject = new ActiveXObject("Scripting.FileSystemObject");
strDocOutputFolder = strExamplesFolder + "GeneraredDocFromJScriptExample\\";

if (!myObject.FolderExists(strDocOutputFolder))
   myObject.CreateFolder(strDocOutputFolder);

strResultFile = strDocOutputFolder + "Bank_MultiLanguage.html";
objDoc.generateDocumentation(docDlg, strResultFile);

//objUModel.Visible = false;      // will shutdown application if it has no more COM
connections
objUModel.Visible = true;   // will keep application running with UI visible
```

17.4.6.4 Generate Code

The following JScript sample creates a new UModel project, creates some classes and generates code.

This code is available in the sample file `..\UModelExamples\API\JScript\UModelCreateCode.js` (see Example Files).

```
// ##########################################
// access runing UModel.Application or
// launch new one and access it
// ##########################################
```

```
// ##########################################
// CreateCode sample
// shows forward engineering from scratch
// it creates some coding elements in a new UModel project and generates code (saving the
project afterwards)
// ##########################################



// ############ global variables #################
var objUModel = null;
var objWshShell = null;
var objFSO = null;

// ###################### Helpers ##############################

function Exit(strErrorText)
{
   WScript.Echo(strErrorText);

   if (objUModel != null)
      objUModel.Quit();

   WScript.Quit(-1);
}

function CreateGlobalObjects ()
{
   // the Shell and FileSystemObject of the windows scripting host often always useful
   try
   {
      objWshShell = WScript.CreateObject("WScript.Shell");
      objFSO = WScript.CreateObject("Scripting.FileSystemObject");
   }
   catch(err)
   {
      Exit("Can't create WScript.Shell object");
   }

   // create the UModel connection
   // if there is a running instance of UModel (that never had a connection) - use it
   // otherwise, we automatically create a new instance
   try {   objUModel = WScript.GetObject("", "UModel.Application");   }
   catch(err) {}

   if( typeof( objUModel ) == "undefined" )
   {
      try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
      catch(err)
      {
         objUModel = null;
         Exit( "Can't access or create UModel.Application" );
      }
   }
}

function GetSourceCodeDirectory()
```

```
{
   // get directory for source code
   var path =  objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\JScript\
\CreateCode";
   var codeDirectory = objFSO.BuildPath( path, "SampleCode" );
   return codeDirectory;
}

function GetUMPFilePath()
{
   // get file path to save UModel projectfile
   return objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\JScript\\CreateCode\
\CreateCode.ump";
}

function IncludeCSharpProfile( objDocument )
{
   try
   {
      // get dialog for including subprojects:
      var objIncludeSubProjectDialog = objUModel.Dialogs.IncludeSubprojectDlg;

      objIncludeSubProjectDialog.ProjectFile = objUModel.InstallationDirectory + "\
\UModelInclude\\c# Profile.ump";

      return objDocument.IncludeSubproject( objIncludeSubProjectDialog );
   }
   catch(err)
   {
      Exit("Can't include CSharp profile");
   }
}

// ////////////////////// MAIN ////////////////////////////

CreateGlobalObjects();

objUModel.Visible = true;

// open a new, empty document
var objDocument = objUModel.NewDocument();
// get the root-package
var objRootPackage = objDocument.RootPackage;

if ( objDocument    != null &&
     objRootPackage != null &&
    IncludeCSharpProfile( objDocument ) )
{
   // create coding elements
    try
    {
        // make all modifications within one UndoStep; start modification here
      if ( !objDocument.BeginModification() )
         Exit("No modifications allowed");

      // create a namespace root package
      var objCSharpRootNamespace = objRootPackage.InsertPackagedElementAt( -1,
"Package" );
      objCSharpRootNamespace.SetName( "CSharp" );
```

```
	// find C# Profile...
	var objCSharpProfile = objRootPackage.FindPredefinedOwnedElement( 159, false );//
ePredefined_CSharp_Profile = 159,
	// ...and apply it to the package, which is now a CSharp namespace root
	objCSharpRootNamespace.InsertProfileApplicationAt( -1, objCSharpProfile );


	// create a C# namespace package...
	var objCSharpNamespace = objCSharpRootNamespace.InsertPackagedElementAt( -1,
"Package" );
	objCSharpNamespace.SetName( "Namespace1" );
	// ... and apply the predefined C# namespace stereotype
	objCSharpNamespace.ApplyPredefinedStereotype( 223 ); //
ePredefined_CSharp_namespaceStereotypeOfPackage = 223,

	// create new class within the C# namespace
	var objClass     = objCSharpNamespace.InsertPackagedElementAt( -1, "Class" );
	var objClass2    = objCSharpNamespace.InsertPackagedElementAt( -1, "Class" );
	var objBaseClass = objCSharpNamespace.InsertPackagedElementAt( -1, "Class" );
	objClass    .SetName( "MyClass"     );
	objClass2   .SetName( "MyClass2"    );
	objBaseClass.SetName( "MyBaseClass" );

	// set attribute-section "STAThread"
	var objAttributesStereotypeApplication =
objClass.ApplyPredefinedStereotype( 191 );//
ePredefined_CSharp_attributesStereotypeOfClass = 191
	var objSTAThread = objCSharpProfile.FindPredefinedOwnedElement( 185, true ); //
ePredefined_CSharp_AttributePresetsEnumeration_STAThreadEnumerationLiteral = 185
	objAttributesStereotypeApplication.SetPredefinedTaggedValueAt(-1, 192,
objSTAThread.Name); // ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty =
192

	// insert new attribute
	var objProperty = objClass.InsertOwnedAttributeAt( -1 );
	objProperty.SetName( "m_Att" );
	objProperty.Visibility = 2;      // eVisibility_Private = 2
	objProperty.Type = objClass2;

	// insert new operation
	var objOperation = objClass.InsertOwnedOperationAt( -1 );
	objOperation.SetName( "GetAtt" );
	objOperation.Type = objClass2;

	// derive MyClass from MyBaseClass
	objClass.InsertGeneralizationAt( -1, objBaseClass );


	// find the component view package
	var objComponentView = objRootPackage.FindPredefinedOwnedElement( 1, false );//
ePredefined_ComponentViewPackage = 1

	// create a new component for C# 3.0 and set the source code directory, where we
want to generate the source code
	var objComponent = objComponentView.InsertPackagedElementAt( -1, "Component" );
	objComponent.CodeLangVersion= 5; //    eCodeLang_CSharp_3_0   = 5,
	objComponent.CodeProjectFileOrDirectory = GetSourceCodeDirectory();
	objComponent.IsCodeProjectFile = false;
```

```
      // this component should realize our classes:
      objComponent.InsertRealizationAt( -1, objClass     );
      objComponent.InsertRealizationAt( -1, objClass2    );
      objComponent.InsertRealizationAt( -1, objBaseClass );

        // do not forget to end modification and finish UndoStep
      objDocument.EndModification();
    }
   catch( err )
   {
        // rollback made changes
      objDocument.AbortModification();
      Exit("Error when creating UML model elements");
   }

   // update code from model
   try
   {
      // explicitely run a syntax check
      if ( objDocument.CheckProjectSyntax() )
      {
         // get dialog for code <=> model synchonizations and set the wanted options:
         var objSynchronizationSettingsDlg =
objUModel.Dialogs.SynchronizationSettingsDlg;

         objSynchronizationSettingsDlg.CodeFromModel_Synchronization = 0; //
eSynchronization_Merge = 0
         objSynchronizationSettingsDlg.CodeFromModel_UserDefinedSPLTemplatesOverrideDefau
lt = true;

         // update code from model
         if ( !objDocument.SynchronizeCodeFromModel( objSynchronizationSettingsDlg ) )
            Exit("Update code from model failed");
      }
      else
         Exit("Syntax check failed");
   }
   catch( err )
   {
      Exit("Error when updating code from model");
   }

   // save project
   objDocument.SaveAs( GetUMPFilePath() );

   WScript.Echo("Finished successfully");
}

// if something went wrong (and we did not save the project),
// we also do not want get asked for saving => set ModifiedFlag to false
if ( objDocument != null )
   objDocument.ModifiedFlag = false;

objUModel.Visible = false;   // will shutdown application if it was started by this
script
```

17.4.6.5 Update Documentation

The following JScript sample, when running for the first time, reverse engineers all UModel API C# samples found in the **..\UModelExamples\IDEPlugIn** directory and creates HTML and RTF documentation as well as an XMI export of the UModel project. The resulting UMP files, as well as the generated documentation output, are saved to the **..\UModelExamples\API\JScript\UpdateDocumentation** directory. On subsequent runs, it opens the previously generated UModel project files, and creates HTML and RTF documentation, as well as XMI export, provided that something has changed in the UML model.

This code is available in the sample file `..\UModelExamples\API\JScript\UModelUpdateDocumentation.js` (see Example Files).

```
// ##########################################
// access runing UModel.Application or
// launch new one and access it
// ##########################################


// ##########################################
// UpdateDocumentation sample
// *) When running the first time (= when no UMP file exists), reverse engineer all C#
UModelAPI samples
//    and create HTML and RTF documentation, make XMI export and save UMP file
// *) when UMP file already exists, open it and synchronize model from code
//    create HTML and RTF documentation and XMI export only if something has been changed
(listen to all different UML data events)
// ##########################################

var bRunVisible = true;
var bShowDialogs = bRunVisible && false;

// ////////////// global variables /////////////////
var objUModel   = null;
var objWshShell = null;
var objFSO      = null;

var bChangedAnything = false;
var nAddedClasses    = 0;
var nAddedInterfaces= 0;
var nAddedProperties= 0;
var nAddedOperations= 0;

// ////////////////////// Helpers ////////////////////////////////

function Exit(strErrorText)
{
   WScript.Echo(strErrorText);

   if (objUModel != null)
      objUModel.Quit();

   WScript.Quit(-1);
}

function CreateGlobalObjects ()
{
```

```
   // the Shell and FileSystemObject of the windows scripting host often always useful
   try
   {
      objWshShell = WScript.CreateObject("WScript.Shell");
      objFSO = WScript.CreateObject("Scripting.FileSystemObject");
   }
   catch(err)
   {
      Exit("Can't create WScript.Shell object");
   }

   // create the UModel connection
   // if there is a running instance of UModel (that never had a connection) - use it
   // otherwise, we automatically create a new instance
   try {   objUModel = WScript.GetObject("", "UModel.Application");   }
   catch(err) {}

   if( typeof( objUModel ) == "undefined" )
   {
      try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
      catch(err)
      {
         objUModel = null;
         Exit( "Can't access or create UModel.Application" );
      }
   }
}

// ////////////////////// get different filepathes / ensure folders are
created  ////////////////////////////
function GetScriptPath()
{
   var path = objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\JScript\
\UpdateDocumentation";

   if ( !objFSO.FolderExists( path ) )
      objFSO.CreateFolder( path );

    return path;
}

function GetFilePath( subdir, filename )
{
   var path = objFSO.BuildPath( GetScriptPath(), subdir );

   if ( !objFSO.FolderExists( path ) )
      objFSO.CreateFolder( path );

   return path + "\\" + filename;
}

function GetUMPFilePath ()   { return GetFilePath( "UMP",      "UModelAPI.ump" ); }
function GetXMIFilePath ()   { return GetFilePath( "Output_XMI", "UModelAPI.xmi" ); }
function GetHTMLFilePath()   { return GetFilePath( "Output_HTML","UModelAPI.html"); }
function GetRTFFilePath ()   { return GetFilePath( "Output_RTF", "UModelAPI.rtf" ); }

// ////////////////////// UML data event handlers ////////////////////////////
function objRootPackage_OnChanged( objData, strHint )
{
```

```
   bChangedAnything = true;
}

// recursively count newly added classes, interfaces, properties and operations
function CountAddedElements( objNewChild )
{
   if ( objNewChild != null )
   {
      if ( objNewChild.KindName == "Class"    ) ++nAddedClasses;
      if ( objNewChild.KindName == "Interface" ) ++nAddedInterfaces;
      if ( objNewChild.KindName == "Property"  ) ++nAddedProperties;
      if ( objNewChild.KindName == "Operation" ) ++nAddedOperations;

      var ownedElements = objNewChild.OwnedElements;
      var itr = new Enumerator( ownedElements );
      for ( ; !itr.atEnd(); itr.moveNext() )
         CountAddedElements( itr.item() );
   }
}

function objRootPackage_OnAfterAddChild( objParent, objNewChild )
{
    bChangedAnything = true;

   // recursively count newly added classes, interfaces, properties and operations
   CountAddedElements( objNewChild );
}

function objRootPackage_OnBeforeErase( objData )
{
    bChangedAnything = true;
}

function objRootPackage_OnMoveData( objParent, objChild, bAttach )
{
    bChangedAnything = true;
}

// ////////////////////// MAIN ////////////////////////////

CreateGlobalObjects();

if ( bRunVisible )
   objUModel.Visible = true;

var objDocument = null;

try
{
   // open document if it exists; create new one otherwise
   var bDocumentExisted = false;

   if ( objFSO.FileExists( GetUMPFilePath() ) )
   {
      objDocument = objUModel.OpenDocument( GetUMPFilePath() );
      bDocumentExisted = true;
   }
   else
   {
```

```
      objDocument = objUModel.NewDocument();
      objDocument.SaveAs( GetUMPFilePath() );
   }

   if ( objDocument == null )
      Exit( "Cannot create or open UModel projectfile" );

   // connect to receive _IUMLDataEvents from the root-package and all its children:
   var objRootPackage = objDocument.RootPackage;
    WScript.ConnectObject (objRootPackage, "objRootPackage_" );

   // ensure we get *all* events from root-package and *all* children:
   objRootPackage.EventFilter = 2 + // eUMLDataEvent_EraseDataOrChild      = 2,
                          8 + // eUMLDataEvent_AddChildOrGrandChild   = 8,
                         32 + // eUMLDataEvent_ChangeDataOrChild      = 32,
                         128; // eUMLDataEvent_MoveChildOrGrandChild   = 128
   if ( bDocumentExisted )
   {
      // UModel projectfile already exists => update model from code

      // get dialog for code <=> model synchonizations and set the wanted options:
      var objSynchronizationSettingsDlg = objUModel.Dialogs.SynchronizationSettingsDlg;
      objSynchronizationSettingsDlg.ShowDialog = bShowDialogs;

      objSynchronizationSettingsDlg.ModelFromCode_Synchronization = 0; //
eSynchronization_Merge = 0

      // update model from code
      if ( !objDocument.SynchronizeModelFromCode( objSynchronizationSettingsDlg ) )
         Exit("Update model from code failed");
   }
   else
   {
      // UModel projectfile did not exist => newly import code into model

      var objImportSourceDirectoryDlg = objUModel.Dialogs.ImportSourceDirectoryDlg;
      objImportSourceDirectoryDlg.ShowDialog = bShowDialogs;

      // set source code directory to import
      objImportSourceDirectoryDlg.Directory = objUModel.PersonalDataDirectory + "\
\UModelExamples\\IDEPlugIn";
      objImportSourceDirectoryDlg.ProcessSubdirectories = true;
      // set source code language to import (C# 3.0)
      objImportSourceDirectoryDlg.Language      = 5; // eCodeLang_CSharp_3_0      = 5
      objImportSourceDirectoryDlg.Synchronization = 0; // eSynchronization_Merge = 0
      // import in a new package
      objImportSourceDirectoryDlg.ImportInNewPackage = true;

      objImportSourceDirectoryDlg.DiagramGeneration = true;

      // content diagram generation settings
      objImportSourceDirectoryDlg.Content_GenerateSingleDiagram            = true;
      objImportSourceDirectoryDlg.Content_GenerateDiagramPerPackage         = true;
      objImportSourceDirectoryDlg.Content_ShowNestedClassifiersSeparately      = false;
      objImportSourceDirectoryDlg.Content_ShowAnonymousBoundElements         = false;
      objImportSourceDirectoryDlg.Content_HyperlinkPackagesToDiagrams         = true;
      objImportSourceDirectoryDlg.Content_ShowAttributesCompartment         = true;
      objImportSourceDirectoryDlg.Content_ShowOperationsCompartment         = true;
      objImportSourceDirectoryDlg.Content_ShowNestedClassifiersCompartment   = false;
```

```
      objImportSourceDirectoryDlg.Content_ShowEnumerationLiteralsCompartment   = true;
      objImportSourceDirectoryDlg.Content_ShowTaggedValues                     = true;
      objImportSourceDirectoryDlg.Content_Autolayout                           = 1; //
eDiagramLayout_Hierarchic = 1
      // open diagrams that autolayout is done:
      objImportSourceDirectoryDlg.Content_OpenDiagrams                        = true;

      // package dependency diagram generation settings (disabled)
      objImportSourceDirectoryDlg.PackageDependency_GenerateDiagram = false;

      // import source directory
      if ( !objDocument.ImportSourceDirectory( objImportSourceDirectoryDlg) )
      {
         // also delete newly created (empty) UMP file that source code directory import
is retried the next time
         objFSO.DeleteFile( GetUMPFilePath() );
         Exit( "Error on importing source directory" );
      }
   }

   // disconnect from getting root-package events
   WScript.DisconnectObject( objRootPackage );
}
catch( err )
{
   // also delete newly created (empty) UMP file that source code directory import is
retried the next time
   objFSO.DeleteFile( GetUMPFilePath() );
   Exit( "Error on importing source directory" );
}

//if something has changed, update the outputs:
if ( bChangedAnything )
{
   try
   {
      // make XMI export for UML2.1.2
      var objIExportXMIFileDlg = objUModel.Dialogs.ExportXMIFileDlg;
      objIExportXMIFileDlg.ShowDialog = bShowDialogs;
      objIExportXMIFileDlg.XMIFile            = GetXMIFilePath();
      objIExportXMIFileDlg.PrettyPrintXMIOutput   = true;
      objIExportXMIFileDlg.ExportUUIDs         = true;
      objIExportXMIFileDlg.ExportExtensions      = true;
      objIExportXMIFileDlg.ExportDiagrams         = true;
      objIExportXMIFileDlg.XMIType            = 1; // eXMI21ForUML212 = 1

      // export to XMI file:
      if ( !objDocument.ExportToXMIFile( objIExportXMIFileDlg ) )
      {
         // error on XMI generation
      }
   }
   catch( err )
   {
      // error on XMI generation
   }

   try
   {
```

```
      var objIDocumentationGenerationDlg = objUModel.Dialogs.GenerateDocumentationDlg;
      objIDocumentationGenerationDlg.ShowDialog = bShowDialogs;

      objIDocumentationGenerationDlg.GenerateLinksToLocalFiles = 1; //
eDocumentationFilePath_RelativeToResultFile   = 1
      objIDocumentationGenerationDlg.SplitOutputToMultipleFiles = true;
      objIDocumentationGenerationDlg.ShowResultFileAfterGeneration = true;

      objIDocumentationGenerationDlg.Details_SelectAll();
      // show up to 10 base class/interface hierarchies
      objIDocumentationGenerationDlg.Details_HierarchyDiagramNestingDepthUp = 10;
      // only show directly derived classes/interfaces
      objIDocumentationGenerationDlg.Details_HierarchyDiagramNestingDepthDown = 1;
      // keep hierarchy diagram as small as possible => expand each element only once
      objIDocumentationGenerationDlg.Details_HierarchyDiagramExpandItemsOnlyOnce = true;

      objIDocumentationGenerationDlg.Include_SelectAllDiagrams();
      objIDocumentationGenerationDlg.Include_SelectNoElements();
      objIDocumentationGenerationDlg.Include_Index = true;
      objIDocumentationGenerationDlg.Include_IncludedSubprojects = false;
      objIDocumentationGenerationDlg.Include_NamedElementsOnly = true;
      objIDocumentationGenerationDlg.Include_UnknownExternals = false;

      var objIncludeElements = objIDocumentationGenerationDlg.Include_Elements;
      var itrIncludeElements = new Enumerator( objIncludeElements );
      for ( ; !itrIncludeElements.atEnd(); itrIncludeElements.moveNext() )
      {
         var objElemSel = itrIncludeElements.item();

         if ( objElemSel.KindName == "Class"         ||
              objElemSel.KindName == "Interface"     ||
              objElemSel.KindName == "Enumeration"   ||
              objElemSel.KindName == "Operation"     ||
              objElemSel.KindName == "Package"      )
         {
            objElemSel.Selection = true;
         }
      }

      // generate HTML documentation (with PNG pictures)
      objIDocumentationGenerationDlg.OutputFormat = 0; // eDocumentationOutputFormat_HTML
  = 0
      objIDocumentationGenerationDlg.DiagramImageFormat = 0; // eOutputImageFormat_PNG =
0
      objIDocumentationGenerationDlg.EmbedDiagrams = false;
      if ( !objDocument.GenerateDocumentation( objIDocumentationGenerationDlg,
GetHTMLFilePath() ) )
      {
         // error on HTML documentation generation
      }

      // generate RTF documentation (with embeded EMF pictures)
      objIDocumentationGenerationDlg.ShowDialog = false; // don't show dialog again
      objIDocumentationGenerationDlg.OutputFormat = 2; // eDocumentationOutputFormat_RTF
  = 2
      objIDocumentationGenerationDlg.DiagramImageFormat = 1; // eOutputImageFormat_EMF =
1
      objIDocumentationGenerationDlg.EmbedDiagrams = true;
      if ( !objDocument.GenerateDocumentation( objIDocumentationGenerationDlg,
```

```
GetRTFFilePath() ) )
      {
         // error on RTF documentation generation
      }
   }
   catch( err )
   {
      // error on documentation generation
   }

   // show the number of newly added classes, interfaces, properties and operations
   if ( bRunVisible )
   {
      WScript.Echo( "Added classes: "       + nAddedClasses   +
                    "\nAdded interfaces: " + nAddedInterfaces +
                    "\nAdded properties: " + nAddedProperties +
                    "\nAdded operations: " + nAddedOperations );
   }
}
else
{
   if ( bRunVisible )
      WScript.Echo( "Nothing has changed" );
}

// always save document (although it's not really necessary when nothing has been
changed)
objDocument.Save();

if ( bRunVisible )
   objUModel.Visible = false;   // will shutdown application if it was started by this
script
```

17.5 UModel API Reference

This section describes all interfaces, operations, enumerations and events of UModel Plugins and of the UModel API.

17.5.1 UModel Plug-Ins

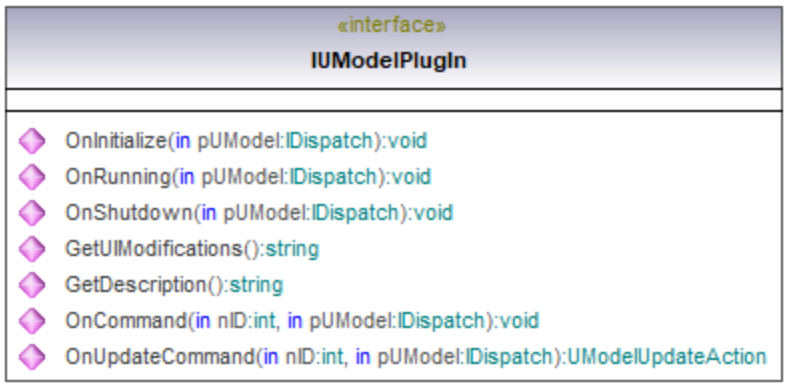
This section provides reference to the API interfaces required for integrating your own plug-ins into UModel. For conceptual information and instructions about creating UModel IDE plug-ins, see UModel IDE Plug-Ins.

For C# code samples illustrating plug-ins integrated into UModel, see the following topics:

- "Set Styles" Sample
- "C# Delegate" Sample
- "Set Prefix" Sample
- "Statistics" Sample

17.5.1.1 UModelAPI - IUModelPlugIn

Interface **IUModelPlugIn**

diagram	

Operation **IUModelPlugIn::GetDescription**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUModelPlugIn::GetUIModifications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUModelPlugIn::OnCommand**

parameter	name	direction	type	type modifier	multiplicity	default
	nID	**in**	**int**			
	pUModel	**in**	**IDispatch**			
	return	**return**	**void**			

Operation **IUModelPlugIn::OnInitialize**

parameter	name	direction	type	type modifier	multiplicity	default
	pUModel	**in**	**IDispatch**			
	return	**return**	**void**			

Operation **IUModelPlugIn::OnRunning**

parameter	name	direction	type	type modifier	multiplicity	default
	pUModel	**in**	**IDispatch**			
	return	**return**	**void**			

Operation **IUModelPlugIn::OnShutdown**

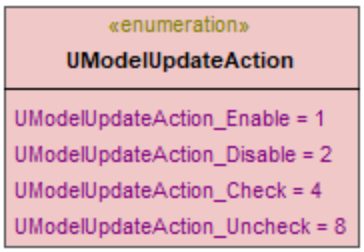
parameter	name	direction	type	type modifier	multiplicity	default
	pUModel	**in**	**IDispatch**			
	return	**return**	**void**			

Operation **IUModelPlugIn::OnUpdateCommand**

parameter	name	direction	type	type modifier	multiplicity	default
	nID	**in**	**int**			
	pUModel	**in**	**IDispatch**			
	return	**return**	**UModelUpdateAction**			

17.5.1.2 UModelAPI - UModelUpdateAction

Enumeration **UModelUpdateAction**

diagram	
typedElements	Interface **IUModelPlugIn** Operation **OnUpdateCommand**

17.5.2 UModel API Interfaces

This section provides reference to the objects of the UModel COM API. The objects are described in a generic manner, since the API may be used with virtually any language that supports calling a COM object. For language-specific examples, see:

- Example C# Project
- Example Java Project
- JScript Examples

The API reference contains two main sections, each describing the interfaces and the enumeration types used in the API, respectively. The enumeration values contain both the string name and a numeric value. If your scripting environment does not support enumerations, use the number-values instead.

In .NET, for every interface of the UModel COM automation interface, a .NET class exists with the same name. Also, COM types will be converted to the appropriate .NET type. For example, a type such as `Long` in the COM API would appear as `System.Int32` in .NET.

In Java, note the following syntax variations:

- **Classes and class names**. For every interface of the COM automation interface, a Java class exists with the name of the interface.
- **Method names**. Method names on the Java interface are the same as used on the COM interfaces, but start with a small letter to conform to Java naming conventions. To access COM properties, Java methods that prefix the property name with get and set can be used. If a property does not support write-access, no setter method is available. For example, for the `Name` property of the `Document` interface, the Java methods `getName` and `setName` are available.
- **Enumerations**. For every enumeration defined in the automation interface, a Java enumeration is defined with the same name and values.
- **Events and event handlers**. For every interface in the automation interface that supports events, a Java interface with the same name plus 'Event' is available. To simplify the overloading of single events, a Java class with default implementations for all events is provided. The name of this Java class is the name of the event interface plus 'DefaultHandler'. For example:

```
Application // Java class to access the application
ApplicationEvents // Events interface for the application
ApplicationEventsDefaultHandler // Default handler for "ApplicationEvents"
```

UModel API Errors

The UModel API may return the error codes listed below.

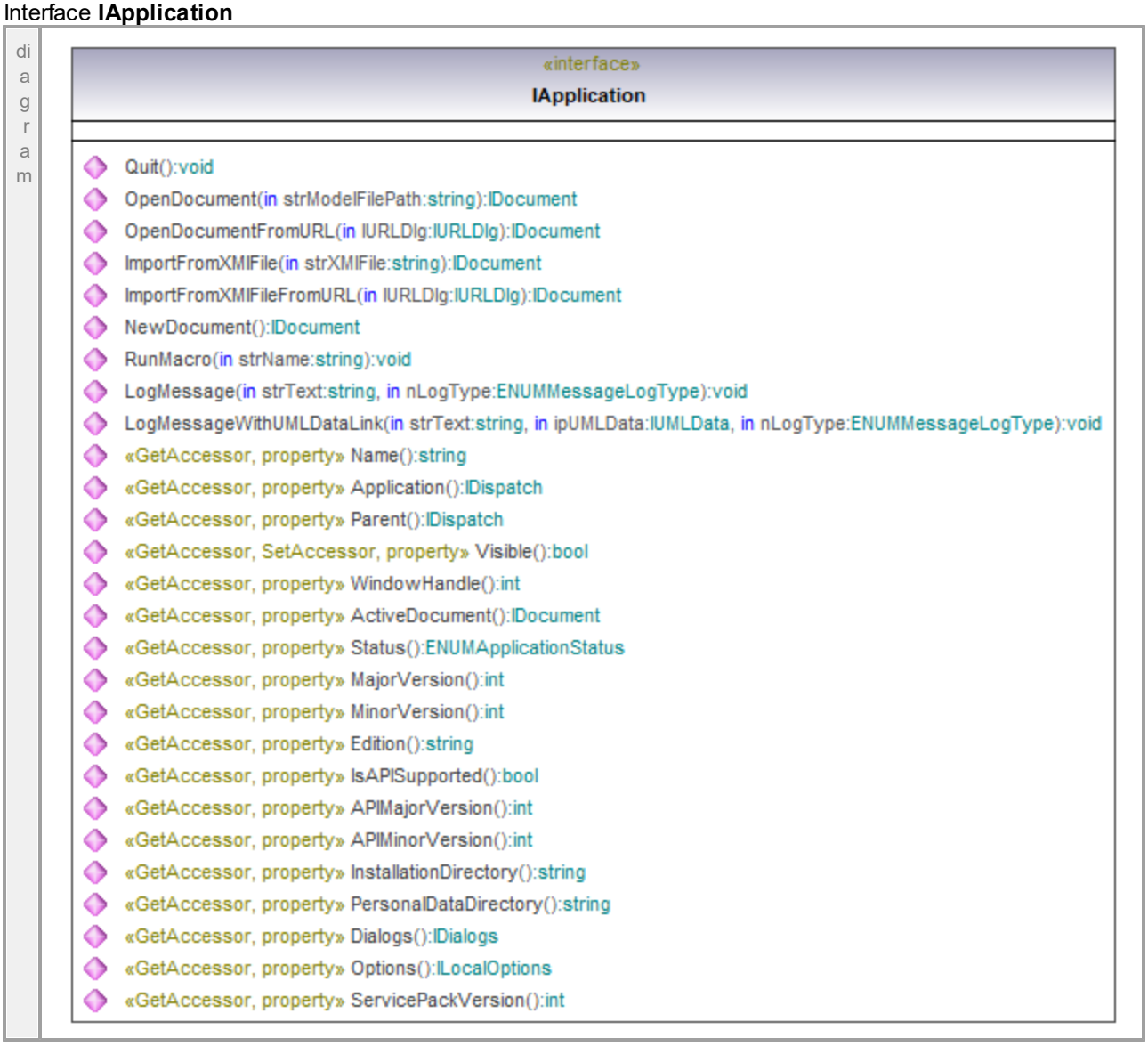
1000	The application object is no longer valid.
1001	Invalid parameter or invalid address for the return parameter was specified.
1002	UModel API is not available in the current edition.
1003	Model Transformations are not supported in the current edition.

1050	Macro not found
1051	Invalid (nested) macro execution
1100	Error when saving file, probably the file name is invalid.
1101	Invalid (duplicate) call to BeginModification.
1102	EndModification called without BeginModification
1200	Error deleting file at URL.
1201	Error creating directory at URL.

The `UMLData` interfaces have specific errors, see UMLData Interfaces.

17.5.2.1 UModelAPI - IApplication

Interface **IApplication**



Operation **IApplication::ActiveDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDocument**			

Operation **IApplication::APIMajorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			
documentation	A change in the **APIMajorVersion** of the type library (e.g. 1.0 => 2.0) means that non-scripting clients (e.g. IDE PlugIns written in C#, VB.NET, C++,...) should be recompiled.					

Operation **IApplication::APIMinorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IApplication::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IApplication::Dialogs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDialogs**			

Operation **IApplication::Edition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IApplication::ImportFromXMIFile**

parameter	name	direction	type	type modifier	multiplicity	default
	strXMIFile	**in**	**string**			
	return	**return**	**IDocument**			

Operation **IApplication::ImportFromXMIFileFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	**IURLDlg**			
	return	**return**	**IDocument**			

Operation **IApplication::InstallationDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IApplication::IsAPISupported**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IApplication::LogMessage**

parameter	name	direction	type	type modifier	multiplicity	default
	strText	**in**	**string**			
	nLogType	**in**	**ENUMMessageLogType**			
	return	**return**	**void**			

Operation **IApplication::LogMessageWithUMLDataLink**

parameter	name	direction	type	type modifier	multiplicity	default
	strText	**in**	**string**			
	ipUMLData	**in**	**IUMLData**			
	nLogType	**in**	**ENUMMessageLogType**			

	return	**return**	**void**

Operation **IApplication::MajorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IApplication::MinorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IApplication::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IApplication::NewDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDocument**			

Operation **IApplication::OpenDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	strModelFilePath	**in**	**string**			
	return	**return**	**IDocument**			

Operation **IApplication::OpenDocumentFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	**IURLDlg**			
	return	**return**	**IDocument**			

Operation **IApplication::Options**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptions**			

Operation **IApplication::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IApplication::PersonalDataDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IApplication::Quit**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IApplication::RunMacro**

parameter	name	direction	type	type modifier	multiplicity	default
	strName	**in**	**string**			
	return	**return**	**void**			

Operation **IApplication::ServicePackVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IApplication::Status**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMApplicationStatus**			

Operation **IApplication::Visible**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

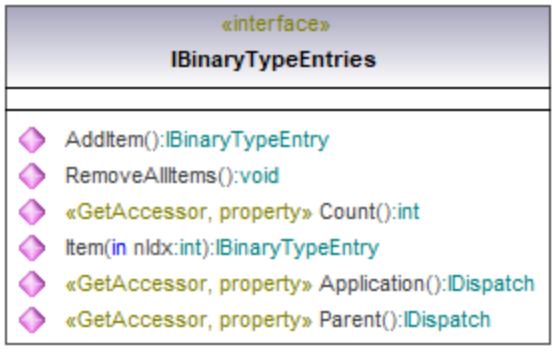
Operation **IApplication::WindowHandle**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.2 UModelAPI - IBinaryTypeEntries

Interface **IBinaryTypeEntries**

diagram	
typedElements	Interface **IImportBinaryTypesDlg** Operation **CSharp_BinaryTypes** **Java_BinaryTypes** **VBasic_BinaryTypes**

Operation **IBinaryTypeEntries::AddItem**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IBinaryTypeEntry**

Operation **IBinaryTypeEntries::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IBinaryTypeEntries::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IBinaryTypeEntries::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IBinaryTypeEntry**			

Operation **IBinaryTypeEntries::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

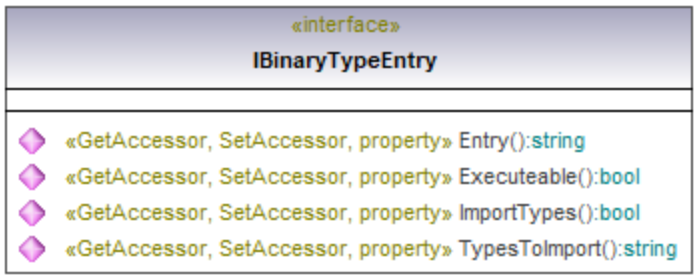
Operation **IBinaryTypeEntries::RemoveAllItems**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.3 UModelAPI - IBinaryTypeEntry

Interface **IBinaryTypeEntry**

diagram	
typedElements	Interface **IBinaryTypeEntries** Operation **AddItem** **Item**

Operation **IBinaryTypeEntry::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IBinaryTypeEntry::Executeable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IBinaryTypeEntry::ImportTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

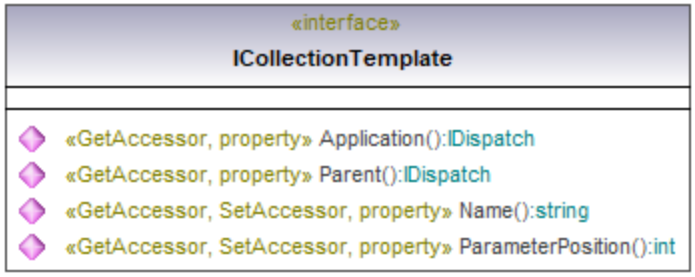
Operation **IBinaryTypeEntry::TypesToImport**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.4 UModelAPI - ICollectionTemplate

Interface **ICollectionTemplate**

diagram	
typedElements	Interface **ICollectionTemplates** Operation **InsertItemAt** **Item**

Operation **ICollectionTemplate::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ICollectionTemplate::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **ICollectionTemplate::ParameterPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

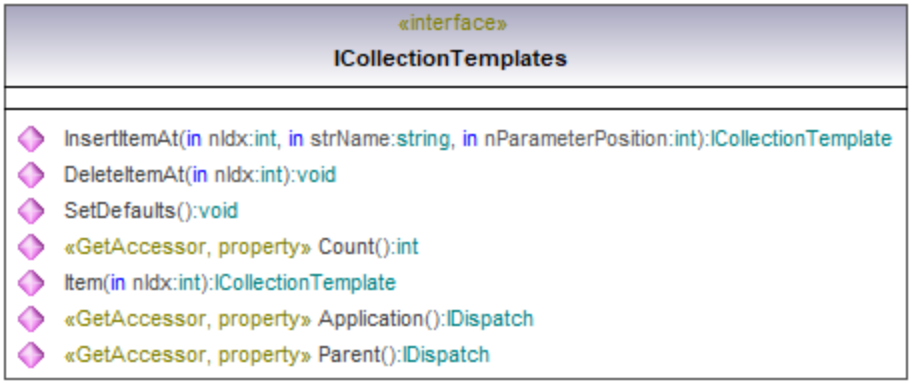
Operation **ICollectionTemplate::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

17.5.2.5 UModelAPI - ICollectionTemplates

Interface **ICollectionTemplates**

diagram



typedElements	Interface	Operation
	ILocalOptionsDiagramEditing	**CollectionTemplates_CSharp** **CollectionTemplates_Java** **CollectionTemplates_UML** **CollectionTemplates_VBasic**

Operation **ICollectionTemplates::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ICollectionTemplates::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ICollectionTemplates::DeleteItemAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **ICollectionTemplates::InsertItemAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strName	**in**	**string**			

	nParameterPosition	**in**	**int**
	return	**return**	**ICollectionTemplate**

Operation **ICollectionTemplates::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**ICollectionTemplate**			

Operation **ICollectionTemplates::Parent**

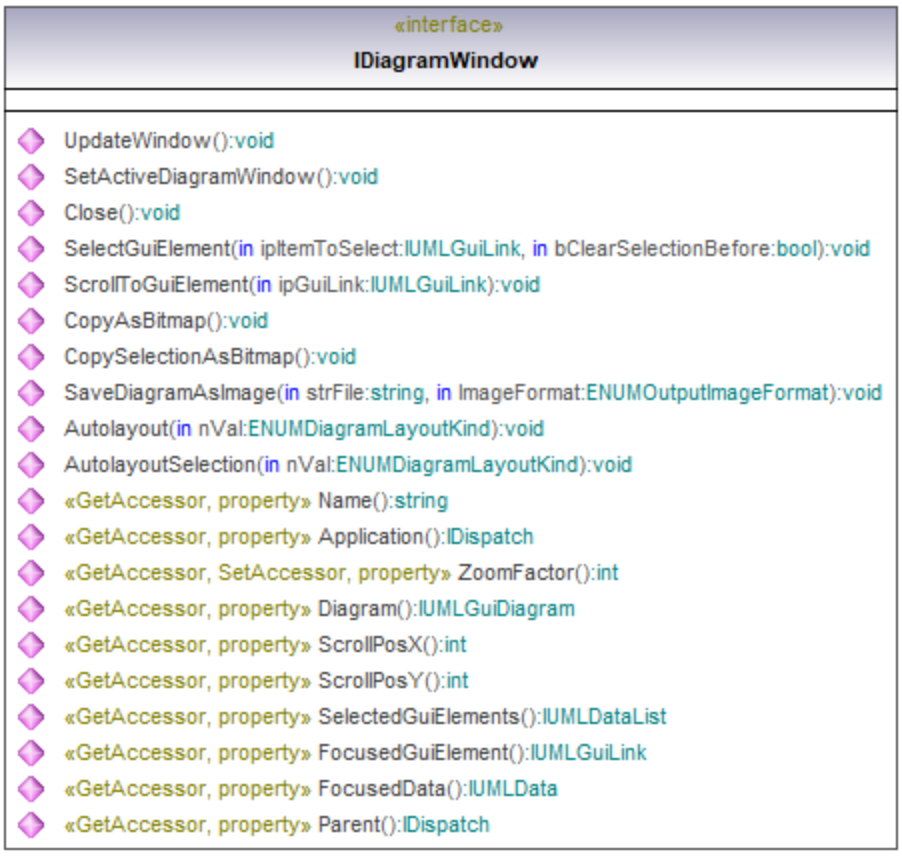
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ICollectionTemplates::SetDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

17.5.2.6 UModelAPI - IDiagramWindow

Interface **IDiagramWindow**

diagram	
typedElements	Interface **_IDiagramWindowEvents** Interface **_IDocumentEvents** Interface **IDiagramWindows** Interface **IDocument** Operation **OnDiagramWindowClosed** Operation **OnActivateDiagramWindow** **OnDiagramWindowClosed** **OnDiagramWindowOpened** Operation **Item** Operation **ActiveDiagramWindow** **OpenDiagram**

Operation **IDiagramWindow::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDiagramWindow::Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
	nVal	**in**	**ENUMDiagramLayoutKind**			
	return	**return**	**void**			

Operation **IDiagramWindow::AutolayoutSelection**

parameter	name	direction	type	type modifier	multiplicity	default
	nVal	**in**	**ENUMDiagramLayoutKind**			
	return	**return**	**void**			

Operation **IDiagramWindow::Close**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDiagramWindow::CopyAsBitmap**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDiagramWindow::CopySelectionAsBitmap**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDiagramWindow::Diagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagram**			

Operation **IDiagramWindow::FocusedData**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData**			

Operation **IDiagramWindow::FocusedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiLink**			

Operation **IDiagramWindow::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IDiagramWindow::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDiagramWindow::SaveDiagramAsImage**

parameter	name	direction	type	type modifier	multiplicity	default
	strFile	**in**	**string**			
	ImageFormat	**in**	**ENUMOutputImageFormat**			
	return	**return**	**void**			

Operation **IDiagramWindow::ScrollPosX**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**int**

Operation **IDiagramWindow::ScrollPosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IDiagramWindow::ScrollToGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	ipGuiLink	**in**	**IUMLGuiLink**			
	return	**return**	**void**			

Operation **IDiagramWindow::SelectedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLGuiElement.					

Operation **IDiagramWindow::SelectGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	ipItemToSelect	**in**	**IUMLGuiLink**			
	bClearSelectionBefore	**in**	**bool**			
	return	**return**	**void**			

Operation **IDiagramWindow::SetActiveDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDiagramWindow::UpdateWindow**

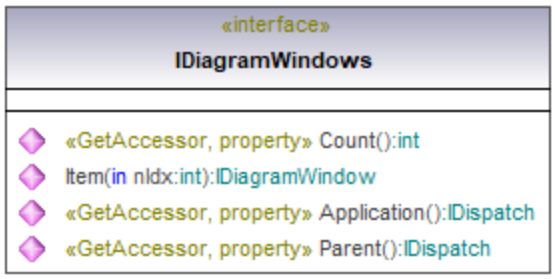
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDiagramWindow::ZoomFactor**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

17.5.2.7 UModelAPI - IDiagramWindows

Interface **IDiagramWindows**

diagram	
typedElements	Interface **IDocument** Operation **DiagramWindows**

Operation **IDiagramWindows::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDiagramWindows::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IDiagramWindows::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IDiagramWindow**			

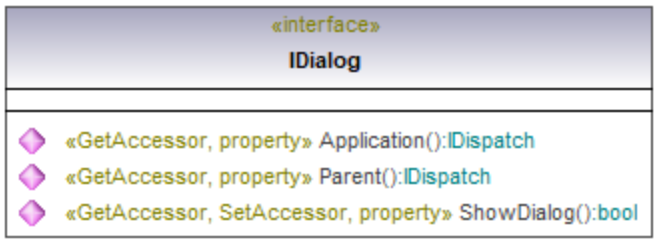
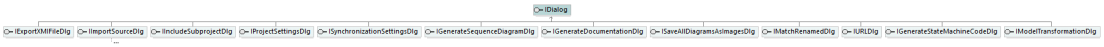
Operation **IDiagramWindows::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.8 UModelAPI - IDialog

Interface **IDialog**

diagram	
hierarchy	

Operation **IDialog::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDialog::Parent**

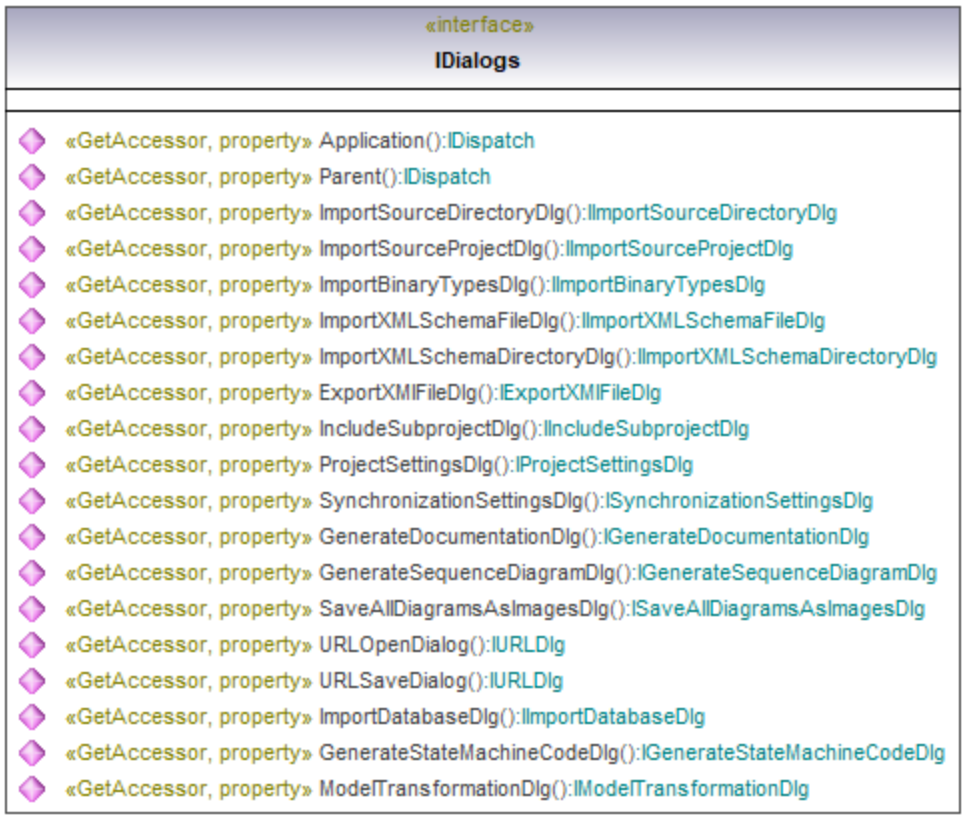
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDialog::ShowDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.2.9 UModelAPI - IDialogs

Interface **IDialogs**

diagram	
typedElements	Interface **IApplication** Operation **Dialogs**

Operation **IDialogs::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDialogs::ExportXMIFileDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IExportXMIFileDlg**			

Operation **IDialogs::GenerateDocumentationDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IGenerateDocumentationDlg**			

Operation **IDialogs::GenerateSequenceDiagramDlg**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	IGenerateSequenceDiagramDlg

Operation **IDialogs::GenerateStateMachineCodeDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IGenerateStateMachineCodeDlg			

Operation **IDialogs::ImportBinaryTypesDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportBinaryTypesDlg			

Operation **IDialogs::ImportDatabaseDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportDatabaseDlg			

Operation **IDialogs::ImportSourceDirectoryDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportSourceDirectoryDlg			

Operation **IDialogs::ImportSourceProjectDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportSourceProjectDlg			

Operation **IDialogs::ImportXMLSchemaDirectoryDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportXMLSchemaDirectoryDlg			

Operation **IDialogs::ImportXMLSchemaFileDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportXMLSchemaFileDlg			

Operation **IDialogs::IncludeSubprojectDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IIncludeSubprojectDlg			

Operation **IDialogs::ModelTransformationDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IModelTransformationDlg			

Operation **IDialogs::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDialogs::ProjectSettingsDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IProjectSettingsDlg**			

Operation **IDialogs::SaveAllDiagramsAsImagesDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ISaveAllDiagramsAsImagesDlg**			

Operation **IDialogs::SynchronizationSettingsDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ISynchronizationSettingsDlg**			

Operation **IDialogs::URLOpenDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IURLDlg**			

Operation **IDialogs::URLSaveDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IURLDlg**			

17.5.2.10 UModelAPI - IDocument

Interface **IDocument**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
typedElements	Interface **_IApplicationEvents** Operation **OnDocumentClosed** **OnDocumentOpened** **OnNewDocument** Interface **_IDocumentEvents** Operation **OnAfterReloadDocument** **OnBeforeReloadDocument** **OnDocumentClosed** **OnDocumentSaved** **OnDocumentSavedAs** **OnModifiedFlagChanged**

	Interface **_ITransactionEvents** Interface **IApplication**	Operation **OnBeginDataModification** **OnEndDataModification** Operation **ActiveDocument** **ImportFromXMIFile** **ImportFromXMIFileFromURL** **NewDocument** **OpenDocument** **OpenDocumentFromURL**

Operation **IDocument::AbortModification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDocument::ActiveDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDiagramWindow**			

Operation **IDocument::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDocument::BeginModification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::CanFocusUMLDataInModelTree**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	return	**return**	**bool**			

Operation **IDocument::CheckProjectSyntax**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::DiagramWindows**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDiagramWindows**			

Operation **IDocument::ElementFamilyStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLGuiStyles**			

Operation **IDocument::EndModification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDocument::ExportToXMIFile**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IExportXMIFileDlg**			
	return	**return**	**bool**			

Operation **IDocument::FocusedUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData**			
documentation	Get the focused UML data of the docoument. Normally this is the one which is shown in the "Properties" window.					

Operation **IDocument::FocusedUMLDataNotifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IFocusedUMLDataNotifier**			

Operation **IDocument::FocusUMLDataInModelTree**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	bFocusModelTree	**in**	**bool**			
	bEnsureModelTreeVisible	**in**	**bool**			
	return	**return**	**void**			

Operation **IDocument::FullName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IDocument::GenerateDocumentation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IGenerateDocumentationDlg**			
	strResultFile	**in**	**string**			
	return	**return**	**bool**			

Operation **IDocument::GenerateSequenceDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IGenerateSequenceDiagramDlg**			
	ipOp	**in**	**IUMLOperation**			
	return	**return**	**bool**			

Operation **IDocument::GenerateSequenceDiagramsForAllOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	bAllPublicOnly	**in**	**bool**			

	bIncludeGetters AndSetters	**in**	**bool**
	return	**return**	**void**

Operation **IDocument::GenerateStateMachineCode**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IGenerateStateMachineCodeDlg**			
	ipStateMachine	**in**	**IUMLStateMachine**			
	return	**return**	**bool**			

Operation **IDocument::GuiRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiRootElement**			

Operation **IDocument::ImportBinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IImportBinaryTypesDlg**			
	return	**return**	**bool**			

Operation **IDocument::ImportDatabase**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IImportDatabaseDlg**			
	return	**return**	**bool**			

Operation **IDocument::ImportSourceDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IImportSourceDirectoryDlg**			
	return	**return**	**bool**			

Operation **IDocument::ImportSourceProject**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IImportSourceProjectDlg**			
	return	**return**	**bool**			

Operation **IDocument::ImportXMLSchemaDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IImportXMLSchemaDirectoryDlg**			
	return	**return**	**bool**			

Operation **IDocument::ImportXMLSchemaFile**

parameter	name	direction	type	type modifier	multiplicity	default

	ipDlg	**in**	**IImportXMLSchemaFileDlg**
	return	**return**	**bool**

Operation **IDocument::IncludeSubproject**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IIncludeSubprojectDlg**			
	return	**return**	**bool**			

Operation **IDocument::IsInUndoRedo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::IsLoadedFromPreviousFileFormat**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	True, when the document has been loaded from a project file with a previous file format version. When saving the document, previous versions of UModel will not be able to load this file anymore.					

Operation **IDocument::LineStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

Operation **IDocument::MergeProject**

parameter	name	direction	type	type modifier	multiplicity	default
	strProjectFile	**in**	**string**			
	return	**return**	**bool**			

Operation **IDocument::MergeProject3Way**

parameter	name	direction	type	type modifier	multiplicity	default
	strProjectFile	**in**	**string**			
	strCommonAncestorFile	**in**	**string**			
	return	**return**	**bool**			

Operation **IDocument::MergeProjectFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	**IURLDlg**			
	return	**return**	**bool**			

Operation **IDocument::ModelTransformation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IModelTransformationDlg**			
	return	**return**	**bool**			

Operation **IDocument::ModifiedFlag**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IDocument::NodeStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

Operation **IDocument::OpenAllDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDocument::OpenDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLDiagram	**in**	**IUMLGuiDiagram**			
	return	**return**	**IDiagramWindow**			

Operation **IDocument::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDocument::Path**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IDocument::ProjectSettings**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IProjectSettingsDlg**			
	return	**return**	**void**			

Operation **IDocument::ProjectStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

Operation **IDocument::Reload**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::RootPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IDocument::Save**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDocument::SaveAllDiagramsAsImages**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISaveAllDiagramsAsImagesDlg**			
	return	**return**	**bool**			

Operation **IDocument::SaveAs**

parameter	name	direction	type	type modifier	multiplicity	default
	strFileName	**in**	**string**			
	return	**return**	**void**			

Operation **IDocument::SaveCopyAs**

parameter	name	direction	type	type modifier	multiplicity	default
	strFileName	**in**	**string**			
	return	**return**	**void**			

Operation **IDocument::Saved**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::SaveToURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	**IURLDlg**			
	return	**return**	**void**			

Operation **IDocument::SynchronizationSettings**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISynchronizationSettingsDlg**			
	return	**return**	**void**			

Operation **IDocument::SynchronizeCodeFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISynchronizationSettingsDlg**			
	return	**return**	**bool**			

Operation **IDocument::SynchronizeModelFromCode**

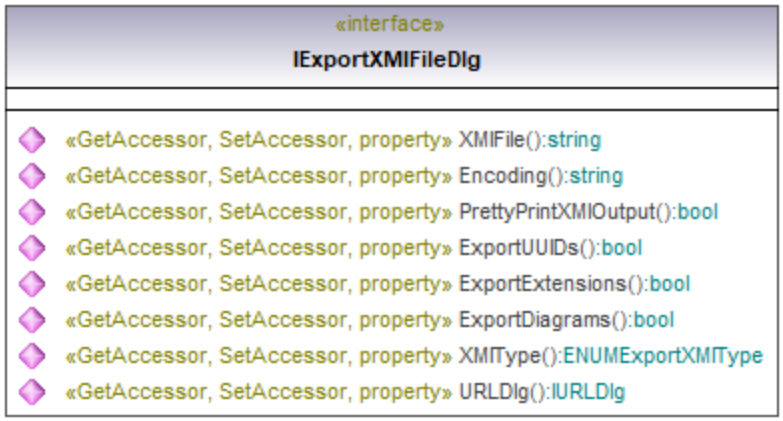
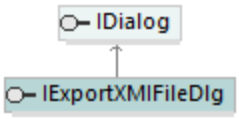
parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISynchronizationSettingsDlg**			
	return	**return**	**bool**			

Operation **IDocument::TransactionNotifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ITransactionNotifier**			

17.5.2.11 UModelAPI - IExportXMIFileDlg

Interface **IExportXMIFileDlg**

diagram	
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **ExportXMIFileDlg** Operation **ExportToXMIFile**

Operation **IExportXMIFileDlg::Encoding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IExportXMIFileDlg::ExportDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IExportXMIFileDlg::ExportExtensions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IExportXMIFileDlg::ExportUUIDs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IExportXMIFileDlg::PrettyPrintXMIOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IExportXMIFileDlg::URLDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IURLDlg**			

Operation **IExportXMIFileDlg::XMIFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

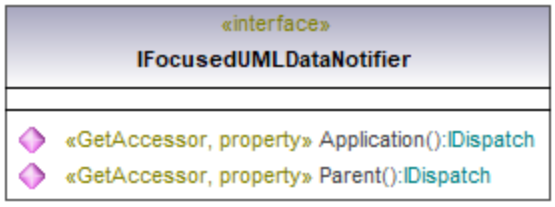
Operation **IExportXMIFileDlg::XMIType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMExportXMIType**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.12 UModelAPI - IFocusedUMLDataNotifier

Interface **IFocusedUMLDataNotifier**

diagram	
typedElements	Interface **IDocument** Operation **FocusedUMLDataNotifier**

Operation **IFocusedUMLDataNotifier::Application**

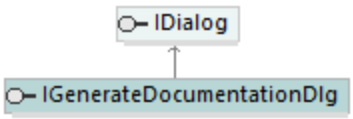
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IFocusedUMLDataNotifier::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

17.5.2.13 UModelAPI - IGenerateDocumentationDlg

Interface **IGenerateDocumentationDlg**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **GenerateDocumentationDlg** Operation **GenerateDocumentation**

Operation **IGenerateDocumentationDlg::CreateFolderForDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Associations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_BoundElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Constraints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Diagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Documentation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_DocumentationAsHTML**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Generals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagramExpandItemsOnlyOnce**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagramNestingDepthDown**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagramNestingDepthUp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Details_Hyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_ImplementedInterfaces**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_OperationParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_OwnedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Owner**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**bool**

Operation **IGenerateDocumentationDlg::Details_Properties**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_SelectAll**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Details_SelectNone**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Details_ShownOnDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_SourceTargetOfRelations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Specifics**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_TemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_TemplateParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_TypedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::DiagramImageFormat**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMOutputImageFormat**			

Operation **IGenerateDocumentationDlg::EmbedCSSinHTML**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::EmbedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_GetFace**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**string**			

Operation **IGenerateDocumentationDlg::Fonts_GetSize**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Fonts_GetTextColor**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Fonts_IsBold**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_IsItalic**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_IsUnderline**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_SetBold**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetFace**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetItalic**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetSize**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetTextColor**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetUnderline**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::GenerateLinksToLocalFiles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMDocumentationFilePathKind**			

Operation **IGenerateDocumentationDlg::Include_Diagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IKindSelectionList**			

Operation **IGenerateDocumentationDlg::Include_Elements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IKindSelectionList**			

Operation **IGenerateDocumentationDlg::Include_IncludedPredefinedSubprojects**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Include_IncludedSubprojects**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Include_Index**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Include_NamedElementsOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Include_SelectAllDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectAllElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectAllKindsOf**

parameter	name	direction	type	type modifier	multiplicity	default
	strKindName	**in**	**string**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectKind**

parameter	name	direction	type	type modifier	multiplicity	default
	strKindName	**in**	**string**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectNoDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**void**

Operation **IGenerateDocumentationDlg::Include_SelectNoElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_UnknownExternals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::OutputFormat**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMDocumentationOutputFormat**			

Operation **IGenerateDocumentationDlg::ShowResultFileAfterGeneration**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::SplitOutputToMultipleFiles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::SPSFile**

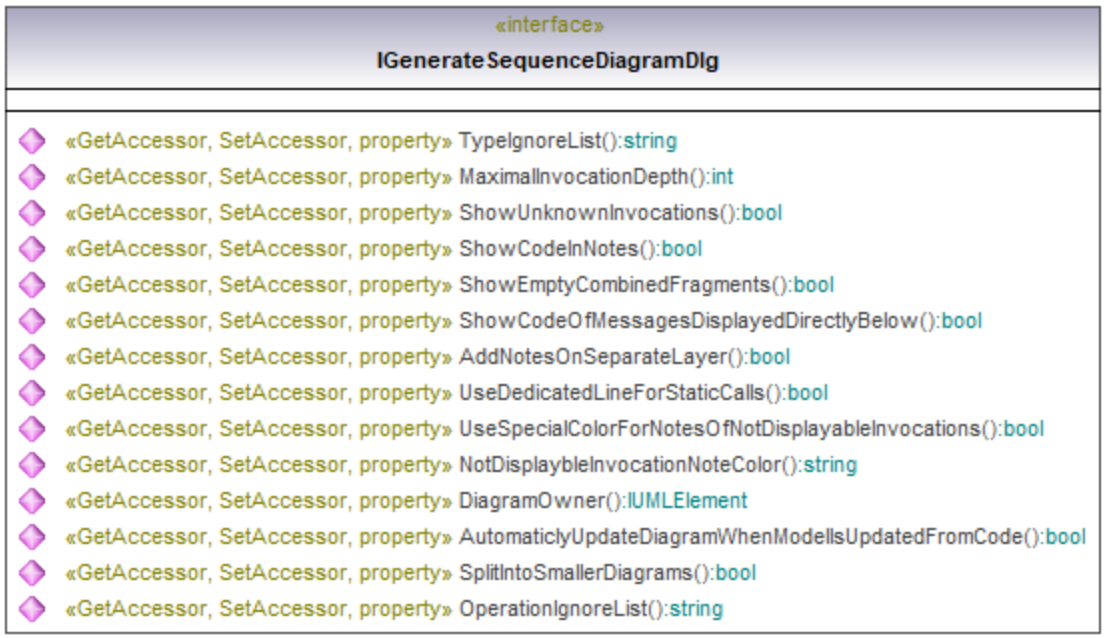
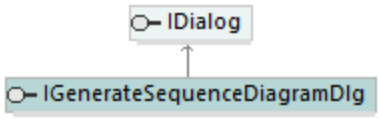
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateDocumentationDlg::UseFixedDesign**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.2.14 UModelAPI - IGenerateSequenceDiagramDlg

Interface **IGenerateSequenceDiagramDlg**

diagram	
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **GenerateSequenceDiagramDlg** Operation **GenerateSequenceDiagram**

Operation **IGenerateSequenceDiagramDlg::AddNotesOnSeparateLayer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::AutomaticlyUpdateDiagramWhenModelIsUpdatedFromCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::DiagramOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IGenerateSequenceDiagramDlg::MaximalInvocationDepth**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**int**

Operation **IGenerateSequenceDiagramDlg::NotDisplayableInvocationNoteColor**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateSequenceDiagramDlg::OperationIgnoreList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateSequenceDiagramDlg::ShowCodeInNotes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::ShowCodeOfMessagesDisplayedDirectlyBelow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::ShowEmptyCombinedFragments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::ShowUnknownInvocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::SplitIntoSmallerDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::TypeIgnoreList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateSequenceDiagramDlg::UseDedicatedLineForStaticCalls**

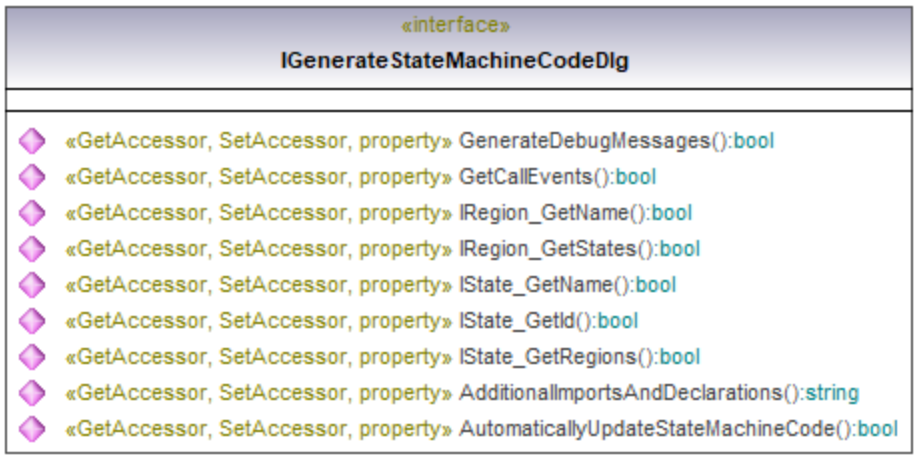
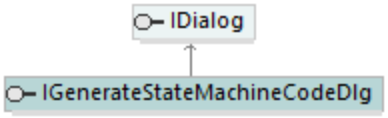
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::UseSpecialColorForNotesOfNotDisplayableInvocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.2.15 UModelAPI - IGenerateStateMachineCodeDlg

Interface **IGenerateStateMachineCodeDlg**

diagram	
hierarchy	
typedElements	Interface IDialogs Interface IDocument Operation GenerateStateMachineCodeDlg Operation GenerateStateMachineCode

Operation **IGenerateStateMachineCodeDlg::AdditionalImportsAndDeclarations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateStateMachineCodeDlg::AutomaticallyUpdateStateMachineCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::GenerateDebugMessages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::GetCallEvents**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::IRegion_GetName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::IRegion_GetStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::IState_GetId**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::IState_GetName**

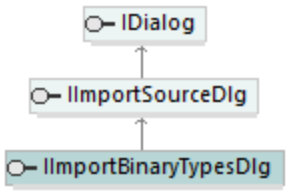
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::IState_GetRegions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.2.16 UModelAPI - IImportBinaryTypesDlg

Interface **IImportBinaryTypesDlg**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **ImportBinaryTypesDlg** Operation **ImportBinaryTypes**

Operation **IImportBinaryTypesDlg::Content_GenerateDiagramPerPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Content_GenerateSingleDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Content_HyperlinkPackagesToDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Content_ShowAnonymousBoundElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Content_ShowNestedClassifiersSeparately**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_BinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IBinaryTypeEntries**			

Operation **IImportBinaryTypesDlg::CSharp_ImportReferencedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_ImportReferencedTypesRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::CSharp_ImportTypesOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_ImportVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_ImportVisibilityRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::CSharp_OneAttributePerAttributeSection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_OverridePathForNativeLibraries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::CSharp_ReflectionOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_SuppressAttributeSections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_SuppressAttributeSuffix**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_BinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IBinaryTypeEntries**			

Operation **IImportBinaryTypesDlg::Java_ImportReferencedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_ImportReferencedTypesRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::Java_ImportTypesOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_ImportVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_ImportVisibilityRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::Java_OverridePathForNativeLibraries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::Java_SuppressAnnotationModifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Runtime**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::VBasic_BinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IBinaryTypeEntries**			

Operation **IImportBinaryTypesDlg::VBasic_ImportReferencedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_ImportReferencedTypesRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::VBasic_ImportTypesOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_ImportVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_ImportVisibilityRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::VBasic_OneAttributePerAttributeSection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_OverridePathForNativeLibraries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::VBasic_ReflectionOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_SuppressAttributeSections**

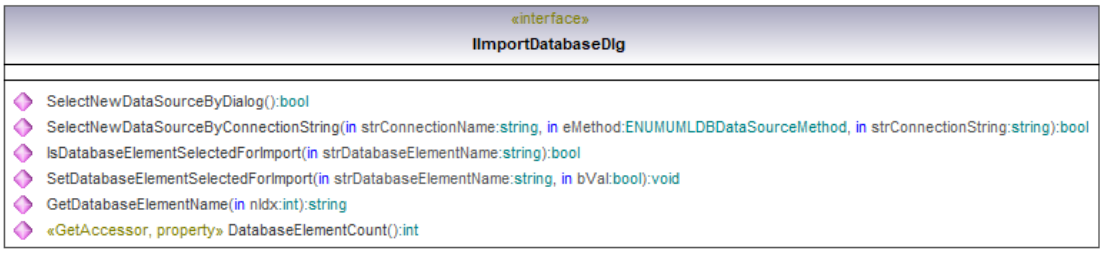
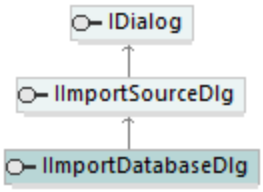
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_SuppressAttributeSuffix**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.2.17 UModelAPI - IImportDatabaseDlg

Interface **IImportDatabaseDlg**

diagram	
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **ImportDatabaseDlg** Operation **ImportDatabase**

Operation **IImportDatabaseDlg::DatabaseElementCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IImportDatabaseDlg::GetDatabaseElementName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IImportDatabaseDlg::IsDatabaseElementSelectedForImport**

parameter	name	direction	type	type modifier	multiplicity	default
	strDatabaseElementName	**in**	**string**			
	return	**return**	**bool**			

Operation **IImportDatabaseDlg::SelectNewDataSourceByConnectionString**

parameter	name	direction	type	type modifier	multiplicity	default

	strConnectionName	**in**	**string**
	eMethod	**in**	**ENUMUMLDBDataSourceMethod**
	strConnectionString	**in**	**string**
	return	**return**	**bool**

Operation **IImportDatabaseDlg::SelectNewDataSourceByDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

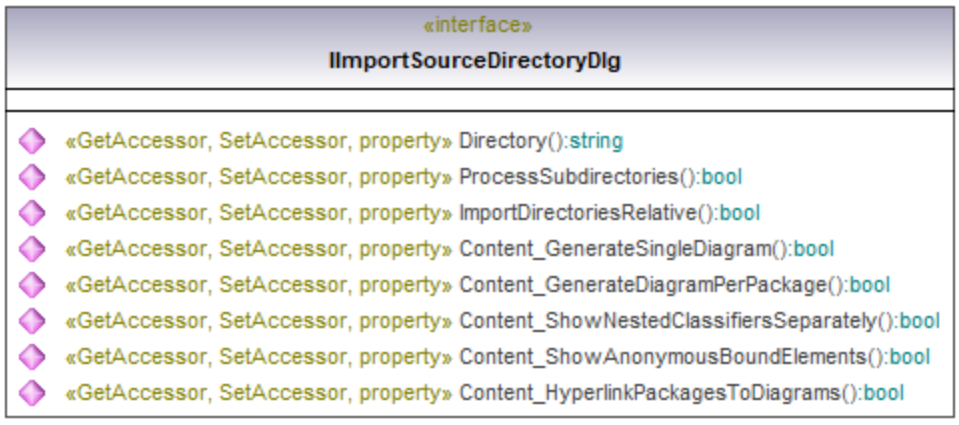
Operation **IImportDatabaseDlg::SetDatabaseElementSelectedForImport**

parameter	name	direction	type	type modifier	multiplicity	default
	strDatabaseElementName	**in**	**string**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

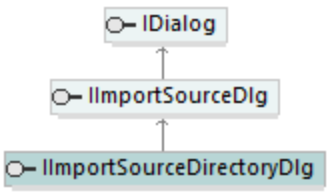
17.5.2.18 UModelAPI - IImportSourceDirectoryDlg

Interface **IImportSourceDirectoryDlg**

diagram



hierarchy



typedElements	Interface **IDialogs** Interface **IDocument**	Operation **ImportSourceDirectoryDlg** Operation **ImportSourceDirectory**

Operation **IImportSourceDirectoryDlg::Content_GenerateDiagramPerPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Content_GenerateSingleDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Content_HyperlinkPackagesToDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Content_ShowAnonymousBoundElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Content_ShowNestedClassifiersSeparately**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Directory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportSourceDirectoryDlg::ImportDirectoriesRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

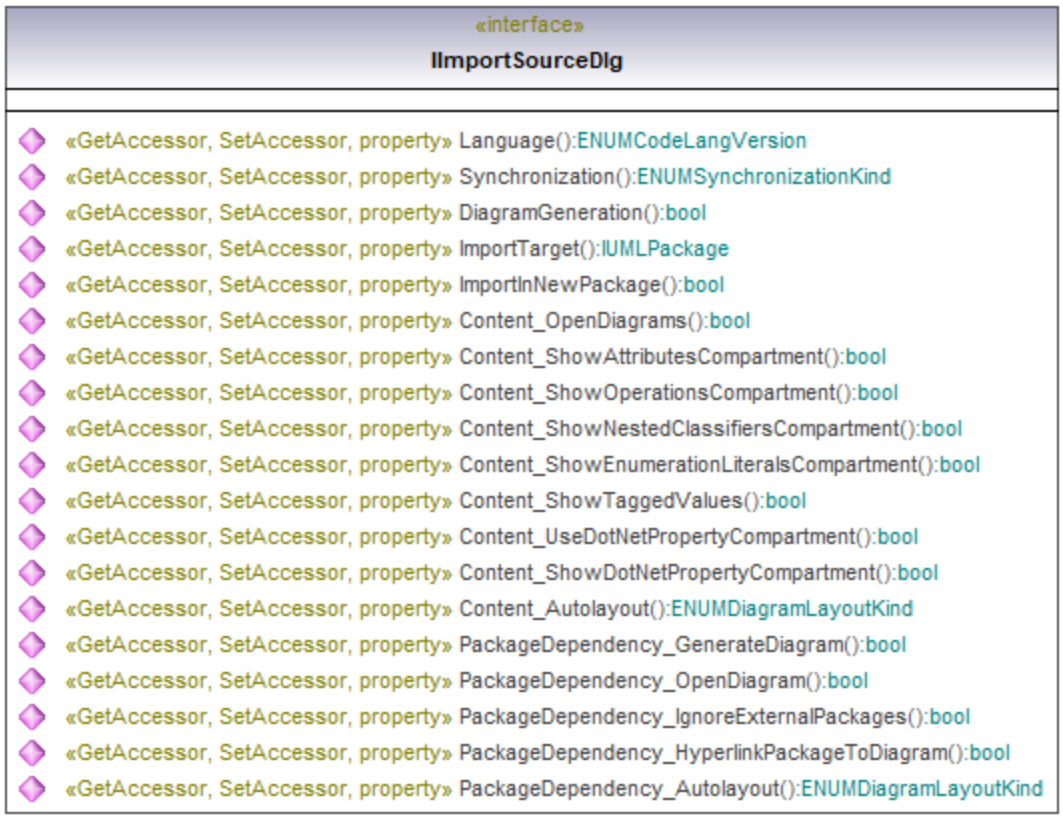
Operation **IImportSourceDirectoryDlg::ProcessSubdirectories**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

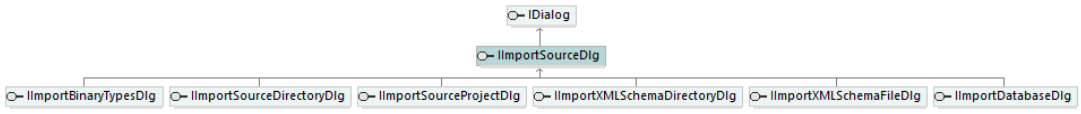
17.5.2.19 UModelAPI - IImportSourceDlg

Interface **IImportSourceDlg**

diagram



hierarchy



Operation **IImportSourceDlg::Content_Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMDiagramLayoutKind			

Operation **IImportSourceDlg::Content_OpenDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowAttributesCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowEnumerationLiteralsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowNestedClassifiersCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowOperationsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowTaggedValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_UseDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::DiagramGeneration**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::ImportInNewPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::ImportTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IImportSourceDlg::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion**			

Operation **IImportSourceDlg::PackageDependency_Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMDiagramLayoutKind**			

Operation **IImportSourceDlg::PackageDependency_GenerateDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::PackageDependency_HyperlinkPackageToDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::PackageDependency_IgnoreExternalPackages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::PackageDependency_OpenDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Synchronization**

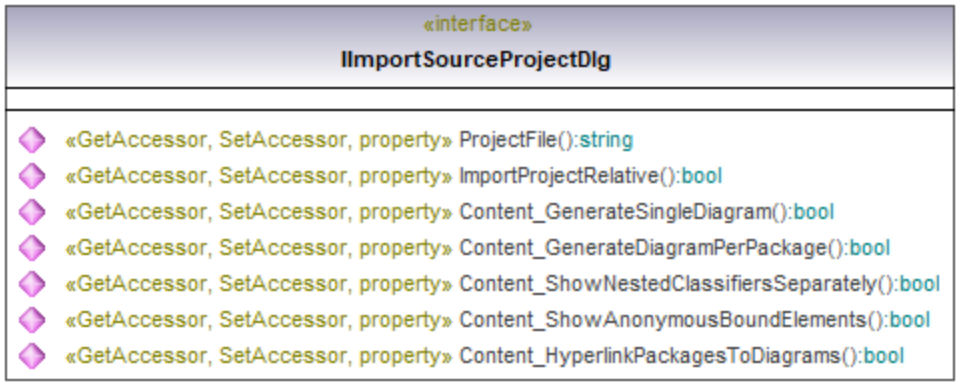
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSynchronizationKind**			

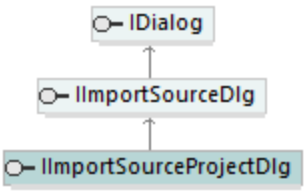
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.20 UModelAPI - IImportSourceProjectDlg

Interface **IImportSourceProjectDlg**

diagram



hierarchy	
typed Elements	Interface **IDialogs** Interface **IDocument** Operation **ImportSourceProjectDlg** Operation **ImportSourceProject**

Operation **IImportSourceProjectDlg::Content_GenerateDiagramPerPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::Content_GenerateSingleDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::Content_HyperlinkPackagesToDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::Content_ShowAnonymousBoundElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::Content_ShowNestedClassifiersSeparately**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::ImportProjectRelative**

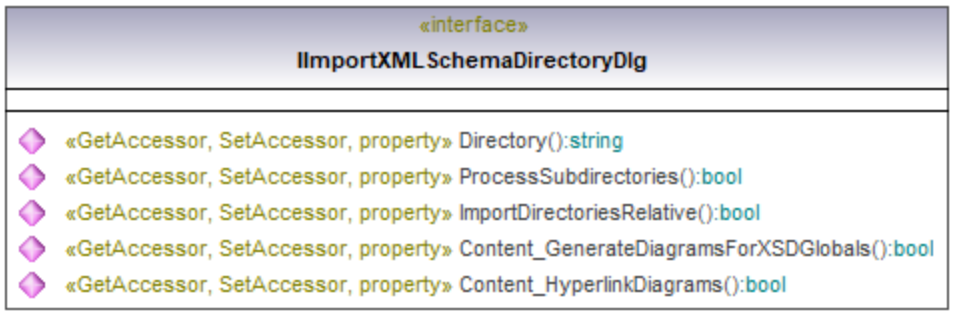
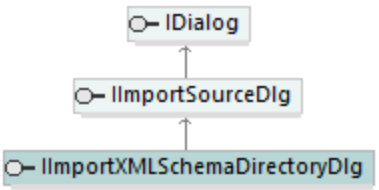
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::ProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.2.21 UModelAPI - IImportXMLSchemaDirectoryDlg

Interface **IImportXMLSchemaDirectoryDlg**

diagram	
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **ImportXMLSchemaDirectoryDlg** Operation **ImportXMLSchemaDirectory**

Operation **IImportXMLSchemaDirectoryDlg::Content_GenerateDiagramsForXSDGlobals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaDirectoryDlg::Content_HyperlinkDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaDirectoryDlg::Directory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportXMLSchemaDirectoryDlg::ImportDirectoriesRelative**

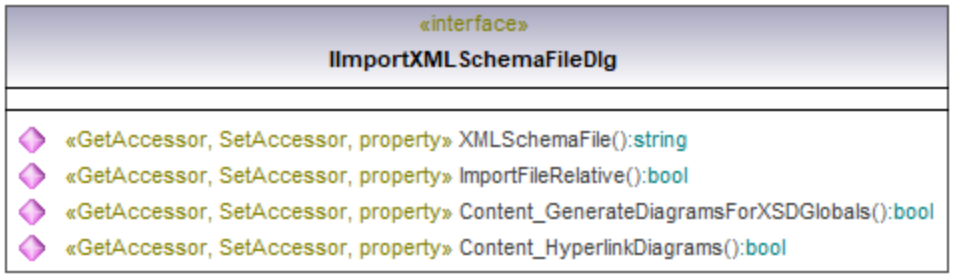
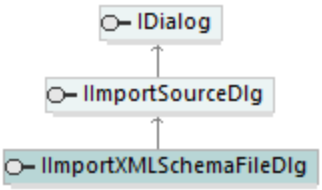
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaDirectoryDlg::ProcessSubdirectories**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.2.22 UModelAPI - IImportXMLSchemaFileDlg

Interface **IImportXMLSchemaFileDlg**

diagram	
hierarchy	
typedElements	Interface IDialogs Interface IDocument Operation ImportXMLSchemaFileDlg Operation ImportXMLSchemaFile

Operation **IImportXMLSchemaFileDlg::Content_GenerateDiagramsForXSDGlobals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaFileDlg::Content_HyperlinkDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaFileDlg::ImportFileRelative**

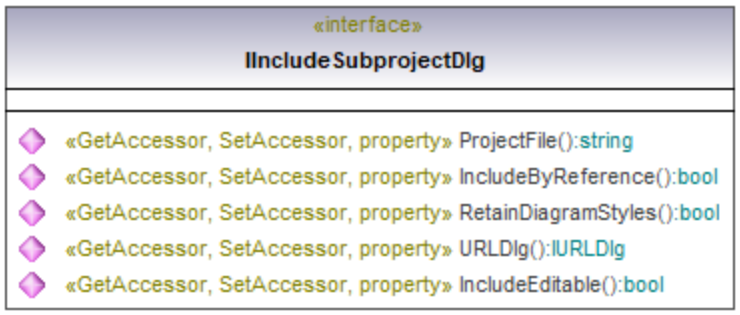
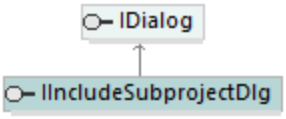
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaFileDlg::XMLSchemaFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.2.23 UModelAPI - IIncludeSubprojectDlg

Interface **IIncludeSubprojectDlg**

diagram	
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **IncludeSubprojectDlg** Operation **IncludeSubproject**

Operation **IIncludeSubprojectDlg::IncludeByReference**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IIncludeSubprojectDlg::IncludeEditable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IIncludeSubprojectDlg::ProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IIncludeSubprojectDlg::RetainDiagramStyles**

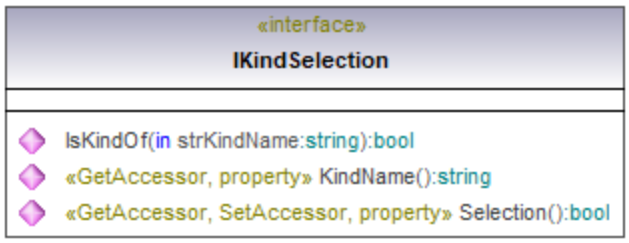
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IIncludeSubprojectDlg::URLDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IURLDlg**			

17.5.2.24 UModelAPI - IKindSelection

Interface **IKindSelection**

diagram	
typedElements	Interface **IKindSelectionList** Operation **Item**

Operation **IKindSelection::IsKindOf**

parameter	name	direction	type	type modifier	multiplicity	default
	strKindName	**in**	**string**			
	return	**return**	**bool**			

Operation **IKindSelection::KindName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IKindSelection::Selection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.25 UModelAPI - IKindSelectionList

Interface **IKindSelectionList**

diagram	
typedElements	Interface **IGenerateDocumentationDlg** Operation **Include_Diagrams** **Include_Elements**

Operation **IKindSelectionList::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IKindSelectionList::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IKindSelectionList::Item**

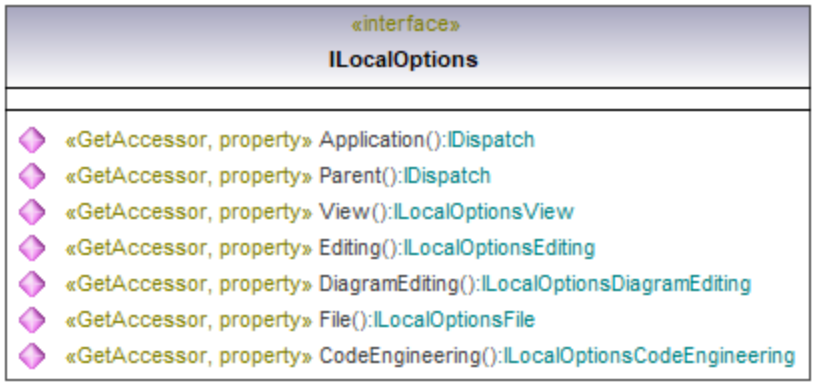
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IKindSelection**			

Operation **IKindSelectionList::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

17.5.2.26 UModelAPI - ILocalOptions

Interface **ILocalOptions**

diagram	
typedElements	Interface **IApplication** Operation **Options**

Operation **ILocalOptions::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptions::CodeEngineering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsCodeEngineering**			

Operation **ILocalOptions::DiagramEditing**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsDiagramEditing**			

Operation **ILocalOptions::Editing**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsEditing**			

Operation **ILocalOptions::File**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsFile**			

Operation **ILocalOptions::Parent**

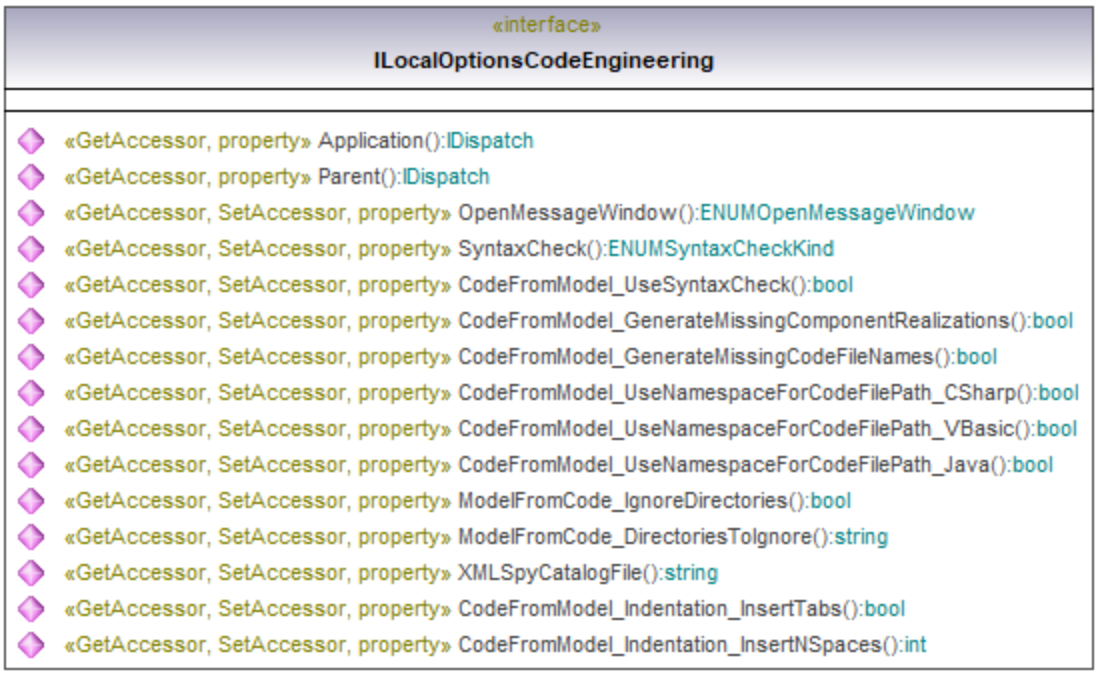
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptions::View**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsView**			

17.5.2.27 UModelAPI - ILocalOptionsCodeEngineering

Interface **ILocalOptionsCodeEngineering**

diagram	
typedElements	Interface **ILocalOptions** Operation **CodeEngineering**

Operation **ILocalOptionsCodeEngineering::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_GenerateMissingCodeFileNames**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_GenerateMissingComponentRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_Indentation_InsertNSpaces**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_Indentation_InsertTabs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_UseNamespaceForCodeFilePath_CSharp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_UseNamespaceForCodeFilePath_Java**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_UseNamespaceForCodeFilePath_VBasic**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_UseSyntaxCheck**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::ModelFromCode_DirectoriesToIgnore**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **ILocalOptionsCodeEngineering::ModelFromCode_IgnoreDirectories**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::OpenMessageWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMOpenMessageWindow**			

Operation **ILocalOptionsCodeEngineering::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsCodeEngineering::SyntaxCheck**

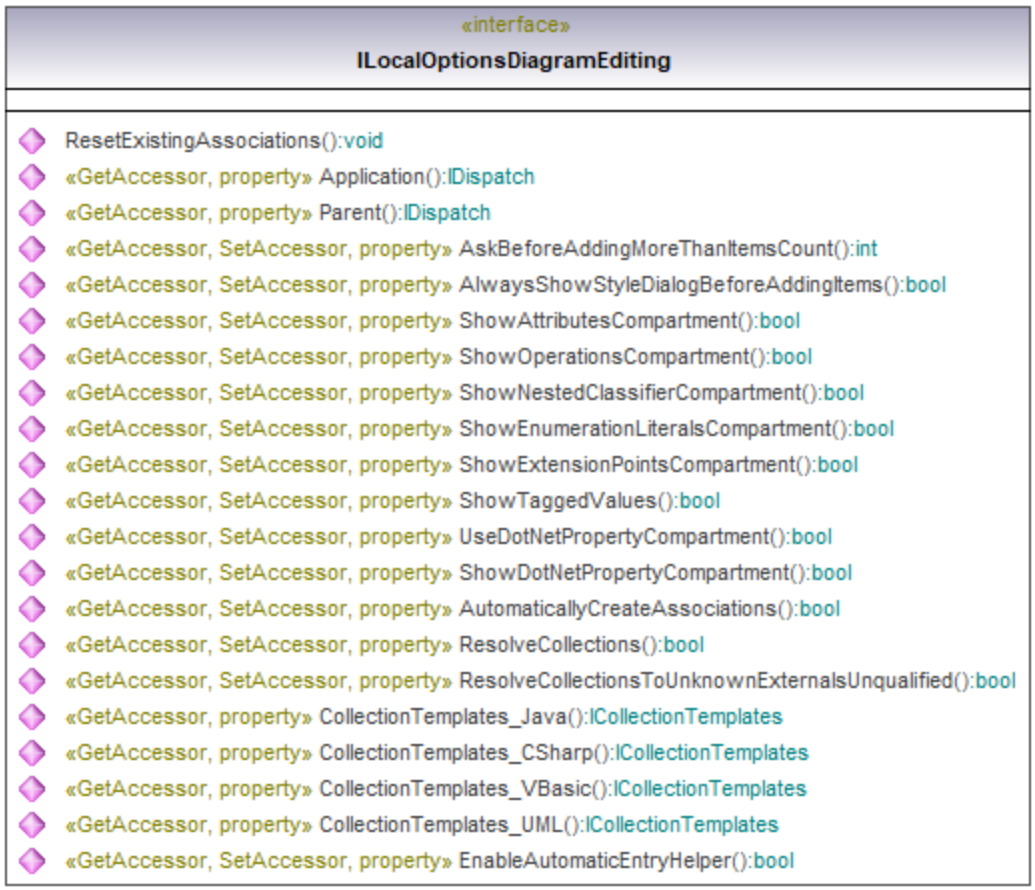
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSyntaxCheckKind**			

Operation **ILocalOptionsCodeEngineering::XMLSpyCatalogFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.2.28 UModelAPI - ILocalOptionsDiagramEditing

Interface **ILocalOptionsDiagramEditing**

diagram	
typedElements	Interface **ILocalOptions** Operation **DiagramEditing**

Operation **ILocalOptionsDiagramEditing::AlwaysShowStyleDialogBeforeAddingItems**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsDiagramEditing::AskBeforeAddingMoreThanItemsCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsDiagramEditing::AutomaticallyCreateAssociations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_CSharp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates**			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_Java**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates**			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_UML**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates**			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_VBasic**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates**			

Operation **ILocalOptionsDiagramEditing::EnableAutomaticEntryHelper**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsDiagramEditing::ResetExistingAssociations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **ILocalOptionsDiagramEditing::ResolveCollections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ResolveCollectionsToUnknownExternalsUnqualified**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowAttributesCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowEnumerationLiteralsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowExtensionPointsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowNestedClassifierCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowOperationsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowTaggedValues**

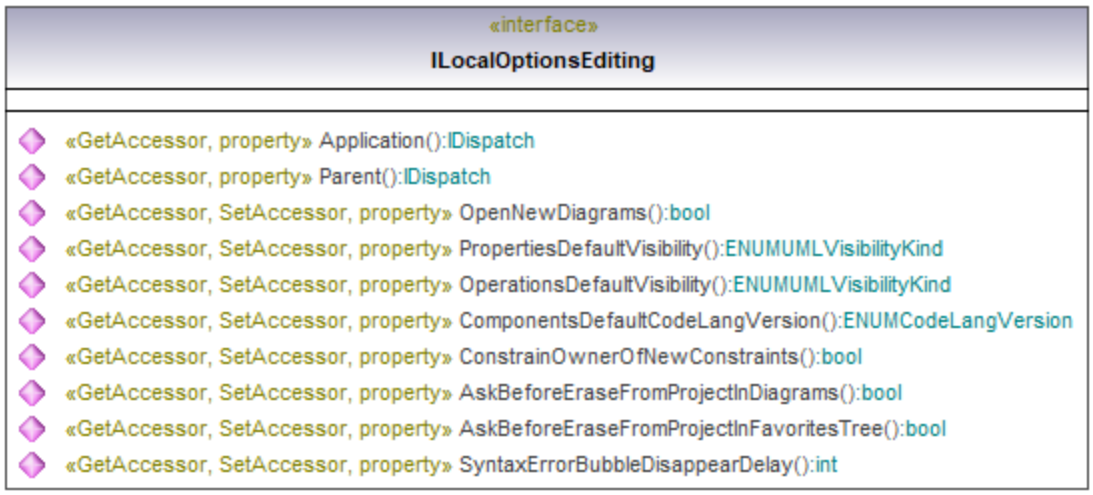
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::UseDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.2.29 UModelAPI - ILocalOptionsEditing

Interface **ILocalOptionsEditing**

diagram	
typedElements	Interface **ILocalOptions** Operation **Editing**

Operation **ILocalOptionsEditing::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsEditing::AskBeforeEraseFromProjectInDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsEditing::AskBeforeEraseFromProjectInFavoritesTree**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsEditing::ComponentsDefaultCodeLangVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion**			

Operation **ILocalOptionsEditing::ConstrainOwnerOfNewConstraints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsEditing::OpenNewDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsEditing::OperationsDefaultVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

Operation **ILocalOptionsEditing::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsEditing::PropertiesDefaultVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

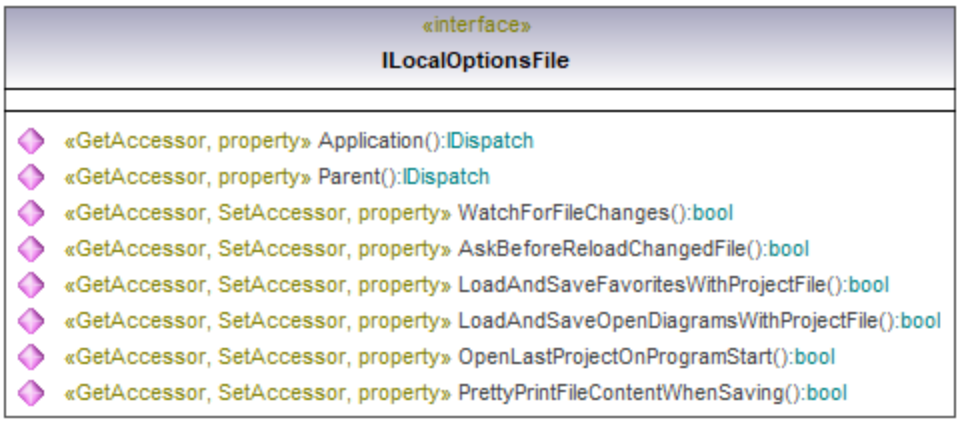
Operation **ILocalOptionsEditing::SyntaxErrorBubbleDisappearDelay**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

17.5.2.30 UModelAPI - ILocalOptionsFile

Interface **ILocalOptionsFile**

diagram



typedElements	Interface **ILocalOptions**	Operation **File**

Operation **ILocalOptionsFile::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsFile::AskBeforeReloadChangedFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsFile::LoadAndSaveFavoritesWithProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsFile::LoadAndSaveOpenDiagramsWithProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsFile::OpenLastProjectOnProgramStart**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsFile::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsFile::PrettyPrintFileContentWhenSaving**

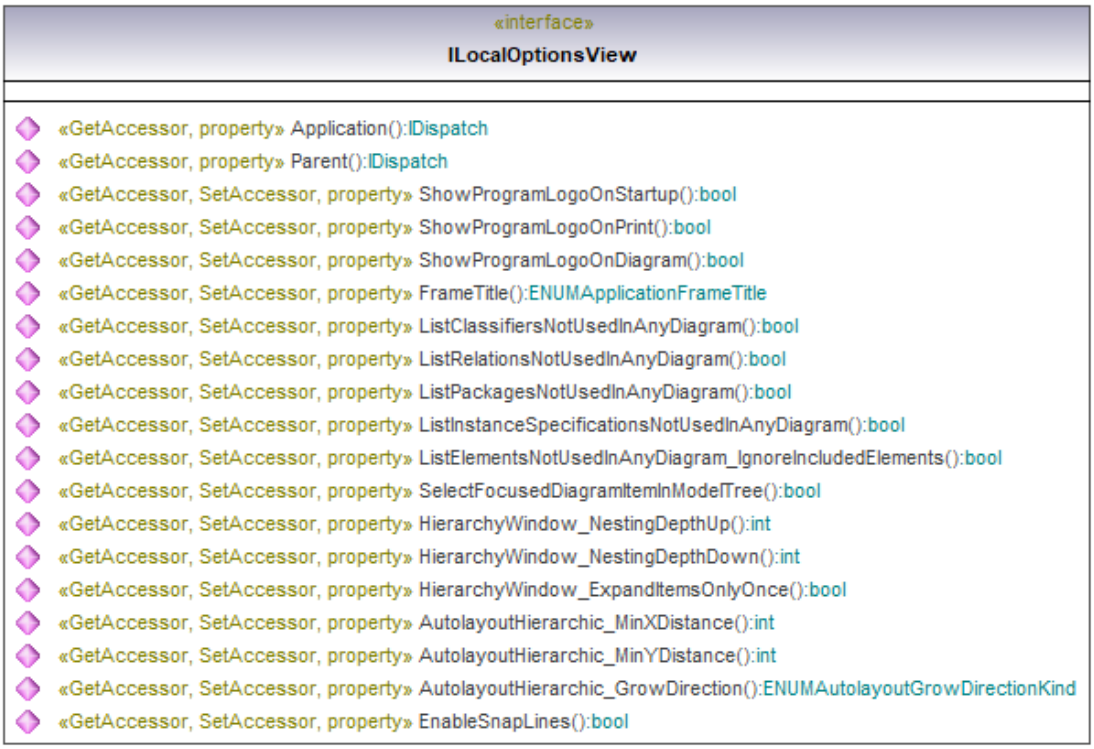
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsFile::WatchForFileChanges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.2.31 UModelAPI - ILocalOptionsView

Interface **ILocalOptionsView**

diagram	
typedElements	Interface **ILocalOptions** Operation **View**

Operation **ILocalOptionsView::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsView::AutolayoutHierarchic_GrowDirection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMAutolayoutGrowDirectionKind**			

Operation **ILocalOptionsView::AutolayoutHierarchic_MinXDistance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsView::AutolayoutHierarchic_MinYDistance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsView::EnableSnapLines**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::FrameTitle**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMApplication FrameTitle**			

Operation **ILocalOptionsView::HierarchyWindow_ExpandItemsOnlyOnce**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::HierarchyWindow_NestingDepthDown**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsView::HierarchyWindow_NestingDepthUp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsView::ListClassifiersNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ListElementsNotUsedInAnyDiagram_IgnoreIncludedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ListInstanceSpecificationsNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ListPackagesNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ListRelationsNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsView::SelectFocusedDiagramItemInModelTree**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ShowProgramLogoOnDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ShowProgramLogoOnPrint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

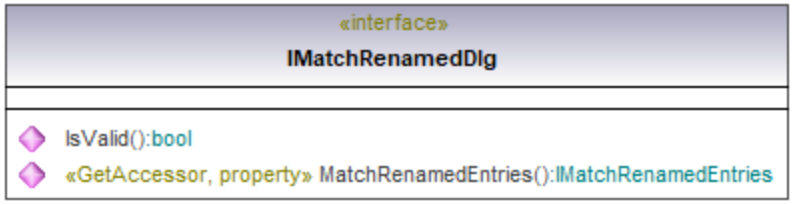
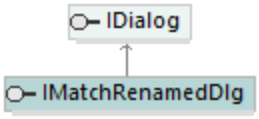
Operation **ILocalOptionsView::ShowProgramLogoOnStartup**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.32 UModelAPI - IMatchRenamedDlg

Interface **IMatchRenamedDlg**

diagram	
hierarchy	
typedElements	Interface **_ISynchronizationEvents** Operation **OnMatchRenamed**

Operation **IMatchRenamedDlg::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

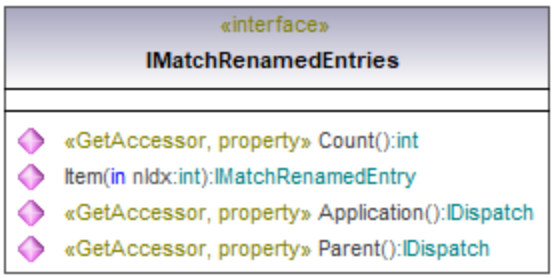
Operation **IMatchRenamedDlg::MatchRenamedEntries**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IMatchRenamedEntries**

17.5.2.33 UModelAPI - IMatchRenamedEntries

Interface **IMatchRenamedEntries**

diagram	
typedElements	Interface **IMatchRenamedDlg** Operation **MatchRenamedEntries**

Operation **IMatchRenamedEntries::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IMatchRenamedEntries::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IMatchRenamedEntries::Item**

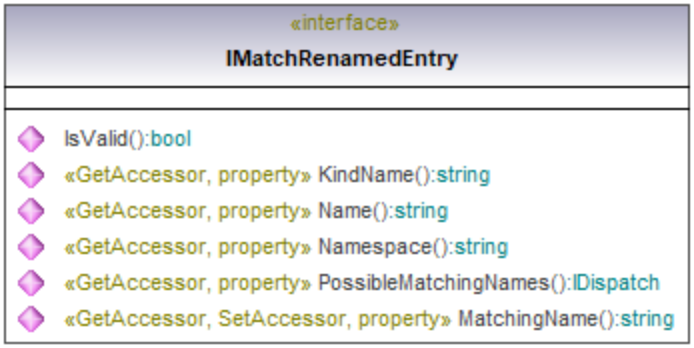
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IMatchRenamedEntry**			

Operation **IMatchRenamedEntries::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

17.5.2.34 UModelAPI - IMatchRenamedEntry

Interface **IMatchRenamedEntry**

diagram	
typedElements	Interface **IMatchRenamedEntries** Operation **Item**

Operation **IMatchRenamedEntry::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IMatchRenamedEntry::KindName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IMatchRenamedEntry::MatchingName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IMatchRenamedEntry::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IMatchRenamedEntry::Namespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IMatchRenamedEntry::PossibleMatchingNames**

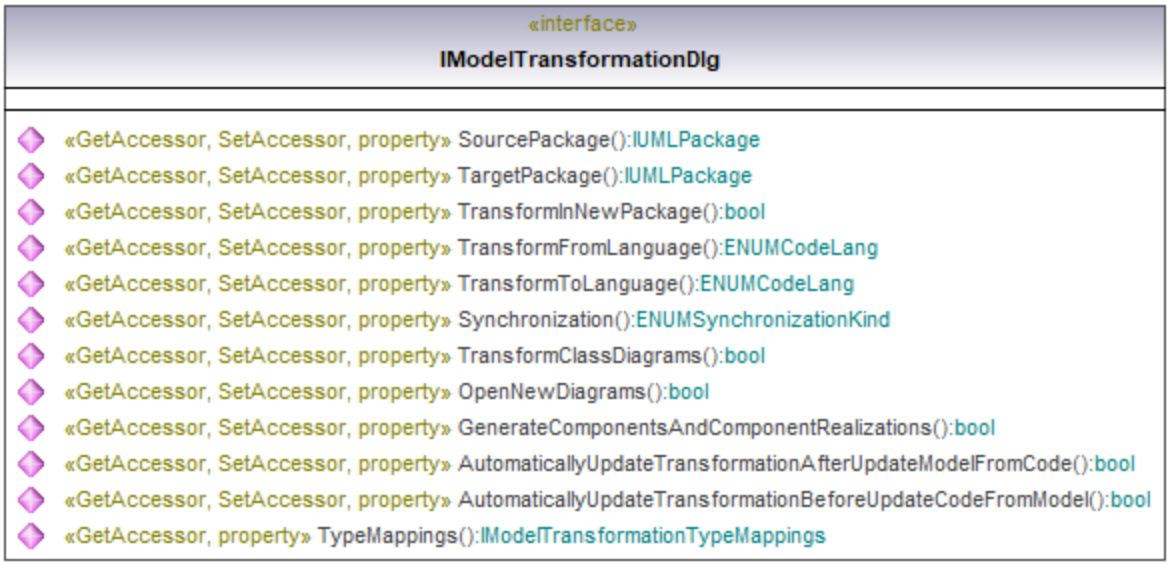
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			
documentation	Returns an array of values of type **string**.					

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

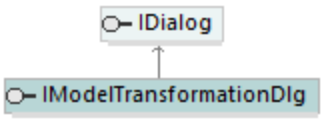
17.5.2.35 UModelAPI - IModelTransformationDlg

Interface **IModelTransformationDlg**

diagram



hierarchy



typedElements

Interface IDialogs	Operation ModelTransformationDlg
Interface IDocument	Operation ModelTransformation

Operation **IModelTransformationDlg::AutomaticallyUpdateTransformationAfterUpdateModelFromCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::AutomaticallyUpdateTransformationBeforeUpdateCodeFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::GenerateComponentsAndComponentRealizations**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**bool**

Operation **IModelTransformationDlg::OpenNewDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::SourcePackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IModelTransformationDlg::Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSynchronizationKind**			

Operation **IModelTransformationDlg::TargetPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IModelTransformationDlg::TransformClassDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::TransformFromLanguage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLang**			

Operation **IModelTransformationDlg::TransformInNewPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::TransformToLanguage**

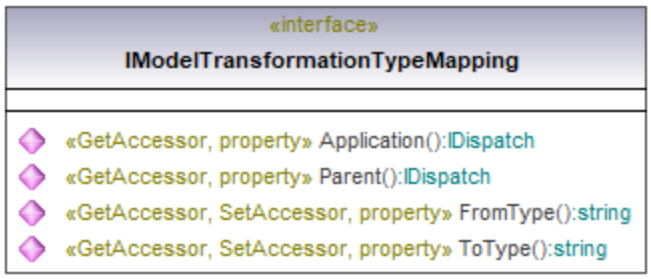
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLang**			

Operation **IModelTransformationDlg::TypeMappings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IModelTransformationTypeMappings**			

17.5.2.36 UModelAPI - IModelTransformationTypeMapping

Interface **IModelTransformationTypeMapping**

diagram	
typedElements	Interface **IModelTransformationTypeMappings** Operation **InsertItemAt Item**

Operation **IModelTransformationTypeMapping::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IModelTransformationTypeMapping::FromType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IModelTransformationTypeMapping::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

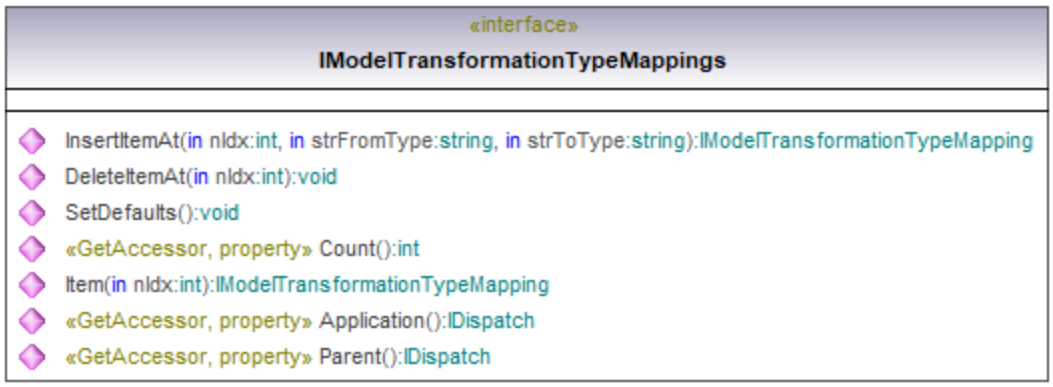
Operation **IModelTransformationTypeMapping::ToType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.37 UModelAPI - IModelTransformationTypeMappings

Interface **IModelTransformationTypeMappings**

diagram	
typedElements	Interface IModelTransformationDlg Operation TypeMappings

Operation **IModelTransformationTypeMappings::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IModelTransformationTypeMappings::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IModelTransformationTypeMappings::DeleteItemAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IModelTransformationTypeMappings::InsertItemAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strFromType	**in**	**string**			
	strToType	**in**	**string**			
	return	**return**	**IModelTransformationTypeMapping**			

Operation **IModelTransformationTypeMappings::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IModelTransformationTypeMapping**			

Operation **IModelTransformationTypeMappings::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

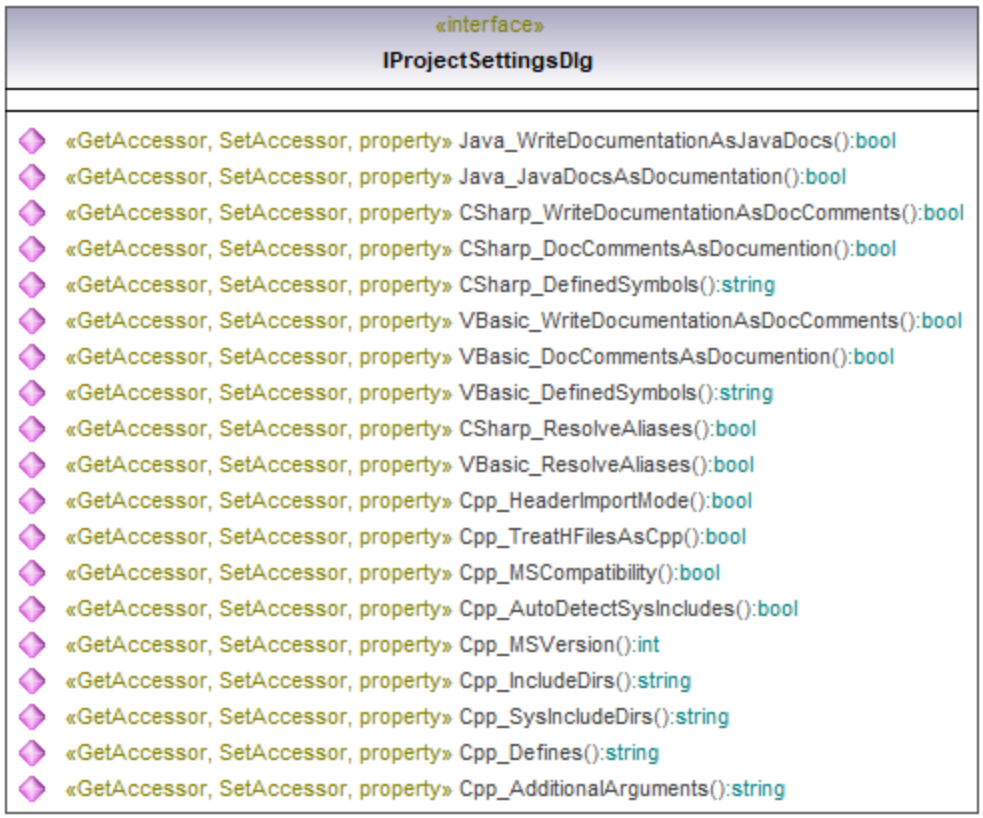
Operation **IModelTransformationTypeMappings::SetDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

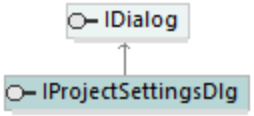
17.5.2.38 UModelAPI - IProjectSettingsDlg

Interface **IProjectSettingsDlg**

diagram



hierarchy



typedElements		
	Interface **IDialogs**	Operation **ProjectSettingsDlg**
	Interface **IDocument**	Operation **ProjectSettings**

Operation **IProjectSettingsDlg::Cpp_AdditionalArguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::Cpp_AutoDetectSysIncludes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Cpp_Defines**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::Cpp_HeaderImportMode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Cpp_IncludeDirs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::Cpp_MSCompatibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Cpp_MSVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IProjectSettingsDlg::Cpp_SysIncludeDirs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::Cpp_TreatHFilesAsCpp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::CSharp_DefinedSymbols**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::CSharp_DocCommentsAsDocumention**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::CSharp_ResolveAliases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::CSharp_WriteDocumentationAsDocComments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Java_JavaDocsAsDocumentation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Java_WriteDocumentationAsJavaDocs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::VBasic_DefinedSymbols**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::VBasic_DocCommentsAsDocumention**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::VBasic_ResolveAliases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::VBasic_WriteDocumentationAsDocComments**

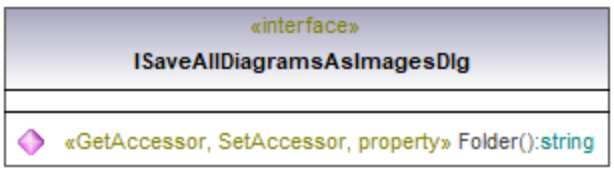
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

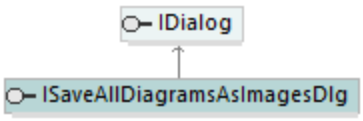
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.39 UModelAPI - ISaveAllDiagramsAsImagesDlg

Interface **ISaveAllDiagramsAsImagesDlg**

diagram	



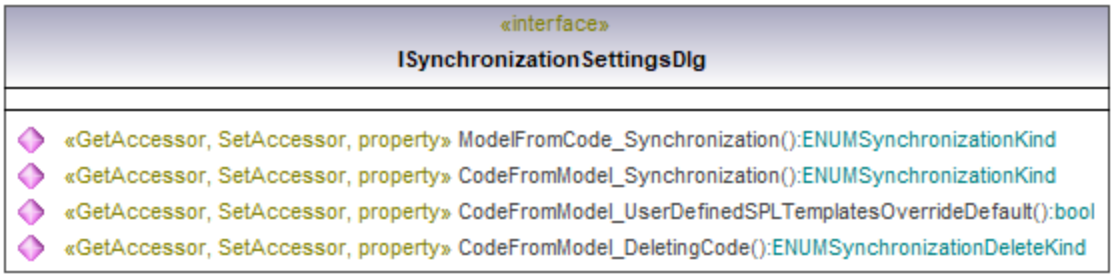
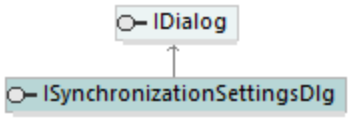
hierarchy	
typedElements	Interface IDialogs Interface IDocument Operation SaveAllDiagramsAsImagesDlg Operation SaveAllDiagramsAsImages

Operation **ISaveAllDiagramsAsImagesDlg::Folder**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.2.40 UModelAPI - ISynchronizationSettingsDlg

Interface **ISynchronizationSettingsDlg**

diagram	
hierarchy	
typedElements	Interface IDialogs Interface IDocument Operation SynchronizationSettingsDlg Operation SynchronizationSettings SynchronizeCodeFromModel SynchronizeModelFromCode

Operation **ISynchronizationSettingsDlg::CodeFromModel_DeletingCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMSynchronizationDeleteKind			

Operation **ISynchronizationSettingsDlg::CodeFromModel_Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSynchronizationKind**			

Operation **ISynchronizationSettingsDlg::CodeFromModel_UserDefinedSPLTemplatesOverrideDefault**

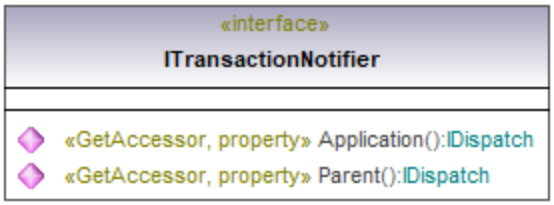
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ISynchronizationSettingsDlg::ModelFromCode_Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSynchronizationKind**			

17.5.2.41 UModelAPI - ITransactionNotifier

Interface **ITransactionNotifier**

diagram	
typedElements	Interface **IDocument** Operation **TransactionNotifier**
documentation	Use this interface to register for _ITransactionEvents.

Operation **ITransactionNotifier::Application**

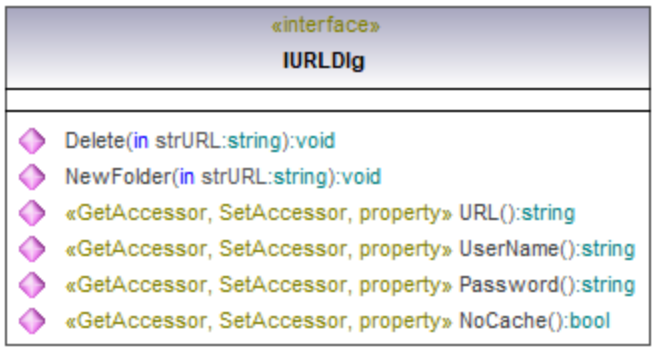
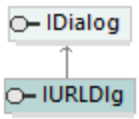
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ITransactionNotifier::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

17.5.2.42 UModelAPI - IURLDlg

Interface **IURLDlg**

diagram	
hierarchy	
typedElements	Interface **IApplication** Interface **IDialogs** Interface **IDocument** Interface **IExportXMIFileDlg** Interface **IIncludeSubprojectDlg** Operation **ImportFromXMIFileFromURL** **OpenDocumentFromURL** Operation **URLOpenDialog** **URLSaveDialog** Operation **MergeProjectFromURL** **SaveToURL** Operation **URLDlg** Operation **URLDlg**

Operation **IURLDlg::Delete**

parameter	name	direction	type	type modifier	multiplicity	default
	strURL	**in**	**string**			
	return	**return**	**void**			

Operation **IURLDlg::NewFolder**

parameter	name	direction	type	type modifier	multiplicity	default
	strURL	**in**	**string**			
	return	**return**	**void**			

Operation **IURLDlg::NoCache**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IURLDlg::Password**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**string**

Operation **IURLDlg::URL**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IURLDlg::UserName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.43 Events

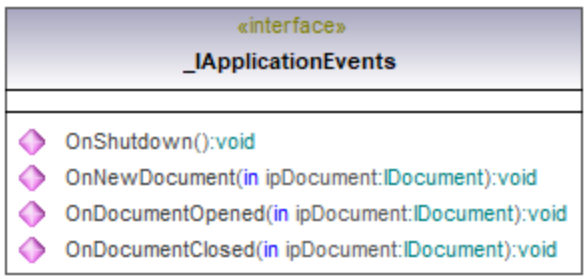
This is a list of all events sent by the UModel API.

A list of events sent on `UMLData` level can be found here.

17.5.2.43.1 UModelAPI - _IApplicationEvents

Interface **_IApplicationEvents**

diagram



Operation **_IApplicationEvents::OnDocumentClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IApplicationEvents::OnDocumentOpened**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IApplicationEvents::OnNewDocument**

parameter	name	direction	type	type modifier	multiplicity	default

	ipDocument	**in**	**IDocument**
	return	**return**	**void**

Operation **_IApplicationEvents::OnShutdown**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

17.5.2.43.2 UModelAPI - _IDiagramWindowEvents

Interface **_IDiagramWindowEvents**

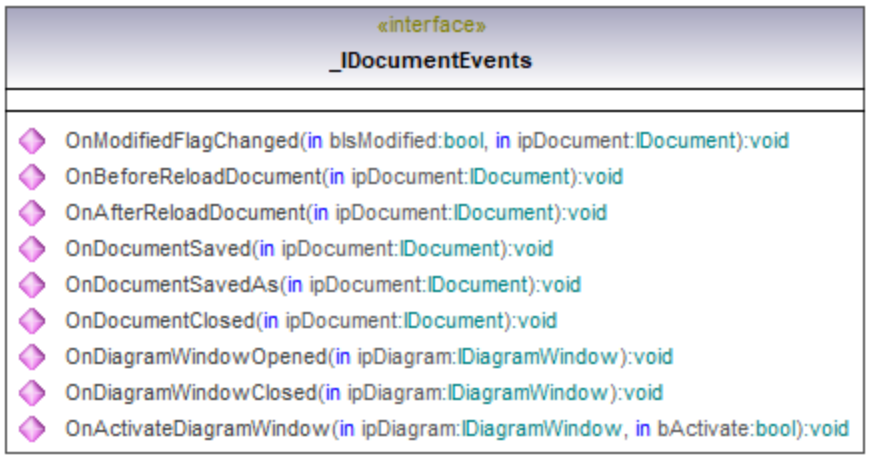
diagram

Operation **_IDiagramWindowEvents::OnDiagramWindowClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	**IDiagramWindow**			
	return	**return**	**void**			

17.5.2.43.3 UModelAPI - _IDocumentEvents

Interface **_IDocumentEvents**

diagram	

Operation **_IDocumentEvents::OnActivateDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	**IDiagramWindow**			
	bActivate	**in**	**bool**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnAfterReloadDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnBeforeReloadDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDiagramWindowClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	**IDiagramWindow**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDiagramWindowOpened**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	**IDiagramWindow**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDocumentClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDocumentSaved**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDocumentSavedAs**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

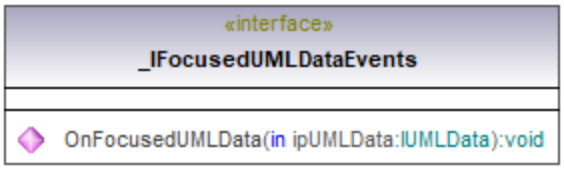
Operation **_IDocumentEvents::OnModifiedFlagChanged**

parameter	name	direction	type	type modifier	multiplicity	default
	bIsModified	**in**	**bool**			
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

17.5.2.43.4 UModelAPI - _IFocusedUMLDataEvents

Interface **_IFocusedUMLDataEvents**

diagram	

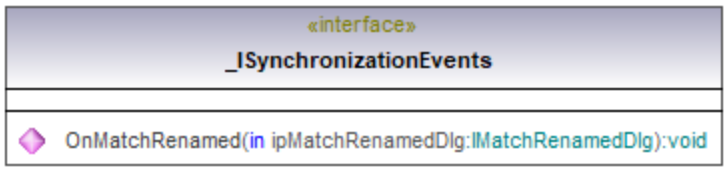


Operation **_IFocusedUMLDataEvents::OnFocusedUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	return	**return**	**void**			

17.5.2.43.5 UModelAPI - _ISynchronizationEvents

Interface **_ISynchronizationEvents**

diagram	

Operation **_ISynchronizationEvents::OnMatchRenamed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipMatchRenamedDlg	**in**	**IMatchRenamedDlg**			
	return	**return**	**void**			

17.5.2.43.6 UModelAPI - _ITransactionEvents

Interface **_ITransactionEvents**

diagram	
documentation	Use ITransactionNotifier to register for these events

Operation **_ITransactionEvents::OnBeginDataModification**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_ITransactionEvents::OnEndDataModification**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

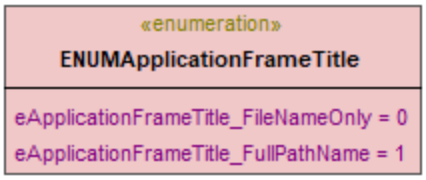
17.5.2.44 Enumerations

This is a list of all enumerations used by the UModel API. If your scripting environment does not support enumerations use the number-values instead.

A list of enumerations defined on `UMLData` level can be found here.

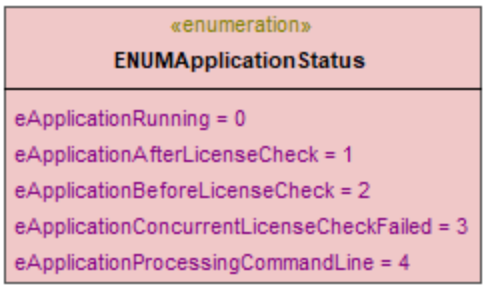
17.5.2.44.1 UModelAPI - ENUMApplicationFrameTitle

Enumeration **ENUMApplicationFrameTitle**

diagram	
typedElements	Interface ILocalOptionsView Operation FrameTitle

17.5.2.44.2 UModelAPI - ENUMApplicationStatus

Enumeration **ENUMApplicationStatus**

diagram	
typedElements	Interface IApplication Operation Status

17.5.2.44.3 UModelAPI - ENUMAutolayoutGrowDirectionKind

Enumeration **ENUMAutolayoutGrowDirectionKind**

diagram	«enumeration» **ENUMAutolayoutGrowDirectionKind** eAutolayoutGrowDirection_TopDown = 0 eAutolayoutGrowDirection_BottomUp = 1 eAutolayoutGrowDirection_LeftToRight = 2 eAutolayoutGrowDirection_RightToLeft = 3	
typedElements	Interface **ILocalOptionsView**	Operation **AutolayoutHierarchic_GrowDirection**

17.5.2.44.4 UModelAPI - ENUMCodeLang

Enumeration **ENUMCodeLang**

diagram	«enumeration» **ENUMCodeLang** eCodeLang_UML = -1 eCodeLang_Java = 0 eCodeLang_CSharp = 1 eCodeLang_XSD = 2 eCodeLang_VBasic = 3 eCodeLang_DB = 4 eCodeLang_Cpp = 5	
typedElements	Interface **IModelTransformationDlg**	Operation **TransformFromLanguage** **TransformToLanguage**
	Interface **IUMLComponent**	Operation **CodeLang**
	Interface **IUMLDataAll**	Operation **CodeLang** **IsCodeLangNamespace** **IsCodeLangNamespaceRoot**
	Interface **IUMLPackage**	Operation **IsCodeLangNamespace** **IsCodeLangNamespaceRoot**

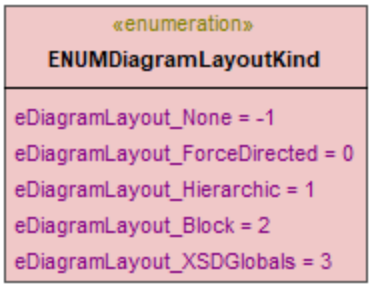
17.5.2.44.5 UModelAPI - ENUMCodeLangVersion

Enumeration **ENUMCodeLangVersion**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
typedElements	Interface **IImportSourceDlg** Interface **ILocalOptionsEditing** Interface **IUMLComponent** Interface **IUMLDataAll** Operation **Language** Operation **ComponentsDefaultCodeLangVersion** Operation **CodeLangVersion** Operation **CodeLangVersion**

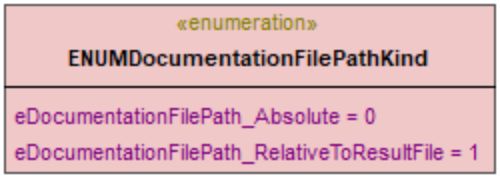
17.5.2.44.6 UModelAPI - ENUMDiagramLayoutKind

Enumeration **ENUMDiagramLayoutKind**

diagram	
typedElements	Interface **IDiagramWindow** Interface **IImportSourceDlg** Operation **Autolayout** **AutolayoutSelection** Operation **Content_Autolayout** **PackageDependency_Autolayout**

17.5.2.44.7 UModelAPI - ENUMDocumentationFilePathKind

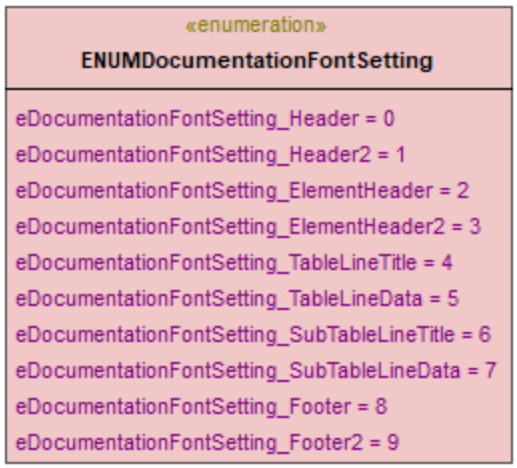
Enumeration **ENUMDocumentationFilePathKind**

diagram	

typedElements	Interface IGenerateDocumentationDlg	Operation GenerateLinksToLocalFiles

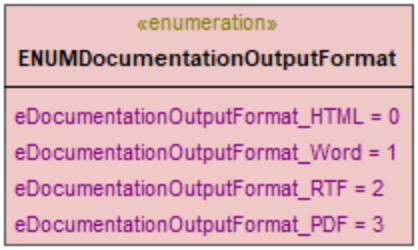
17.5.2.44.8 UModelAPI - ENUMDocumentationFontSetting

Enumeration **ENUMDocumentationFontSetting**

diagram	
typedElements	Interface IGenerateDocumentationDlg Operation Fonts_GetFace Fonts_GetSize Fonts_GetTextColor Fonts_IsBold Fonts_IsItalic Fonts_IsUnderline Fonts_SetBold Fonts_SetFace Fonts_SetItalic Fonts_SetSize Fonts_SetTextColor Fonts_SetUnderline

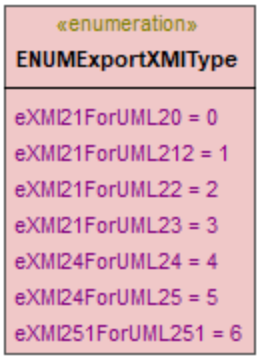
17.5.2.44.9 UModelAPI - ENUMDocumentationOutputFormat

Enumeration **ENUMDocumentationOutputFormat**

diagram	
typedElements	Interface **IGenerateDocumentationDlg** Operation **OutputFormat**

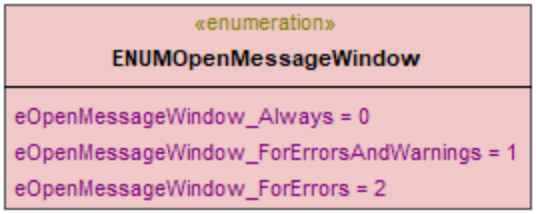
17.5.2.44.10 UModelAPI - ENUMExportXMIType

Enumeration **ENUMExportXMIType**

diagram	
typedElements	Interface **IExportXMIFileDlg** Operation **XMIType**

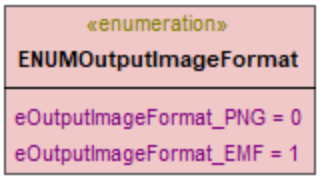
17.5.2.44.11 UModelAPI - ENUMOpenMessageWindow

Enumeration **ENUMOpenMessageWindow**

diagram	
typedElements	Interface **ILocalOptionsCodeEngineering** Operation **OpenMessageWindow**

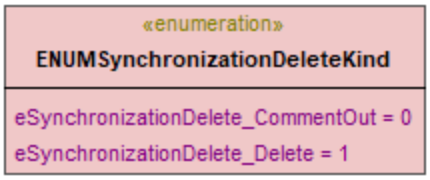
17.5.2.44.12 UModelAPI - ENUMOutputImageFormat

Enumeration **ENUMOutputImageFormat**

diagram	
typedElements	Interface **IDiagramWindow** Operation **SaveDiagramAsImage** Interface **IGenerateDocumentationDlg** Operation **DiagramImageFormat**

17.5.2.44.13 UModelAPI - ENUMSynchronizationDeleteKind

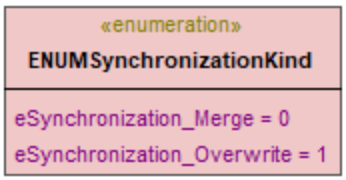
Enumeration **ENUMSynchronizationDeleteKind**

diagram	

typedElements	Interface **ISynchronizationSettingsDlg**	Operation **CodeFromModel_DeletingCode**

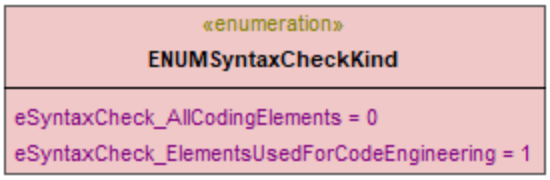
17.5.2.44.14 UModelAPI - ENUMSynchronizationKind

Enumeration **ENUMSynchronizationKind**

diagram	
typedElements	Interface **IImportSourceDlg** Operation **Synchronization** Interface **IModelTransformationDlg** Operation **Synchronization** Interface **ISynchronizationSettingsDlg** Operation **CodeFromModel_Synchronization** Operation **ModelFromCode_Synchronization**

17.5.2.44.15 UModelAPI - ENUMSyntaxCheckKind

Enumeration **ENUMSyntaxCheckKind**

diagram	
typedElements	Interface **ILocalOptionsCodeEngineering** Operation **SyntaxCheck**

17.5.3 UMLData Interfaces

The UMLData interfaces allow direct UML-level access to a UModel document. Using these interfaces, you can read and directly modify the UML representation of the document.

IUMLData is the common base interface of IUMLElement and IUMLGuiElement.

IUMLElements contains elements as defined by the UML specification (see http://www.uml.org).

IUMLGuiElements contains Altova-specific elements for diagrams, and members used to show IUMLElements on diagrams.

For examples of modifying UML elements and GUI elements, see Object model UMLData.

Errors

The `IUMLData` interfaces may return the API error codes listed below.

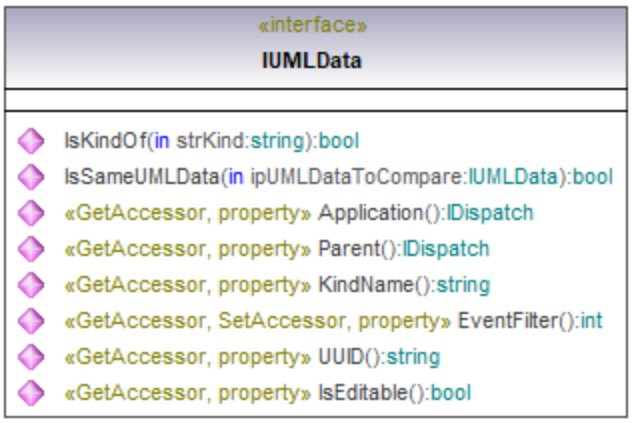
1000	The application object is no longer valid.
1001	Invalid parameter or invalid address for the return parameter was specified.
1002	UModel API is not available in the current edition.
1400	Invalid UMLData modification.
1401	Invalid Waypoint modification.
1402	No changes allowed.
1403	No changes allowed during Undo/Redo.
1404	Element is hidden by Element Style (visibility).
1405	Predefined element not found.
1406	Predefined element is of invalid kind.

For the error codes specific to the UModel API in general, see UModel API Errors.

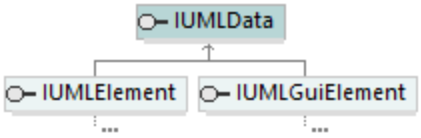
17.5.3.1 UModelAPI - IUMLData

Interface **IUMLData**

diagram



hierarchy



typedElements

Interface **_IFocusedUMLDataEvents**	Operation **OnFocusedUMLData**
Interface **_IUMLDataEvents**	Operation **OnAfterAddChild** **OnBeforeErase** **OnChanged** **OnMoveData**
Interface **IApplication**	Operation **LogMessageWithUMLDataLink**
Interface **IDiagramWindow**	Operation **FocusedData**
Interface **IDocument**	Operation **CanFocusUMLDataInModelTree** **FocusedUMLData** **FocusUMLDataInModelTree**
Interface **IUMLCommentTextHyperlink**	Operation **SetHyperlinkModelElementAddress**
Interface **IUMLData**	Operation **IsSameUMLData**
Interface **IUMLDataAll**	Operation **AddUMLGuiNodeLink** **FindPredefinedOwnedElement** **InsertOwnedDiagramAt** **InsertOwnedHyperlink2ModelAt** **IsSameUMLData** **LinkedModelElement** **SetHyperlinkModelElementAddress**
Interface **IUMLDataList**	Operation **ContainsUMLData** **Item**
Interface **IUMLElement**	Operation **FindPredefinedOwnedElement**
Interface **IUMLGuiDiagram**	Operation **AddUMLGuiNodeLink**
Interface **IUMLGuiRootElement**	Operation **InsertOwnedDiagramAt**

	Interface **IUMLGuiTextHyperlink** Interface **IUMLHyperlink2Model** Interface **IUMLNamedElement**	Operation **SetHyperlinkModelElementAddress** Operation **LinkedModelElement** Operation **InsertOwnedHyperlink2ModelAt**

Operation **IUMLData::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLData::EventFilter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLData::IsEditable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLData::IsKindOf**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**bool**			

Operation **IUMLData::IsSameUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLDataToCompare	**in**	**IUMLData**			
	return	**return**	**bool**			

Operation **IUMLData::KindName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLData::Parent**

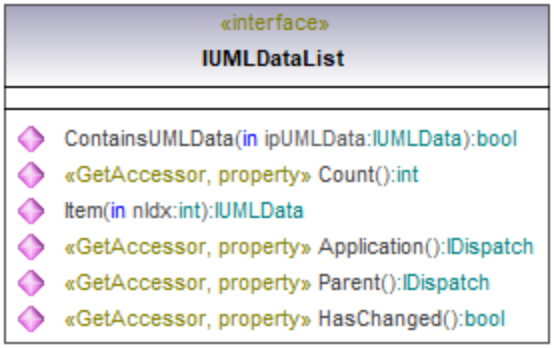
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLData::UUID**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.3.2 UModelAPI - IUMLDataList

Interface **IUMLDataList**

diagram	
typedElements	Interface **IDiagramWindow** Operation **SelectedGuiElements** Interface **IUMLAcceptEventAction** Operation **ActionTriggers** **EventActionResults** Interface **IUMLAction** Operation **InputPins** **LocalPostConditions** **LocalPreConditions** **OutputPins** Interface **IUMLActivity** Operation **ActivityEdges** **ActivityGroups** **ActivityNodes** Interface **IUMLActivityEdge** Operation **ActivityPartitions** **InterruptibleActivityRegions** **StructuredActivityNodes** Interface **IUMLActivityGroup** Operation **ContainedEdges** **ContainedNodes** **SubGroups** Interface **IUMLActivityNode** Operation **IncomingEdges** **OutgoingEdges** Interface **IUMLActivityPartition** Operation **Edges** **Nodes** **SubPartitions** Interface **IUMLArtifact** Operation **Manifestations** **NestedArtifacts** **OwnedAttributes** **OwnedOperations** Interface **IUMLAssociation** Operation **EndTypes** **MemberEnds** **NavigableOwnedEnds** **OwnedEnds** Interface **IUMLBehavior** Operation **OwnedParameters** **Postconditions** **Preconditions** Interface **IUMLBehavioralFeature** Operation **Methods** **OwnedParameters** **RaisedExceptions** Interface **IUMLBehavioredClassifier** Operation **InterfaceRealizations** **OwnedBehaviors** Interface **IUMLCallAction** Operation **Results** Interface **IUMLClass** Operation **NestedClassifiers** **OwnedOperations** **OwnedReceptions** **SuperClasses** Interface **IUMLClassifier** Operation **Attributes** **CollaborationUses** **Features** **Generalizations** **Generals** **InheritedMembers** **OwnedUseCases** **Specifics** **UseCases** Interface **IUMLClassifierTemplateParameter** Operation **ConstrainingClassifiers** Interface **IUMLCollaboration** Operation **CollaborationRoles** Interface **IUMLCollaborationUse** Operation **RoleBindings** Interface **IUMLCombinedFragment** Operation **Operands**

	Interface **IUMLComment**	Operation **AnnotatedElements** **OwnedHyperlinks**
	Interface **IUMLComponent**	Operation **Realizations**
	Interface **IUMLConnectionPointReference**	Operation **Entries** **Exits**
	Interface **IUMLConstraint**	Operation **ConstrainedElements**
	Interface **IUMLDataAll**	Operation **ActionTriggers** **ActivityEdges** **ActivityGroups** **ActivityNodes** **ActivityPartitions** **ActualGates** **AllAppliableStereotypes** **AllWaypoints** **AnnotatedElements** **AppliedStereotypes** **Arguments** **AttachedNodes** **Attributes** **ClientDependencies** **Clients** **CollaborationRoles** **CollaborationUses** **ConnectionPoints** **Connections** **ConstrainedElements** **ConstrainingClassifiers** **ContainedEdges** **ContainedNodes** **Conveyed** **DeployedElements** **Deployments** **Edges** **ElementImports** **EndTypes** **Entries** **EventActionResults** **ExceptionHandlers** **ExceptionTypes** **Exits** **Extends** **ExtensionLocations** **ExtensionPoints** **Features** **FeaturingClassifiers** **FormalGates** **Fragments** **Generalizations** **Generals** **GetOwnedElementsOfKind** **GuiLinks** **Handlers** **ImportedMembers** **Includes** **IncomingEdges** **Incomings** **InformationFlowRealizations** **InformationSources** **InformationTargets** **InheritedMembers** **InputElements** **InputPins** **InputValues** **InStates** **InterfaceRealizations** **InterruptibleActivityRegions** **InterruptingEdges** **Layers** **Lifelines** **LineConnectionWaypoints** **LineLinks** **LocalPostConditions** **LocalPreConditions** **LowerValues** **Manifestations** **MemberEnds** **Members** **Messages** **Methods** **NavigableOwnedEnds** **NestedArtifacts** **NestedClassifiers** **NestedNodes** **NestedPackages** **Nodes** **Observations** **Operands** **OutgoingEdges** **Outgoings** **OutputElements** **OutputPins** **OutputValues** **OwnedArguments** **OwnedAttributes** **OwnedBehaviors** **OwnedComments**

		OwnedConnectors OwnedDiagrams OwnedElements OwnedEnds OwnedGuiElements OwnedGuiNodeLinks OwnedHyperlinks OwnedLiterals OwnedMembers OwnedOperations OwnedParameters OwnedPorts OwnedReceptions OwnedRules OwnedStereotypes OwnedTemplateBindings OwnedTemplateParameters OwnedTypes OwnedUseCases PackagedElements PackageImports PackageMerges ParameterSubstitutions Postconditions Preconditions ProfileApplications Qualifiers RaisedExceptions Realizations RealizingConnectors Referred Regions RelatedElements RelativeNodes Results RoleBindings Slots Sources Specifics StereotypeApplications StructuredActivityNodes SubGroups Subjects SubmachineStates SubPartitions SubVertices SuperClasses SupplierDependencies Suppliers Targets TextLabels Transitions Triggers TypedElements UpperValues UseCases Values Waypoints
	Interface IUMLDataType	Operation OwnedAttributes OwnedOperations
	Interface IUMLDependency	Operation Clients Suppliers
	Interface IUMLDeploymentTarget	Operation DeployedElements Deployments
	Interface IUMLDirectedRelationship	Operation Sources Targets
	Interface IUMLDuration	Operation Observations
	Interface IUMLElement	Operation AllAppliableStereotypes AppliedStereotypes GetOwnedElementsOfKind OwnedComments OwnedElements StereotypeApplications
	Interface IUMLEncapsulatedClassifier	Operation OwnedPorts
	Interface IUMLEnumeration	Operation OwnedLiterals
	Interface IUMLExceptionHandler	Operation ExceptionTypes
	Interface IUMLExecutableNode	Operation Handlers
	Interface IUMLExpansionRegion	Operation InputElements OutputElements
	Interface IUMLExpression	Operation Operands
	Interface IUMLExtend	Operation ExtensionLocations
	Interface IUMLFeature	Operation FeaturingClassifiers
	Interface IUMLGuiDiagram	Operation GuiLinks Layers OwnedHyperlinks
	Interface IUMLGuiElement	Operation OwnedGuiElements

	Interface **IUMLGuiLineLink**	Operation **AllWaypoints** **LineConnectionWaypoints** **Waypoints**
	Interface **IUMLGuiLink**	Operation **AttachedNodes** **RelativeNodes**
	Interface **IUMLGuiNodeLink**	Operation **OwnedGuiNodeLinks**
	Interface **IUMLGuiNote**	Operation **OwnedHyperlinks**
	Interface **IUMLGuiRootElement**	Operation **OwnedDiagrams**
	Interface **IUMLGuiTextLabelWaypoint**	Operation **TextLabels**
	Interface **IUMLGuiWaypoint**	Operation **LineLinks**
	Interface **IUMLInformationFlow**	Operation **Conveyed** **InformationFlowRealizations** **InformationSources** **InformationTargets** **RealizingConnectors**
	Interface **IUMLInstanceSpecification**	Operation **Slots**
	Interface **IUMLInteraction**	Operation **FormalGates** **Fragments** **Lifelines** **Messages**
	Interface **IUMLInteractionUse**	Operation **ActualGates**
	Interface **IUMLInterface**	Operation **NestedClassifiers** **OwnedAttributes** **OwnedOperations** **OwnedReceptions**
	Interface **IUMLInterruptibleActivityRegion**	Operation **InterruptingEdges** **Nodes**
	Interface **IUMLInvocationAction**	Operation **Arguments**
	Interface **IUMLMessage**	Operation **OwnedArguments**
	Interface **IUMLMultiplicityElement**	Operation **LowerValues** **UpperValues**
	Interface **IUMLNamedElement**	Operation **ClientDependencies** **OwnedHyperlinks** **SupplierDependencies**
	Interface **IUMLNamespace**	Operation **ElementImports** **ImportedMembers** **Members** **OwnedMembers** **OwnedRules** **PackageImports** **PackageMerges**
	Interface **IUMLNode**	Operation **NestedNodes**
	Interface **IUMLObjectNode**	Operation **ExceptionHandlers** **InStates**
	Interface **IUMLOpaqueAction**	Operation **InputValues** **OutputValues**
	Interface **IUMLPackage**	Operation **NestedPackages** **OwnedStereotypes** **OwnedTypes** **PackagedElements** **ProfileApplications**
	Interface **IUMLProperty**	Operation **Qualifiers**
	Interface **IUMLProtocolTransition**	Operation **Referred**
	Interface **IUMLRegion**	Operation **SubVertices** **Transitions**
	Interface **IUMLRelationship**	Operation **RelatedElements**
	Interface **IUMLSignal**	Operation **OwnedAttributes**
	Interface **IUMLSlot**	Operation **Values**
	Interface **IUMLState**	Operation **ConnectionPoints** **Connections** **Regions**
	Interface **IUMLStateMachine**	Operation **ConnectionPoints** **Regions** **SubmachineStates**
	Interface **IUMLStructuredActivityNode**	Operation **Edges** **Nodes**
	Interface **IUMLStructuredClassifier**	Operation **OwnedAttributes** **OwnedConnectors**
	Interface **IUMLTemplateableElement**	Operation **OwnedTemplateBindings**

	Interface **IUMLTemplateBinding** Interface **IUMLTemplateSignature** Interface **IUMLTimeExpression** Interface **IUMLTransition** Interface **IUMLType** Interface **IUMLUseCase** Interface **IUMLVertex**	Operation **ParameterSubstitutions** Operation **OwnedTemplateParameters** Operation **Observations** Operation **Triggers** Operation **TypedElements** Operation **Extends** **ExtensionPoints** **Includes** **Subjects** Operation **Incomings** **Outgoings**

Operation **IUMLDataList::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLDataList::ContainsUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	return	**return**	**bool**			

Operation **IUMLDataList::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataList::HasChanged**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataList::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLData**			

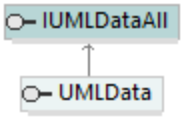
Operation **IUMLDataList::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

17.5.3.3 UModelAPI - IUMLDataAll

Interface **IUMLDataAll**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*

hierarchy	

Operation **IUMLDataAll::Abstraction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComponent**			

Operation **IUMLDataAll::ActionContext**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::ActionInputPin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActionInputPin**			

Operation **IUMLDataAll::ActionTriggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::ActiveLayer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagramLayer**			

Operation **IUMLDataAll::Activity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity**			

Operation **IUMLDataAll::ActivityEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ActivityGroups**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList**

Operation **IUMLDataAll::ActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ActivityPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Actual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLDataAll::ActualGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Addition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::AddOwnedGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	**in**	**IUMLGuiNodeLink**			
	return	**return**	**void**			

Operation **IUMLDataAll::AddUMLElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNodeLink**			

Operation **IUMLDataAll::AddUMLGuiContainmentLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	**in**	**IUMLGuiLink**			
	ipToLink	**in**	**IUMLGuiLink**			
	return	**return**	**IUMLGuiContainmentLink**			

Operation **IUMLDataAll::AddUMLGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	**in**	**IUMLData**			
	nLeft	**in**	**int**			

	nTop	**in**	**int**
	return	**return**	**IUMLGuiNodeLink**

Operation **IUMLDataAll::AddUMLGuiNote**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNote**			

Operation **IUMLDataAll::AddUMLGuiNoteLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote**			
	ipToLink	**in**	**IUMLGuiNodeLink**			
	return	**return**	**IUMLGuiNoteLink**			

Operation **IUMLDataAll::AddUMLGuiNoteLinkToLine**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote**			
	ipToLink	**in**	**IUMLGuiLineLink**			
	nDistanceFromLineBegin	**in**	**int**			
	return	**return**	**IUMLGuiNoteLink**			

Operation **IUMLDataAll::AddUMLLineElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	ipFromNode	**in**	**IUMLGuiNodeLink**			
	ipToNode	**in**	**IUMLGuiNodeLink**			
	return	**return**	**IUMLGuiLineLink**			

Operation **IUMLDataAll::Aggregation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLAggregationKind**			

Operation **IUMLDataAll::Alias**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::AllAppliableStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::AllowSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::AllWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::AnnotatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLDataAll::AppliedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLDataAll::AppliedProfile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProfile**			

Operation **IUMLDataAll::AppliedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ApplyingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::ApplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLDataAll::ApplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLDataAll::Arguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Association**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

Operation **IUMLDataAll::AssociationEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLDataAll::AttachedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::AttachedTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiLink**			

Operation **IUMLDataAll::Attributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::BaseClass**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::BeginOffset**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Behavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::BehaviorExecution**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::BehaviorSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavioralFeature**			

Operation **IUMLDataAll::Body**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::BooleanValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::Bottom**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::BoundElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateableElement**			

Operation **IUMLDataAll::CallOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLDataAll::CallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::ChangeExpression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass**			

Operation **IUMLDataAll::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::ClientDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Clients**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::CodeLang**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLang**			

Operation **IUMLDataAll::CodeLangVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion**			

Operation **IUMLDataAll::CodeOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLDataAll::CodeProjectFileOrDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::CollaborationRoles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::CollaborationType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLCollaboration**			

Operation **IUMLDataAll::CollaborationUses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Comment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Concurrency**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLCallConcurrencyKind**			

Operation **IUMLDataAll::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Connections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ConnectorKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLConnectorKind**			

Operation **IUMLDataAll::ConnectorType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

Operation **IUMLDataAll::ConstrainedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ConstrainingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ConstrainingPointX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::ConstrainingPointY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::ContainedEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ContainedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Container**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLRegion**			

Operation **IUMLDataAll::Context**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLDataAll::Contract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLDataAll::ContrainingAreaIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Conveyed**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLDataAll::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataType**			

Operation **IUMLDataAll::DecisionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::DefaultLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::DefaultParamValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::DefiningFeature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStructuralFeature**			

Operation **IUMLDataAll::DeployedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Deployments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Direction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLParameterDirectionKind**			

Operation **IUMLDataAll::DistanceFromLineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::DoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Effect**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Element**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLDataAll::ElementImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::EndOffset**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::EndTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Entries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Enumeration**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEnumeration**			

Operation **IUMLDataAll::EraseAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseFromDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	**IUMLGuiElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	return	**return**	**void**

Operation **IUMLDataAll::EraseInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::Event**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEvent**			

Operation **IUMLDataAll::EventActionResults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::EventFilter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::ExceptionHandlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ExceptionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLObjectNode**			

Operation **IUMLDataAll::ExceptionTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ExecutionSpecificationFinish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutionSpecification**			

Operation **IUMLDataAll::ExecutionSpecificationStart**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutionSpecification**			

Operation **IUMLDataAll::Exit**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Exits**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList**

Operation **IUMLDataAll::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::Expression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpression**			

Operation **IUMLDataAll::ExtendedCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::Extends**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Extension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::ExtensionLocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ExtensionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Features**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::FeaturingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::FileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::FindOwnedMemberWithQualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default

	strName	**in**	**string**
	return	**return**	**IUMLNamedElement**

Operation **IUMLDataAll::FindPredefinedOwnedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	**ENUMUMLPredefinedElement**			
	bRecursive	**in**	**bool**			
	return	**return**	**IUMLData**			

Operation **IUMLDataAll::Finish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOccurrenceSpecification**			

Operation **IUMLDataAll::Formal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

Operation **IUMLDataAll::FormalGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Fragments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::General**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::Generalizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Generals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::GeneralValueLifelineNameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	return	**return**	**string**

Operation **IUMLDataAll::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IUMLDataAll::GetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::GetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	bWithBrackets	**in**	**bool**			
	return	**return**	**string**			

Operation **IUMLDataAll::GetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLDataAll::GetOwnedElementsOfKind**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	bRecursive	**in**	**bool**			
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::GetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::GetSourceLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLDataAll::GetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::GetStereotypeApplicationForPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLDataAll::GetStereotypeApplicationForStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**IUMLStereotype Application**			

Operation **IUMLDataAll::GetTargetLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLDataAll::GetTextLabelText**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	return	**return**	**string**			

Operation **IUMLDataAll::GetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::GetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nStateIndex	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::GetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::Guard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::GuiLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::GuiOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			

Operation **IUMLDataAll::HandlerBody**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLExecutableNode**

Operation **IUMLDataAll::Handlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::HSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::IconFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::ImplementingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavioredClassifier**			

Operation **IUMLDataAll::ImportedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLDataAll::ImportedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ImportedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::ImportingNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLDataAll::InActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity**			

Operation **IUMLDataAll::Includes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::IncludingCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::IncomingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Incomings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InformationFlowRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InformationSources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InformationTargets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InheritedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InputElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InsertActionTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger**			

Operation **IUMLDataAll::InsertActivityEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLActivityNode**			
	ipTo	**in**	**IUMLActivityNode**			
	return	**return**	**IUMLActivityEdge**			

Operation **IUMLDataAll::InsertActivityGroupAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityGroup**			

Operation **IUMLDataAll::InsertActivityNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityNode**			

Operation **IUMLDataAll::InsertActualGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGate**			

Operation **IUMLDataAll::InsertAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::InsertArgumentOfKindAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			

	return	**return**	**void**			

Operation **IUMLDataAll::InsertCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnectableElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertCollaborationUseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLCollaborationUse**			

Operation **IUMLDataAll::InsertConnectionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConnectionPointReference**			

Operation **IUMLDataAll::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPseudostate**			

Operation **IUMLDataAll::InsertConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	ipVal	**in**	**IUMLInteractionFragment**
	return	**return**	**void**

Operation **IUMLDataAll::InsertDeploymentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDeployedArtifact	**in**	**IUMLDeployedArtifact**			
	return	**return**	**IUMLDeployment**			

Operation **IUMLDataAll::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertElementImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedElement	**in**	**IUMLPackageableElement**			
	return	**return**	**IUMLElementImport**			

Operation **IUMLDataAll::InsertEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertEventActionResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLDataAll::InsertExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate**			

	return	**return**	**void**

Operation **IUMLDataAll::InsertExtendAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipExtendedCase	**in**	**IUMLUseCase**			
	return	**return**	**IUMLExtend**			

Operation **IUMLDataAll::InsertExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pExtensionLocation	**in**	**IUMLExtensionPoint**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertExtensionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExtensionPoint**			

Operation **IUMLDataAll::InsertFormalGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGate**			

Operation **IUMLDataAll::InsertFragmentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLInteractionFragment**			

Operation **IUMLDataAll::InsertGeneralizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipGeneral	**in**	**IUMLClassifier**			
	return	**return**	**IUMLGeneralization**			

Operation **IUMLDataAll::InsertHandlerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExceptionHandler**			

Operation **IUMLDataAll::InsertIncludeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	ipIncludingCase	**in**	**IUMLUseCase**
	return	**return**	**IUMLInclude**

Operation **IUMLDataAll::InsertInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLRelationship**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::InsertInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLState**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInterfaceRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipContract	**in**	**IUMLInterface**			
	return	**return**	**IUMLInterfaceRealization**			

Operation **IUMLDataAll::InsertInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertLayerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiDiagramLayer**			

Operation **IUMLDataAll::InsertLifelineAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLLifeline**			

Operation **IUMLDataAll::InsertLocalPostConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::InsertLocalPreConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::InsertLowerUpperValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strLower	**in**	**string**			
	strUpper	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertManifestationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipUtilizedElement	**in**	**IUMLPackageableElement**			
	return	**return**	**IUMLManifestation**			

Operation **IUMLDataAll::InsertMessageAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLMessage**			

Operation **IUMLDataAll::InsertNestedArtifactAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	return	**return**	**IUMLArtifact**

Operation **IUMLDataAll::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::InsertNestedNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLNode**			

Operation **IUMLDataAll::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLActivityNode**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLObservation**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertOperandAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInteractionOperand**			

Operation **IUMLDataAll::InsertOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertOutputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLDataAll::InsertOwnedArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	strKind	**in**	**string**
	return	**return**	**IUMLValueSpecification**

Operation **IUMLDataAll::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLDataAll::InsertOwnedBehaviorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::InsertOwnedCommentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLComment**			

Operation **IUMLDataAll::InsertOwnedCommentTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLCommentTextHyperlink**			

Operation **IUMLDataAll::InsertOwnedConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFrom	**in**	**IUMLConnectableElement**			
	ipTo	**in**	**IUMLConnectableElement**			
	return	**return**	**IUMLConnector**			

Operation **IUMLDataAll::InsertOwnedDiagramAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipUMLParent	**in**	**IUMLData**			
	strKind	**in**	**string**			
	return	**return**	**IUMLGuiDiagram**			

Operation **IUMLDataAll::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			

	return	**return**	**IUMLGuiTextHyperlink**

Operation **IUMLDataAll::InsertOwnedHyperlink2FileAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strFilePathOrUrl	**in**	**string**			
	return	**return**	**IUMLHyperlink2File**			

Operation **IUMLDataAll::InsertOwnedHyperlink2GuiElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedGuiElement	**in**	**IUMLGuiVisibleElement**			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement**			
	return	**return**	**IUMLHyperlink2GuiElement**			

Operation **IUMLDataAll::InsertOwnedHyperlink2ModelAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedData	**in**	**IUMLData**			
	return	**return**	**IUMLHyperlink2Model**			

Operation **IUMLDataAll::InsertOwnedLiteralAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLEnumerationLiteral**			

Operation **IUMLDataAll::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation**			

Operation **IUMLDataAll::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLParameter**			

Operation **IUMLDataAll::InsertOwnedPortAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPort**			

Operation **IUMLDataAll::InsertOwnedReceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLReception**			

Operation **IUMLDataAll::InsertOwnedRuleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::InsertOwnedTemplateBindingAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSignature	**in**	**IUMLTemplateSignature**			
	return	**return**	**IUMLTemplateBinding**			

Operation **IUMLDataAll::InsertOwnedTemplateParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLClassifierTemplateParameter**			

Operation **IUMLDataAll::InsertOwnedUseCaseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::InsertPackagedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLDataAll::InsertPackagedElementRelationshipAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLElement**			
	ipTo	**in**	**IUMLElement**			
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLDataAll::InsertPackageImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedPackage	**in**	**IUMLPackage**			

	return	**return**	**IUMLPackageImport**

Operation **IUMLDataAll::InsertPackageMergeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipMergedPackage	**in**	**IUMLPackage**			
	return	**return**	**IUMLPackageMerge**			

Operation **IUMLDataAll::InsertParameterSubstitutionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFormalParameter	**in**	**IUMLTemplateParameter**			
	ipActualParameter	**in**	**IUMLParameterableElement**			
	return	**return**	**IUMLTemplateParameterSubstitution**			

Operation **IUMLDataAll::InsertPostconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::InsertPreconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::InsertProfileApplicationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipAppliedProfile	**in**	**IUMLProfile**			
	return	**return**	**IUMLProfileApplication**			

Operation **IUMLDataAll::InsertQualifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLDataAll::InsertRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLType**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipRealizingClassifier	**in**	**IUMLClassifier**			
	return	**return**	**IUMLComponentRealization**			

Operation **IUMLDataAll::InsertRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnector**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLRegion**			

Operation **IUMLDataAll::InsertResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLDataAll::InsertSlotAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	**IUMLStructuralFeature**			
	return	**return**	**IUMLSlot**			

Operation **IUMLDataAll::InsertSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLDataAll::InsertSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pSubject	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertSubPartitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			

	return	**return**	**IUMLActivityPartition**

Operation **IUMLDataAll::InsertSubVertexAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLVertex**			

Operation **IUMLDataAll::InsertTransitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSource	**in**	**IUMLVertex**			
	ipTarget	**in**	**IUMLVertex**			
	return	**return**	**IUMLTransition**			

Operation **IUMLDataAll::InsertTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger**			

Operation **IUMLDataAll::InsertValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::InsertWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiWaypoint**			

Operation **IUMLDataAll::Instance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

Operation **IUMLDataAll::InStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::IntegerValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::InteractionOperator**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLInteractionOperatorKind**			

Operation **IUMLDataAll::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLDataAll::InterfaceRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InterruptibleActivityRegions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InterruptingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Invariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::IsAbstract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsActive**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsActivityReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsBehavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsCodeLangNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	**ENUMCodeLang**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsCodeLangNamespaceRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	**ENUMCodeLang**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsCodeProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsCombineDuplicate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsComputable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsConjugated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsControl**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsControlType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsDerived**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsDerivedUnion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsDimension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsEditable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipElement	**in**	**IUMLElement**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsExternal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsFinalSpecialization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsFirstEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsHorizontal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsIndirectlyInstantiated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsKindOf**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsLeaf**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsLocallyReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsLocked**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsMultiCast**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsMultiReceive**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsNavigable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsNull**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsOrdered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsOrthogonal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsOwnedEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsPositioned**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsPredefinedStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsQuery**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSameUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLDataToCompare	**in**	**IUMLData**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsService**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsShared**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsShowAsGeneralValueLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSimple**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsStatic**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSubmachineState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSynchronous**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsUnique**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsUnmarshall**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsUseForCodeEngineering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsVarArgList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::JoinSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::KindName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Label**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Layer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagramLayer**			

Operation **IUMLDataAll::Layers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Left**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Lifelines**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::LineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			

Operation **IUMLDataAll::LineConnectionWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::LineEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			

Operation **IUMLDataAll::LineLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::LinkedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiVisibleElement**			

Operation **IUMLDataAll::LinkedGuiElementCell**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamedElement**			

Operation **IUMLDataAll::LinkedModelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData**			

Operation **IUMLDataAll::LinkedOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLDataAll::LocalPostConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::LocalPreConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Location**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDeployment Target**			

Operation **IUMLDataAll::LowerValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Manifestations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Mapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueExpr ession**			

Operation **IUMLDataAll::Max**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecif ication**			

Operation **IUMLDataAll::MaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecif ication**			

Operation **IUMLDataAll::MemberEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Members**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::MergedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::MergeLayersAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromIdx	**in**	**int**			
	nToIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::Message**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessage**			

Operation **IUMLDataAll::MessageKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLMessageKind**			

Operation **IUMLDataAll::Messages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::MessageSort**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLMessageSort**			

Operation **IUMLDataAll::MetaClass**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Methods**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::MiddleWaypoint**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLGuiMiddleWaypoint**

Operation **IUMLDataAll::Min**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::MinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::Mode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLExpansionKind**			

Operation **IUMLDataAll::MoveTo**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::MustIsolate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::NameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Namespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLDataAll::NavigableOwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NestedArtifacts**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList**

Operation **IUMLDataAll::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NestedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NestedPackages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NestingInterface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLDataAll::NestingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NoteText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::NoteTextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::NoteTextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OccurringEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEvent**			

Operation **IUMLDataAll::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLDataAll::OperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteractionConstraint**			

Operation **IUMLDataAll::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Operation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLDataAll::Opposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLDataAll::Ordering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLObjectNodeOrderingKind**			

Operation **IUMLDataAll::OutgoingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Outgoings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OutputElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OutputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OutputValues**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList**

Operation **IUMLDataAll::OwnedActual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLDataAll::OwnedArguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedBehaviors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedComments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedDocComment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComment**			

Operation **IUMLDataAll::OwnedDocCommentBody**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::OwnedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList**

Operation **IUMLDataAll::OwnedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedGuiNodeLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedLiterals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameteredElement**			

Operation **IUMLDataAll::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedPorts**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedReceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedRules**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList**

Operation **IUMLDataAll::OwnedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedTemplateBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedTemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

Operation **IUMLDataAll::OwnedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedUseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Owner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLDataAll::OwningAssociation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

Operation **IUMLDataAll::OwningConstraint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::OwningElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLDataAll::OwningGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiNodeLink**			

Operation **IUMLDataAll::OwningInstance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

Operation **IUMLDataAll::OwningInstanceSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

Operation **IUMLDataAll::OwningLower**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMultiplicityElement**			

Operation **IUMLDataAll::OwningPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::OwningParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameter**			

Operation **IUMLDataAll::OwningProperty**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLDataAll::OwningSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

Operation **IUMLDataAll::OwningSlot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSlot**			

Operation **IUMLDataAll::OwningState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

Operation **IUMLDataAll::OwningTemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	IUMLTemplateParameter			

Operation **IUMLDataAll::OwningTransition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLProtocolTransition			

Operation **IUMLDataAll::OwningUpper**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLMultiplicityElement			

Operation **IUMLDataAll::Package**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage			

Operation **IUMLDataAll::PackagedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::PackageImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::PackageMerges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::Parameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLParameter			

Operation **IUMLDataAll::ParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLParameterableElement			

Operation **IUMLDataAll::ParameterSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateSignature			

Operation **IUMLDataAll::ParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLDataAll::PinValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::PostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::Postconditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::PostTypeModifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::PreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::Preconditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ProfileApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ProtectedNode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutableNode**			

Operation **IUMLDataAll::Protocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLProtocolStateMachine			

Operation **IUMLDataAll::PseudostateKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLPseudostateKind			

Operation **IUMLDataAll::QualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Qualifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::RaisedExceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::Realizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::RealizingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLClassifier			

Operation **IUMLDataAll::RealizingConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::ReceiveEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLMessageEnd			

Operation **IUMLDataAll::ReceivingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage			

Operation **IUMLDataAll::ReferencedDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiDiagram			

Operation **IUMLDataAll::Referred**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::RefersTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteraction**			

Operation **IUMLDataAll::RegionAsInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpansionRegion**			

Operation **IUMLDataAll::RegionAsOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpansionRegion**			

Operation **IUMLDataAll::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::RelatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::RelativeNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Represents**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConnectableElement**			

Operation **IUMLDataAll::Result**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOutputPin**			

Operation **IUMLDataAll::Results**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Right**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**int**

Operation **IUMLDataAll::Role**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConnectableElement**			

Operation **IUMLDataAll::RoleBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ScrollPosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::ScrollPosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Selection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Selector**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SendEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessageEnd**			

Operation **IUMLDataAll::SendSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

Operation **IUMLDataAll::SeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::SetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipElement	**in**	**IUMLElement**			
	bVisible	**in**	**bool**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetHyperlinkGuiElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedGuiElement	**in**	**IUMLGuiVisibleElement**			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	**in**	**IUMLData**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			

Operation **IUMLDataAll::SetNewActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewCallTarget**

parameter	name	direction	type	type modifier	multiplicity	default

	strKind	**in**	**string**
	return	**return**	**IUMLInputPin**

Operation **IUMLDataAll::SetNewChangeExpression**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLDataAll::SetNewDefaultValueLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

Operation **IUMLDataAll::SetNewDoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::SetNewEffect**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::SetNewEntry**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::SetNewExit**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::SetNewExpr**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::SetNewMapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueExpression**			

Operation **IUMLDataAll::SetNewMax**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewMaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewMin**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewMinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewOperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteractionConstraint**			

Operation **IUMLDataAll::SetNewOwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLDataAll::SetNewPostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::SetNewPreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::SetNewProtocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine**			

Operation **IUMLDataAll::SetNewSelector**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewSignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLDataAll::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

Operation **IUMLDataAll::SetNewStateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			

	return	**return**	**IUMLConstraint**

Operation **IUMLDataAll::SetNewTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

Operation **IUMLDataAll::SetNewTransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::SetNewWhen**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression**			

Operation **IUMLDataAll::SetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	**IUMLOperation**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetPos**

parameter	name	direction	type	type modifier	multiplicity	default
	x	**in**	**int**			
	y	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetPredefinedTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nProperty	**in**	**ENUMUMLPredefinedElement**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetRect**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	nRight	**in**	**int**			
	nBottom	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetScrollPos**

parameter	name	direction	type	type modifier	multiplicity	default
	nX	**in**	**int**			
	nY	**in**	**int**			

	return	**return**	**void**

Operation **IUMLDataAll::SetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	**IUMLStructuralFeature**			
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLDataAll::SetSlotValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	**IUMLStructuralFeature**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetStateIndexErased**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	**IUMLStructuralFeature**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default

	ipTextLabel	**in**	**IUMLGuiTextLabel**
	bVisible	**in**	**bool**
	return	**return**	**void**

Operation **IUMLDataAll::SetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nStateIndex	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::Signal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

Operation **IUMLDataAll::SignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::Signature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

Operation **IUMLDataAll::SingleExecution**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::Slots**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Source**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityNode**			

Operation **IUMLDataAll::Sources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Specific**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::Specification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::Specifics**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Start**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOccurrenceSpecification**			

Operation **IUMLDataAll::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

Operation **IUMLDataAll::StateCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::StateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine**			

Operation **IUMLDataAll::Stereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStereotype**			

Operation **IUMLDataAll::StereotypeApplications**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::StereotypedElementStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

Operation **IUMLDataAll::StringValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::StructuredActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Styles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

Operation **IUMLDataAll::SubGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Subjects**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Submachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine**			

Operation **IUMLDataAll::SubmachineStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::SubPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::SubVertices**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::SuperClasses**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList**

Operation **IUMLDataAll::SuperGroup**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityGroup**			

Operation **IUMLDataAll::SuperPartition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityPartition**			

Operation **IUMLDataAll::SupplierDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Suppliers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Symbol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Target**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityNode**			

Operation **IUMLDataAll::Targets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Template**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateableElement**			

Operation **IUMLDataAll::TemplateBinding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateBinding**			

Operation **IUMLDataAll::TemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	IUMLTemplateParameter			

Operation **IUMLDataAll::TextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::TextLabelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLElement			

Operation **IUMLDataAll::TextLabelKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLGuiTextLabelKind			

Operation **IUMLDataAll::TextLabels**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::TextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::TimeObservationEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLNamedElement			

Operation **IUMLDataAll::TimeTickLengthCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Top**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Transformation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior			

Operation **IUMLDataAll::TransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint			

Operation **IUMLDataAll::TransitionKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLTransitionKind**			

Operation **IUMLDataAll::Transitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::TransitionSource**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex**			

Operation **IUMLDataAll::TransitionTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex**			

Operation **IUMLDataAll::Triggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Type**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLType**			

Operation **IUMLDataAll::TypedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::UnapplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::UnapplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**void**			

Operation **IUMLDataAll::UnlimitedValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::UpperBound**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::UpperValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::UseCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::UseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::UseForForwardEngineering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::UserDefinedLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::UtilizedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLDataAll::UUID**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Value**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Values**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Viewpoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

Operation **IUMLDataAll::VisualStatePositionCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::VSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::Waypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Weight**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::When**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression**			

Operation **IUMLDataAll::ZoomFactor**

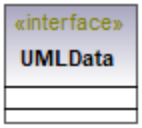
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

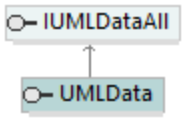
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.4 UModelAPI - UMLData

Interface **UMLData**

diagram	



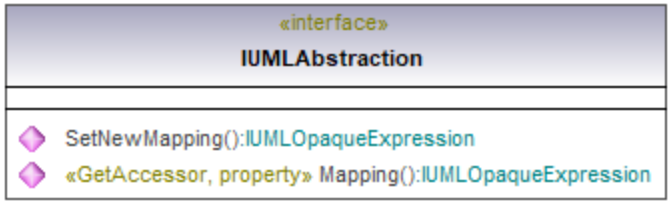
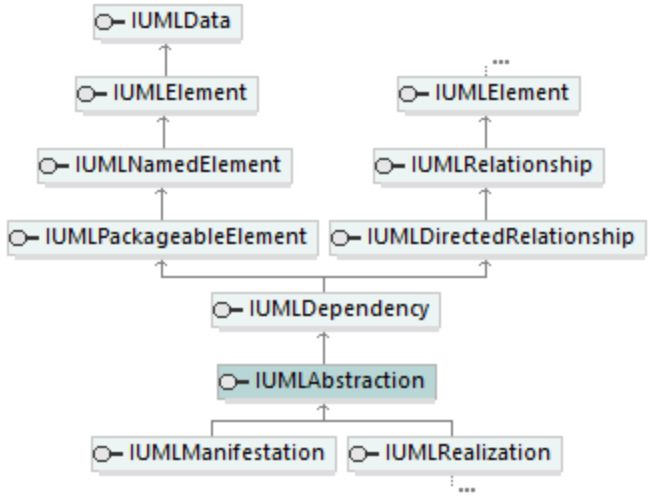
hierarchy	

17.5.3.5 IUMLElement

This is a list of elements as defined by OMG in the UML Specification, see http://www.uml.org.

17.5.3.5.1 UModelAPI - IUMLAbstraction

Interface **IUMLAbstraction**

diagram	
hierarchy	

Operation **IUMLAbstraction::Mapping**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLOpaqueExpression**			

Operation **IUMLAbstraction::SetNewMapping**

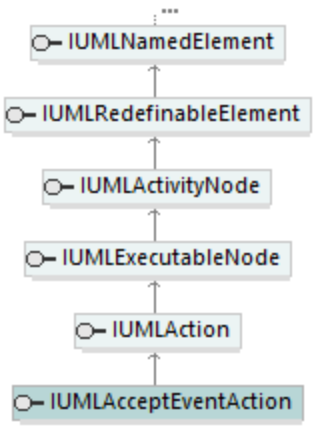
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueExpression**			

17.5.3.5.2 UModelAPI - IUMLAcceptEventAction

Interface **IUMLAcceptEventAction**

diagram

hierarchy



Operation **IUMLAcceptEventAction::ActionTriggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTrigger.					

Operation **IUMLAcceptEventAction::EventActionResults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOutputPin.					

Operation **IUMLAcceptEventAction::InsertActionTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger**			

Operation **IUMLAcceptEventAction::InsertEventActionResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLAcceptEventAction::IsUnmarshall**

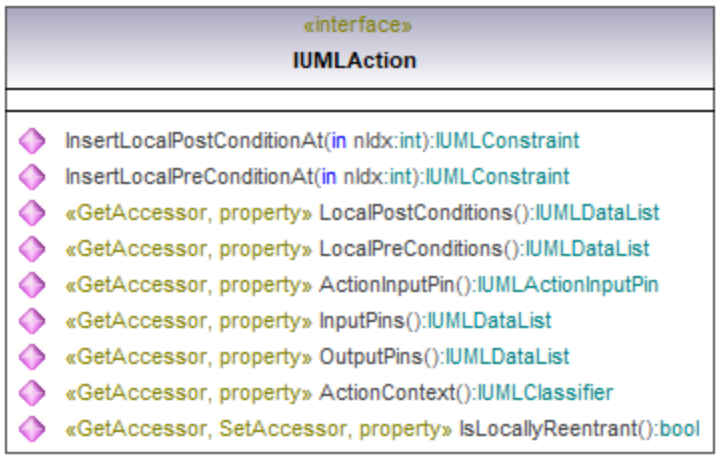
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

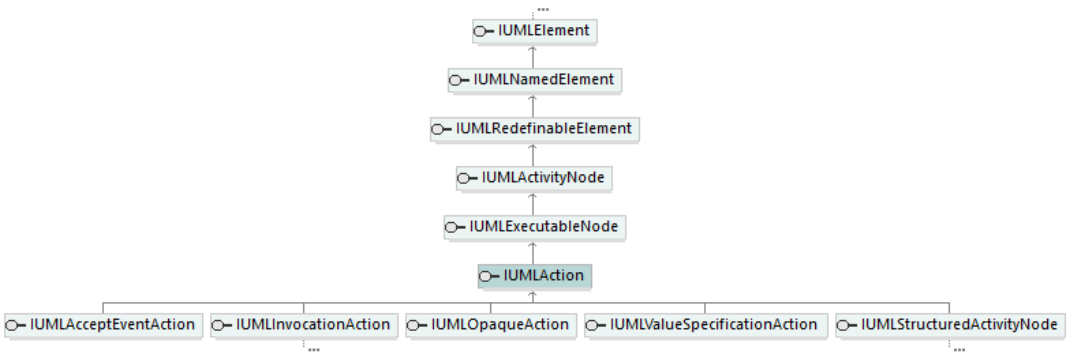
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.3 UModelAPI - IUMLAction

Interface **IUMLAction**

diagram



hierarchy	

Operation **IUMLAction::ActionContext**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLAction::ActionInputPin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActionInputPin**			

Operation **IUMLAction::InputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInputPin.					

Operation **IUMLAction::InsertLocalPostConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLAction::InsertLocalPreConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLAction::IsLocallyReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLAction::LocalPostConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConstraint.					

Operation **IUMLAction::LocalPreConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConstraint.					

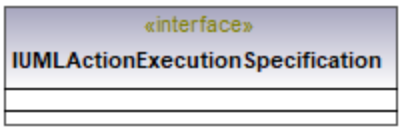
Operation **IUMLAction::OutputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOutputPin.					

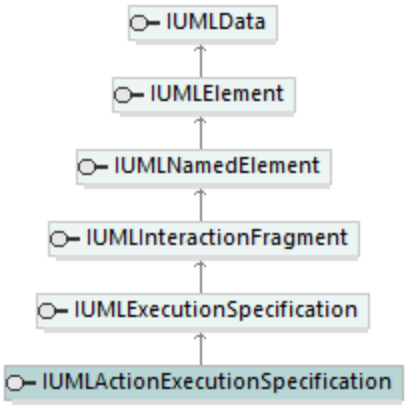
17.5.3.5.4 UModelAPI - IUMLActionExecutionSpecification

Interface **IUMLActionExecutionSpecification**

diagram

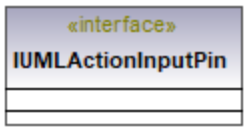
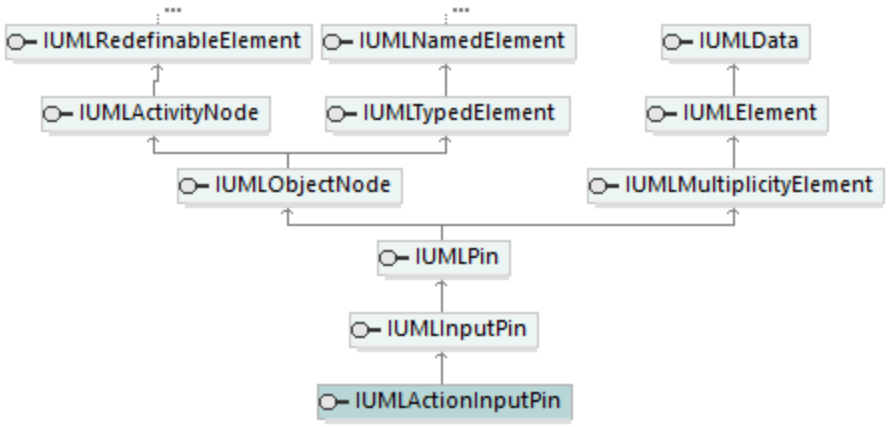


hierarchy



17.5.3.5.5 UModelAPI - IUMLActionInputPin

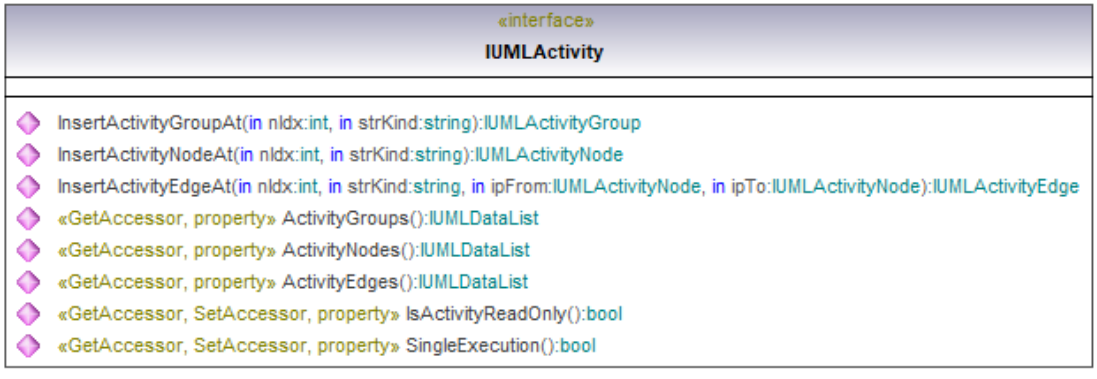
Interface **IUMLActionInputPin**

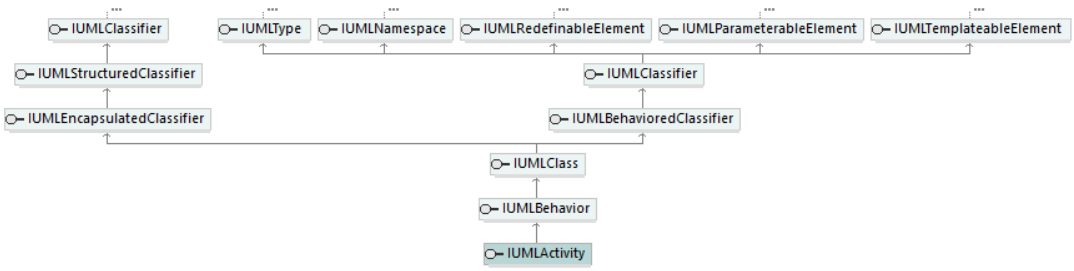
diagram	
hierarchy	
typedElements	Interface **IUMLAction** Interface **IUMLDataAll** Operation **ActionInputPin** Operation **ActionInputPin**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.6 UModelAPI - IUMLActivity

Interface **IUMLActivity**

diagram	

hierarchy	
typedElements	Interface **IUMLActivityEdge** Interface **IUMLActivityGroup** Interface **IUMLDataAll** Operation **Activity** Operation **InActivity** Operation **Activity InActivity**

Operation **IUMLActivity::ActivityEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

Operation **IUMLActivity::ActivityGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityGroup.					

Operation **IUMLActivity::ActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityNode.					

Operation **IUMLActivity::InsertActivityEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLActivityNode**			
	ipTo	**in**	**IUMLActivityNode**			
	return	**return**	**IUMLActivityEdge**			

Operation **IUMLActivity::InsertActivityGroupAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			

	return	**return**	**IUMLActivityGroup**

Operation **IUMLActivity::InsertActivityNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityNode**			

Operation **IUMLActivity::IsActivityReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

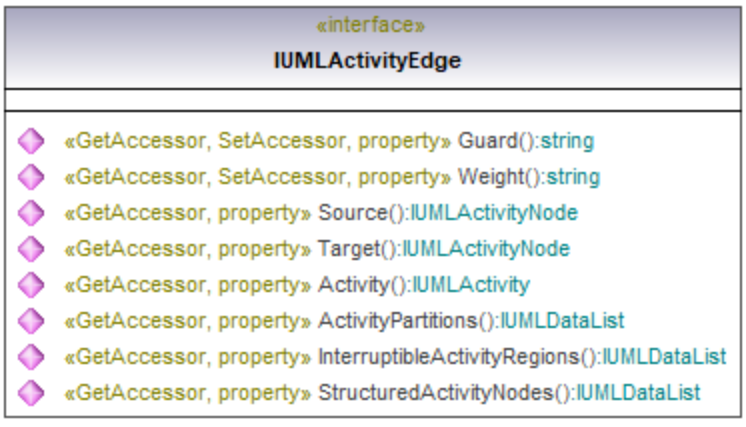
Operation **IUMLActivity::SingleExecution**

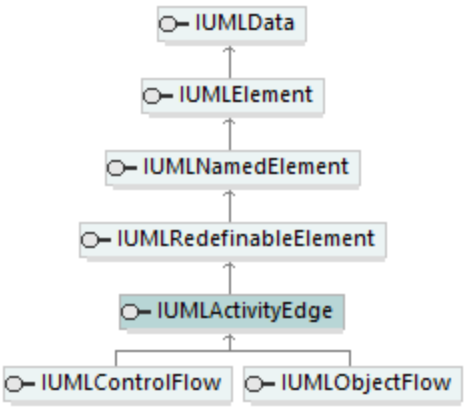
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.3.5.7 UModelAPI - IUMLActivityEdge

Interface **IUMLActivityEdge**

diagram



hierarchy	
typedElements	Interface **IUMLActivity** Interface **IUMLActivityPartition** Interface **IUMLDataAll** Interface **IUMLInterruptibleActivityRegion** Interface **IUMLStructuredActivityNode** Operation **InsertActivityEdgeAt** Operation **InsertEdgeAt** Operation **InsertActivityEdgeAt** **InsertEdgeAt** **InsertInterruptingEdgeAt** Operation **InsertInterruptingEdgeAt** Operation **InsertEdgeAt**

Operation **IUMLActivityEdge::Activity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity**			

Operation **IUMLActivityEdge::ActivityPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityPartition.					

Operation **IUMLActivityEdge::Guard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLActivityEdge::InterruptibleActivityRegions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInterruptibleActivityRegion.					

Operation **IUMLActivityEdge::Source**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityNode**			

Operation **IUMLActivityEdge::StructuredActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLStructuredActivityNode.					

Operation **IUMLActivityEdge::Target**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityNode**			

Operation **IUMLActivityEdge::Weight**

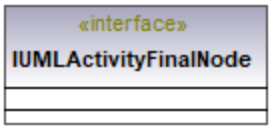
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

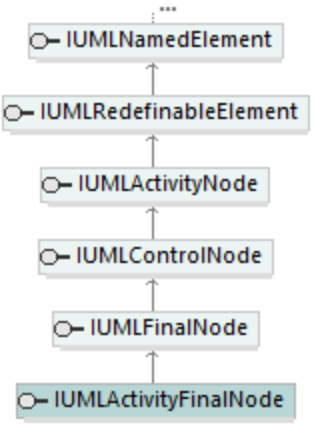
17.5.3.5.8 UModelAPI - IUMLActivityFinalNode

Interface **IUMLActivityFinalNode**

diagram



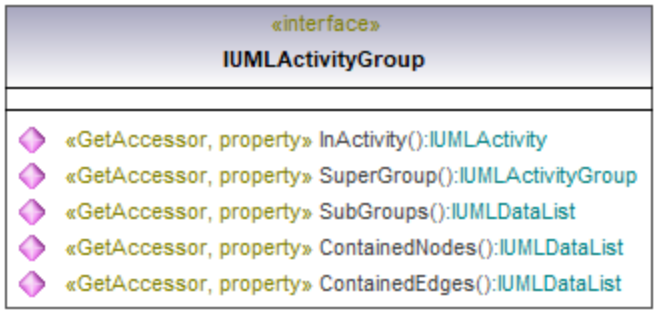
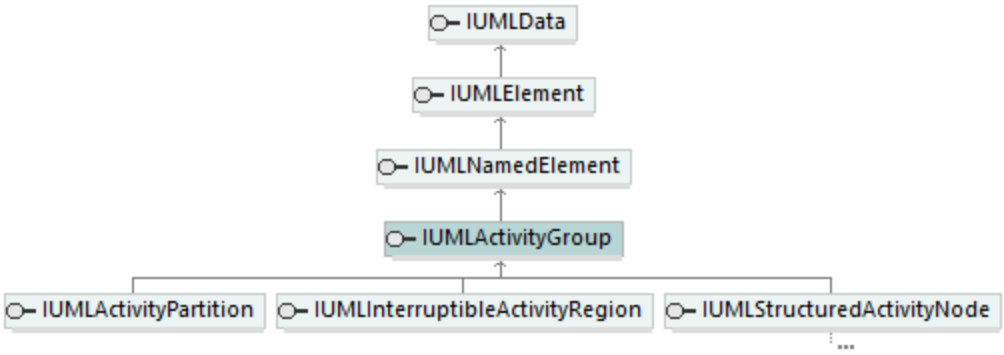
hierarchy



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.9 UModelAPI - IUMLActivityGroup

Interface **IUMLActivityGroup**

diagram	
hierarchy	
typedElements	Interface **IUMLActivity** Interface **IUMLActivityGroup** Interface **IUMLDataAll** Operation **InsertActivityGroupAt** Operation **SuperGroup** Operation **InsertActivityGroupAt SuperGroup**

Operation **IUMLActivityGroup::ContainedEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

Operation **IUMLActivityGroup::ContainedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityNode.					

Operation **IUMLActivityGroup::InActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity**			

Operation **IUMLActivityGroup::SubGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityGroup.					

Operation **IUMLActivityGroup::SuperGroup**

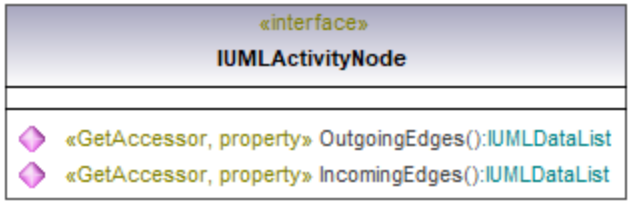
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityGroup**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

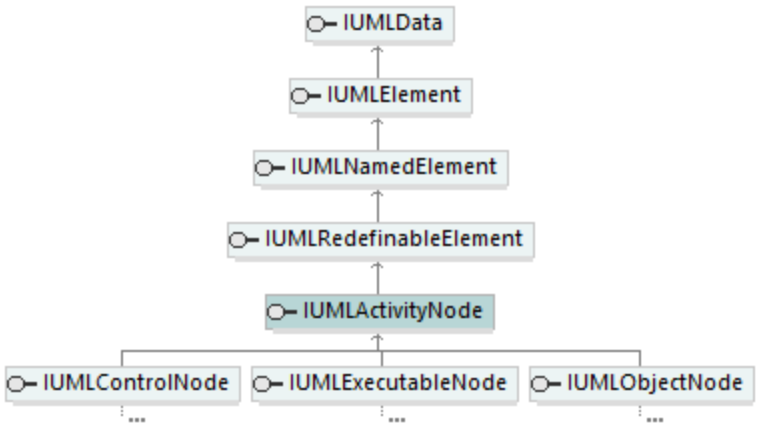
17.5.3.5.10 UModelAPI - IUMLActivityNode

Interface **IUMLActivityNode**

diagram



hierarchy



typedElements		
	Interface **IUMLActivity**	Operation **InsertActivityEdgeAt** **InsertActivityNodeAt**
	Interface **IUMLActivityEdge**	Operation **Source** **Target**

	Interface **IUMLActivityPartition** Interface **IUMLDataAll** Interface **IUMLInterruptibleActivityRegion** Interface **IUMLStructuredActivityNode**	Operation **InsertNodeAt** Operation **InsertActivityEdgeAt** **InsertActivityNodeAt** **InsertNodeAt** **Source** **Target** Operation **InsertNodeAt** Operation **InsertNodeAt**

Operation **IUMLActivityNode::IncomingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

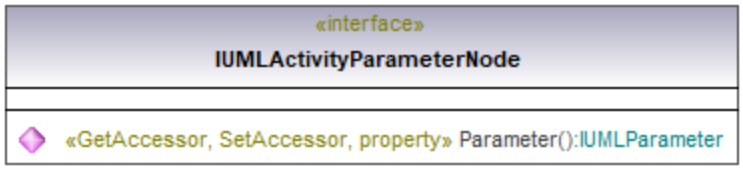
Operation **IUMLActivityNode::OutgoingEdges**

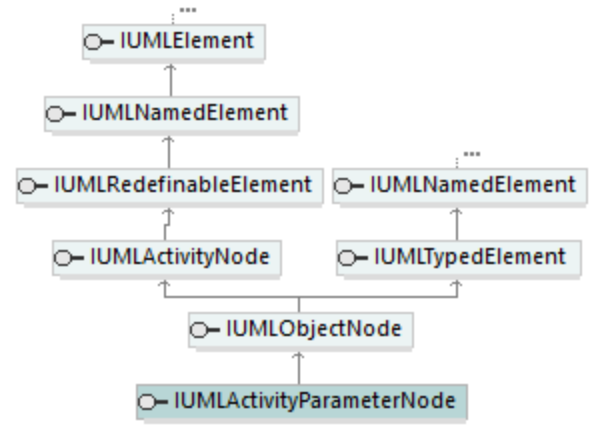
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.11 UModelAPI - IUMLActivityParameterNode

Interface **IUMLActivityParameterNode**

diagram	

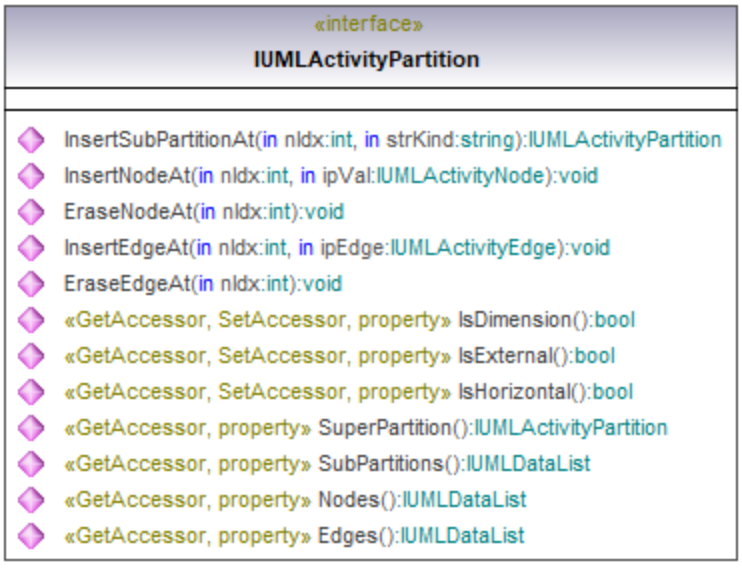
hierarchy	

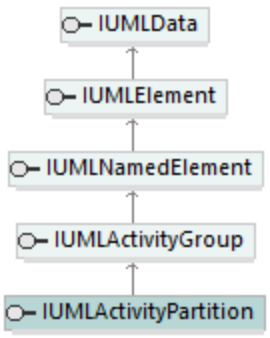
Operation **IUMLActivityParameterNode::Parameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameter**			

17.5.3.5.12 UModelAPI - IUMLActivityPartition

Interface **IUMLActivityPartition**

diagram	

hierarchy	
typedElements	Interface **IUMLActivityPartition** — Operation **InsertSubPartitionAt SuperPartition** Interface **IUMLDataAll** — Operation **InsertSubPartitionAt SuperPartition**

Operation **IUMLActivityPartition::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

Operation **IUMLActivityPartition::EraseEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLActivityPartition::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLActivityPartition::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge**			
	return	**return**	**void**			

Operation **IUMLActivityPartition::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLActivityNode**			
	return	**return**	**void**			

Operation **IUMLActivityPartition::InsertSubPartitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityPartition**			

Operation **IUMLActivityPartition::IsDimension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLActivityPartition::IsExternal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLActivityPartition::IsHorizontal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLActivityPartition::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityNode.					

Operation **IUMLActivityPartition::SubPartitions**

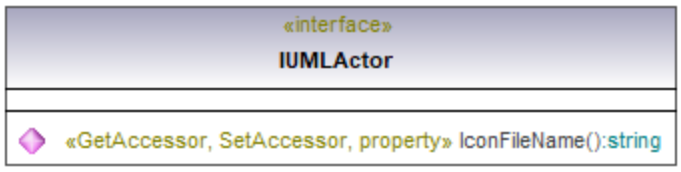
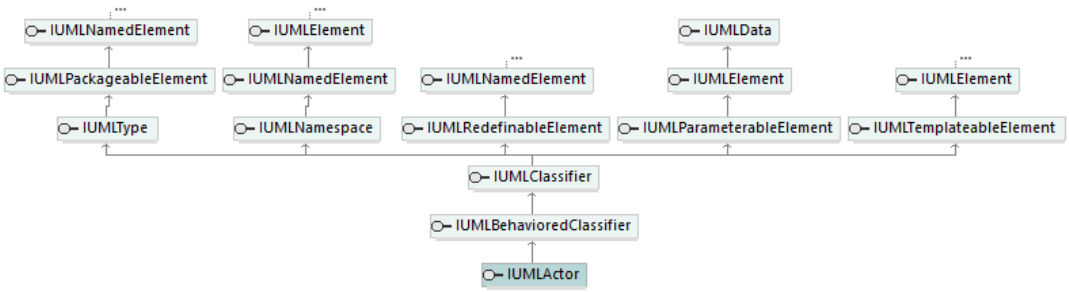
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityPartition.					

Operation **IUMLActivityPartition::SuperPartition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityPartition**			

17.5.3.5.13 UModelAPI - IUMLActor

Interface **IUMLActor**

diagram	
hierarchy	

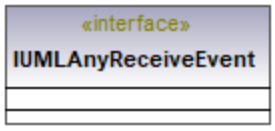
Operation **IUMLActor::IconFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

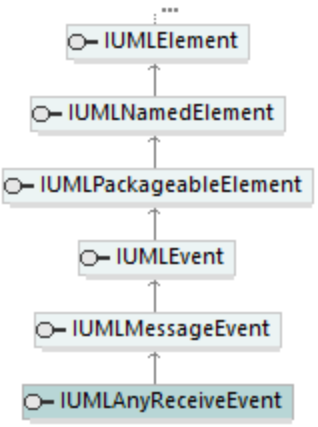
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.14 UModelAPI - IUMLAnyReceiveEvent

Interface **IUMLAnyReceiveEvent**

diagram	

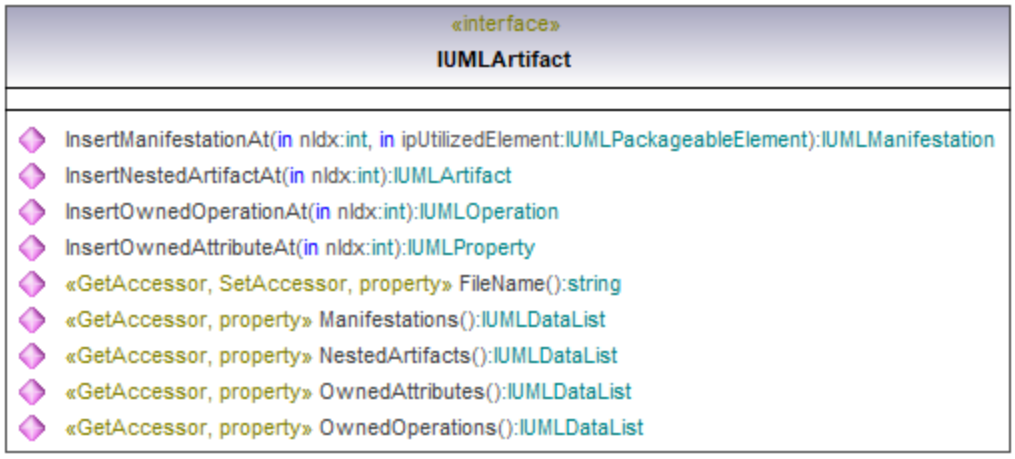
hierarchy



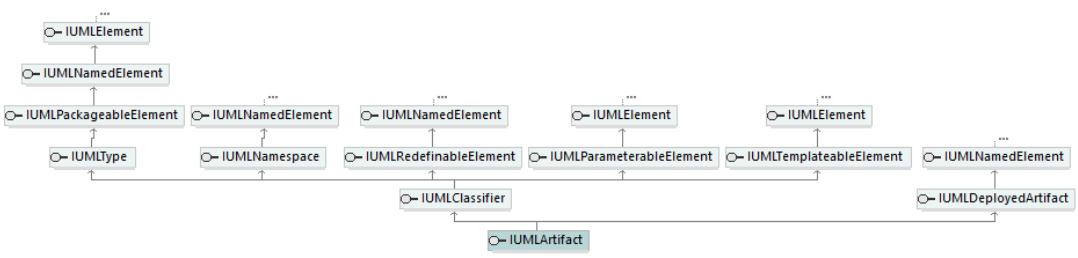
17.5.3.5.15 UModelAPI - IUMLArtifact

Interface **IUMLArtifact**

diagram



hierarchy



typedElements	Interface **IUMLArtifact** Interface **IUMLDataAll**	Operation **InsertNestedArtifactAt** Operation **InsertNestedArtifactAt**

Operation **IUMLArtifact::FileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLArtifact::InsertManifestationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipUtilizedElement	**in**	**IUMLPackageableElement**			
	return	**return**	**IUMLManifestation**			

Operation **IUMLArtifact::InsertNestedArtifactAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLArtifact**			

Operation **IUMLArtifact::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLArtifact::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation**			

Operation **IUMLArtifact::Manifestations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLManifestation.					

Operation **IUMLArtifact::NestedArtifacts**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLArtifact.					

Operation **IUMLArtifact::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

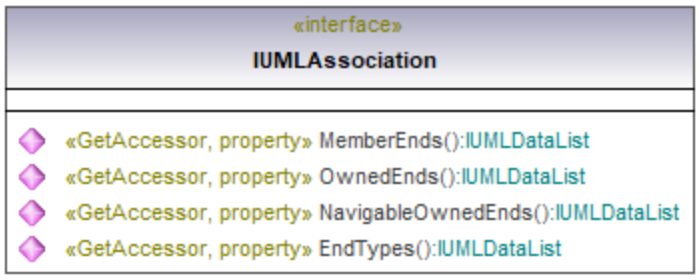
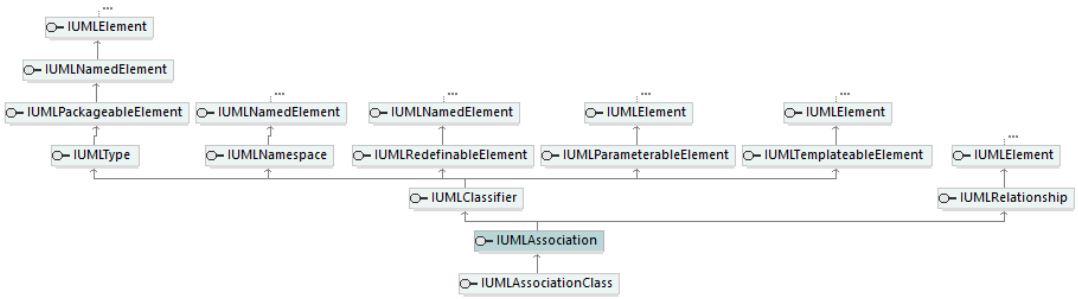
documentation	A list of elements of type IUMLProperty.

Operation **IUMLArtifact::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOperation.					

17.5.3.5.16 UModelAPI - IUMLAssociation

Interface **IUMLAssociation**

diagram	
hierarchy	
typedElements	Interface **IUMLConnector** Interface **IUMLDataAll** Interface **IUMLProperty** Operation **ConnectorType** Operation **Association ConnectorType OwningAssociation** Operation **Association OwningAssociation**	

Operation **IUMLAssociation::EndTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLType.					

Operation **IUMLAssociation::MemberEnds**

parameter	name **return**	direction **return**	type **IUMLDataList**	type modifier	multiplicity	default
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLAssociation::NavigableOwnedEnds**

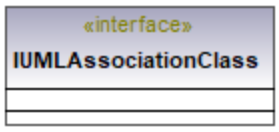
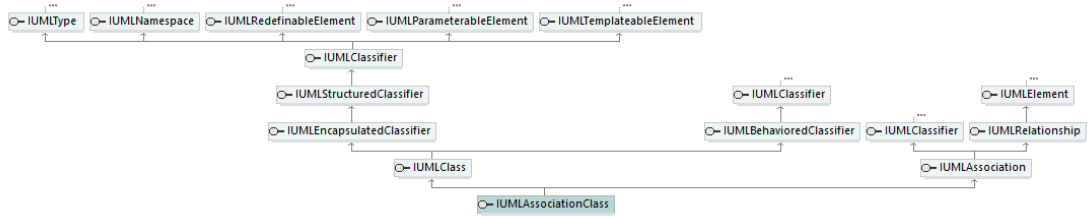
parameter	name **return**	direction **return**	type **IUMLDataList**	type modifier	multiplicity	default
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLAssociation::OwnedEnds**

parameter	name **return**	direction **return**	type **IUMLDataList**	type modifier	multiplicity	default
documentation	A list of elements of type IUMLProperty.					

17.5.3.5.17 UModelAPI - IUMLAssociationClass

Interface **IUMLAssociationClass**

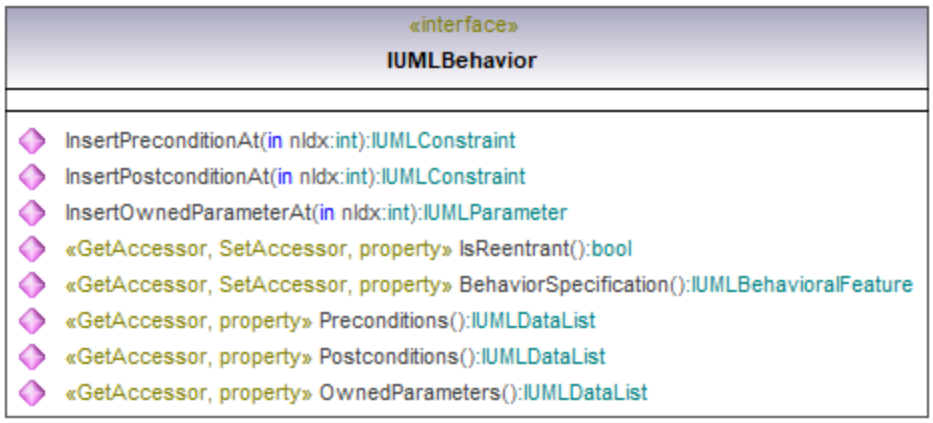
diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

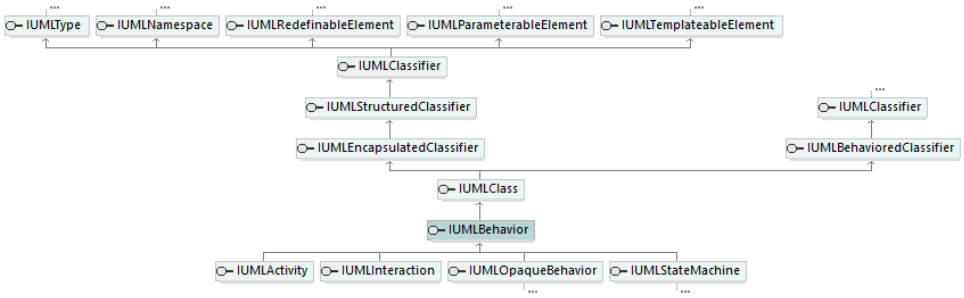
17.5.3.5.18 UModelAPI - IUMLBehavior

Interface **IUMLBehavior**

diagram



hierarchy



typedElements

Interface **IUMLBehavioredClassifier**	Operation **InsertOwnedBehaviorAt**
Interface **IUMLBehaviorExecutionSpecification**	Operation **BehaviorExecution**
Interface **IUMLCallBehaviorAction**	Operation **Behavior**
Interface **IUMLDataAll**	Operation **Behavior BehaviorExecution DecisionInput DoActivity Effect Entry Exit InsertOwnedBehaviorAt Selection SetNewDoActivity SetNewEffect SetNewEntry SetNewExit Transformation**
Interface **IUMLDecisionNode**	Operation **DecisionInput**
Interface **IUMLObjectFlow**	Operation **Transformation**
Interface **IUMLObjectNode**	Operation **Selection**
Interface **IUMLState**	Operation **DoActivity Entry Exit SetNewDoActivity SetNewEntry SetNewExit**
Interface **IUMLTransition**	Operation **Effect SetNewEffect**

Operation **IUMLBehavior::BehaviorSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavioralFeature**			

Operation **IUMLBehavior::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLParameter**			

Operation **IUMLBehavior::InsertPostconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLBehavior::InsertPreconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLBehavior::IsReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLBehavior::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLParameter.					

Operation **IUMLBehavior::Postconditions**

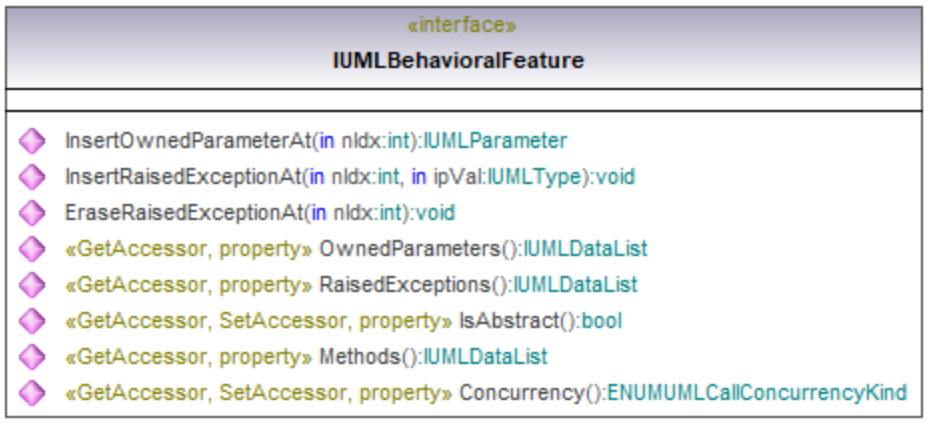
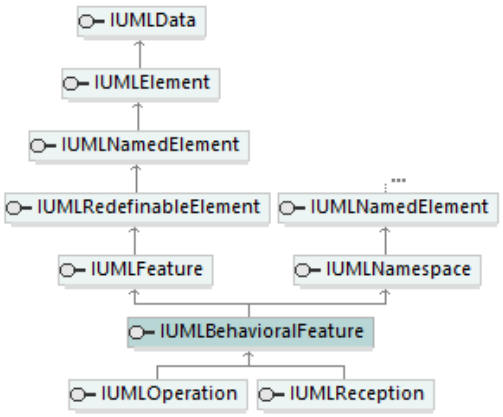
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConstraint.					

Operation **IUMLBehavior::Preconditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConstraint.					

17.5.3.5.19 UModelAPI - IUMLBehavioralFeature

Interface **IUMLBehavioralFeature**

diagram	
hierarchy	
typedElements	Interface **IUMLBehavior** Interface **IUMLDataAll**	Operation **BehaviorSpecification** Operation **BehaviorSpecification**

Operation **IUMLBehavioralFeature::Concurrency**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLCallConcurrencyKind**			

Operation **IUMLBehavioralFeature::EraseRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLBehavioralFeature::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	return	**return**	**IUMLParameter**

Operation **IUMLBehavioralFeature::InsertRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLType**			
	return	**return**	**void**			

Operation **IUMLBehavioralFeature::IsAbstract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLBehavioralFeature::Methods**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLBehavior.					

Operation **IUMLBehavioralFeature::OwnedParameters**

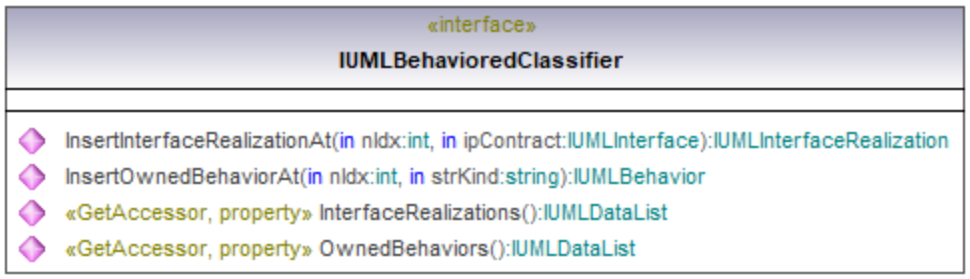
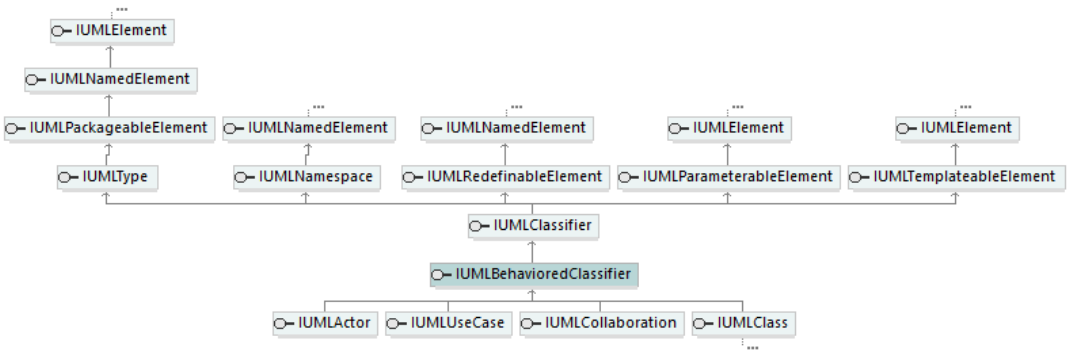
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLParameter.					

Operation **IUMLBehavioralFeature::RaisedExceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLType.					

17.5.3.5.20 UModelAPI - IUMLBehavioredClassifier

Interface **IUMLBehavioredClassifier**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInterfaceRealization** Operation **ImplementingClassifier** Operation **ImplementingClassifier**

Operation **IUMLBehavioredClassifier::InsertInterfaceRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipContract	**in**	**IUMLInterface**			
	return	**return**	**IUMLInterfaceRealization**			

Operation **IUMLBehavioredClassifier::InsertOwnedBehaviorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLBehavioredClassifier::InterfaceRealizations**

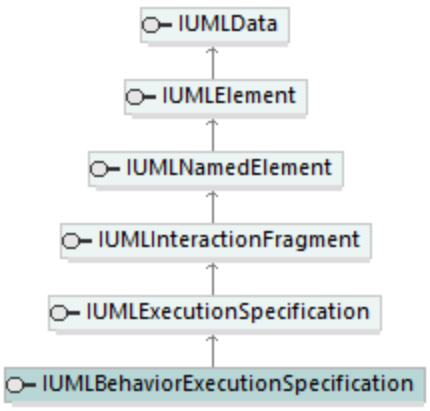
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInterfaceRealizations.					

Operation **IUMLBehavioredClassifier::OwnedBehaviors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLBehavior.					

17.5.3.5.21 UModelAPI - IUMLBehaviorExecutionSpecification

Interface **IUMLBehaviorExecutionSpecification**

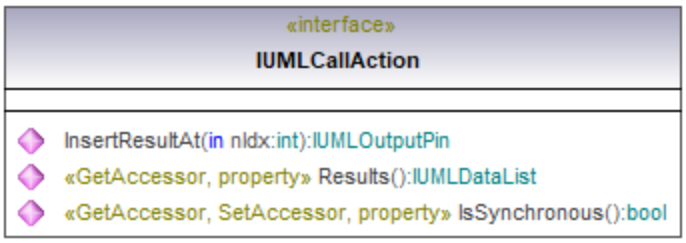
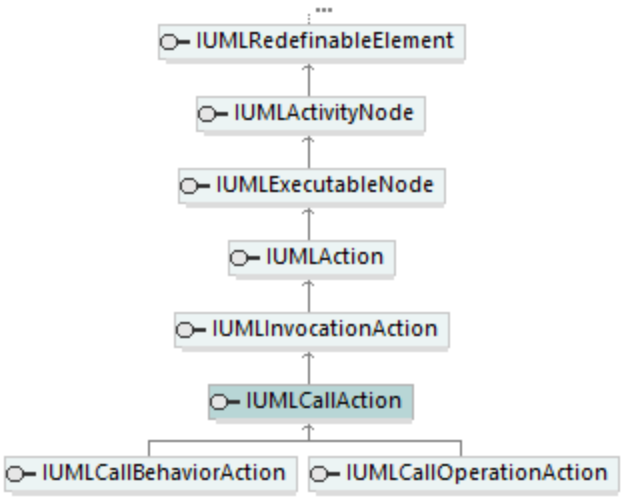
diagram	
hierarchy	

Operation **IUMLBehaviorExecutionSpecification::BehaviorExecution**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

17.5.3.5.22 UModelAPI - IUMLCallAction

Interface **IUMLCallAction**

diagram	
hierarchy	

Operation **IUMLCallAction::InsertResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLCallAction::IsSynchronous**

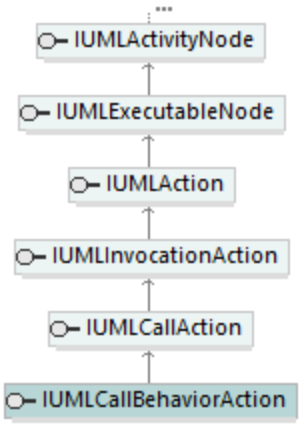
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLCallAction::Results**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOutputPin.					

17.5.3.5.23 UModelAPI - IUMLCallBehaviorAction

Interface **IUMLCallBehaviorAction**

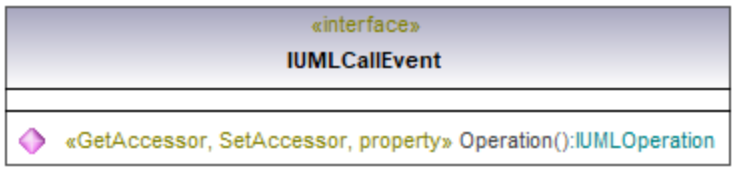
diagram	
hierarchy	

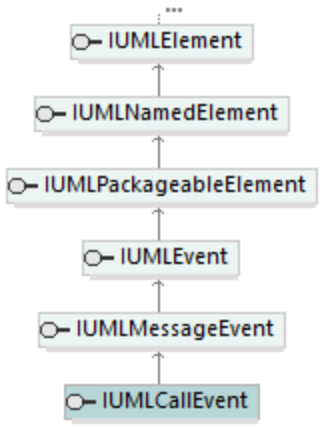
Operation **IUMLCallBehaviorAction::Behavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

17.5.3.5.24 UModelAPI - IUMLCallEvent

Interface **IUMLCallEvent**

diagram	

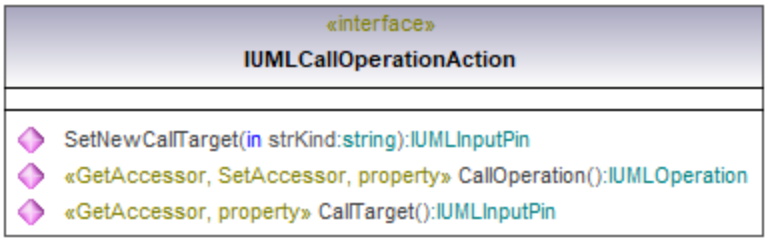
hierarchy	

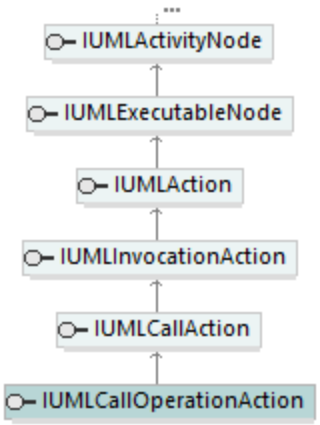
Operation **IUMLCallEvent::Operation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

17.5.3.5.25 UModelAPI - IUMLCallOperationAction

Interface **IUMLCallOperationAction**

diagram	

hierarchy	

Operation **IUMLCallOperationAction::CallOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLCallOperationAction::CallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin**			

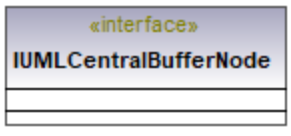
Operation **IUMLCallOperationAction::SetNewCallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin**			

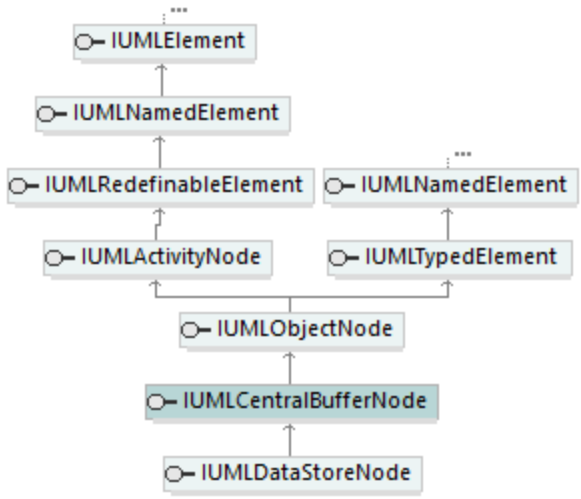
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.26 UModelAPI - IUMLCentralBufferNode

Interface **IUMLCentralBufferNode**

diagram	

hierarchy

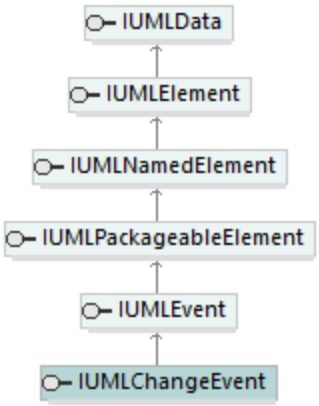


17.5.3.5.27 UModelAPI - IUMLChangeEvent

Interface **IUMLChangeEvent**

diagram

hierarchy



Operation **IUMLChangeEvent::ChangeExpression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

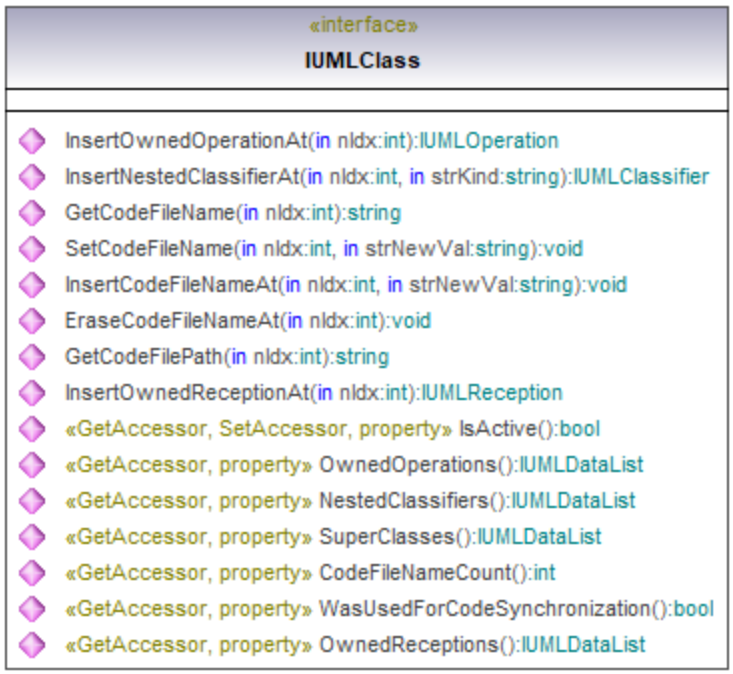
Operation **IUMLChangeEvent::SetNewChangeExpression**

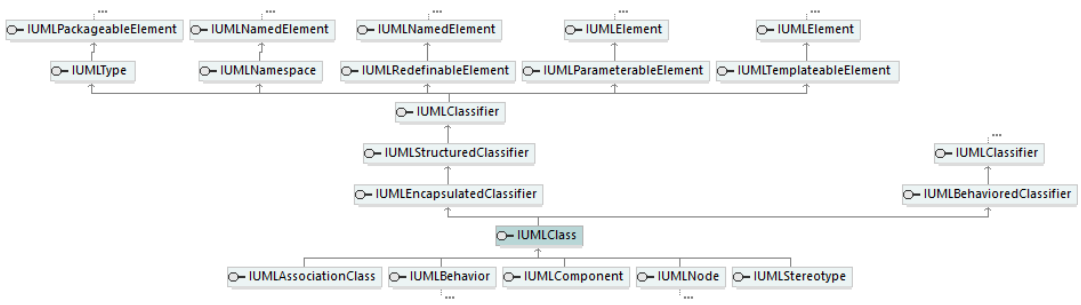
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

17.5.3.5.28 UModelAPI - IUMLClass

Interface **IUMLClass**

diagram



hierarchy	
typedElements	Interface **IUMLClassifier** Interface **IUMLDataAll** Interface **IUMLOperation** Interface **IUMLProperty** Interface **IUMLReception** Operation **Class** Operation **Class** Operation **Class** Operation **Class** Operation **Class**

Operation **IUMLClass::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLClass::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLClass::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IUMLClass::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			
documentation	get the full code file path					

Operation **IUMLClass::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLClass::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLClassifier**			

Operation **IUMLClass::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation**			

Operation **IUMLClass::InsertOwnedReceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLReception**			

Operation **IUMLClass::IsActive**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLClass::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLClass::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOperation.					

Operation **IUMLClass::OwnedReceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLClass::SetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLClass::SuperClasses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClass.					

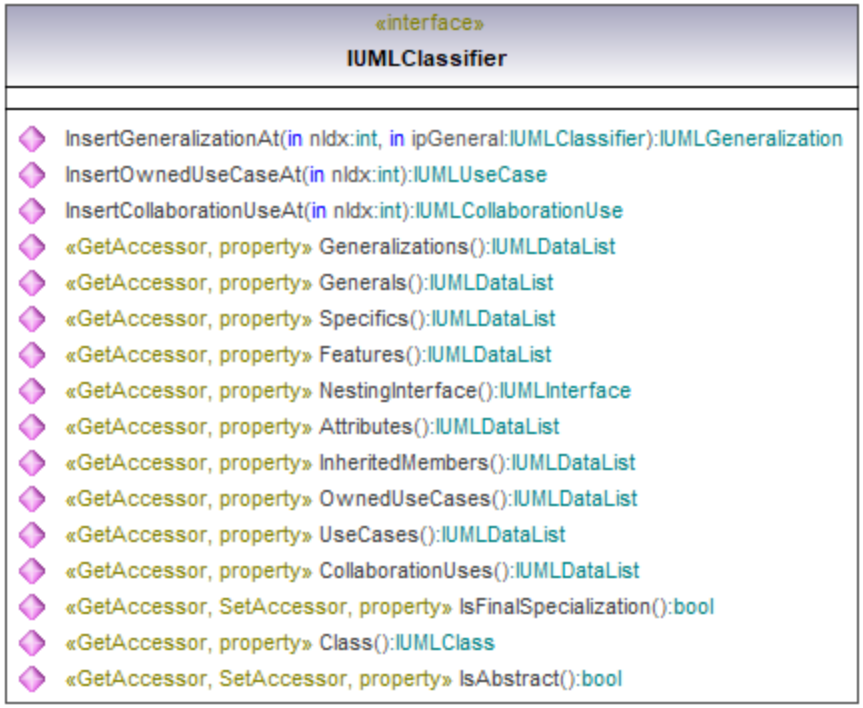
Operation **IUMLClass::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

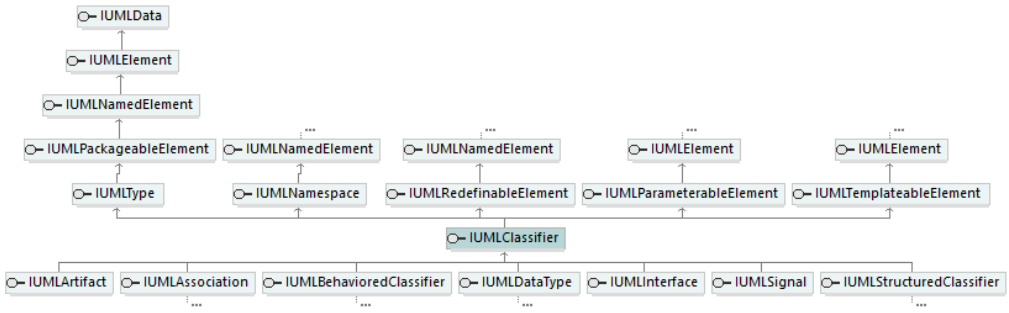
17.5.3.5.29 UModelAPI - IUMLClassifier

Interface **IUMLClassifier**

diagram



hierarchy



typedElements

Interface **IUMLAction**
Interface **IUMLClass**
Interface **IUMLClassifier**
Interface **IUMLClassifierTemplateParameter**
Interface **IUMLComponent**

Operation **ActionContext**
Operation **InsertNestedClassifierAt**
Operation **InsertGeneralizationAt**
Operation **InsertConstrainingClassifierAt**
Operation **InsertRealizationAt**

	Interface **IUMLComponentRealization**	Operation **RealizingClassifier**
	Interface **IUMLDataAll**	Operation **ActionContext** **Classifier** **General** **InsertConstrainingClassifierAt** **InsertConveyedAt** **InsertExceptionTypeAt** **InsertGeneralizationAt** **InsertNestedClassifierAt** **InsertRealizationAt** **InsertSubjectAt** **RealizingClassifier** **Specific**
	Interface **IUMLExceptionHandler**	Operation **InsertExceptionTypeAt**
	Interface **IUMLGeneralization**	Operation **General** **Specific**
	Interface **IUMLInformationFlow**	Operation **InsertConveyedAt**
	Interface **IUMLInstanceSpecification**	Operation **Classifier**
	Interface **IUMLInterface**	Operation **InsertNestedClassifierAt**
	Interface **IUMLProperty**	Operation **Classifier**
	Interface **IUMLUseCase**	Operation **InsertSubjectAt**

Operation **IUMLClassifier::Attributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLClassifier::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass**			

Operation **IUMLClassifier::CollaborationUses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLCollaborationUse.					

Operation **IUMLClassifier::Features**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLFeature.					

Operation **IUMLClassifier::Generalizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLGeneralization.					

Operation **IUMLClassifier::Generals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLClassifier::InheritedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

Operation **IUMLClassifier::InsertCollaborationUseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLCollaborationUse**			

Operation **IUMLClassifier::InsertGeneralizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipGeneral	**in**	**IUMLClassifier**			
	return	**return**	**IUMLGeneralization**			

Operation **IUMLClassifier::InsertOwnedUseCaseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLUseCase**			

Operation **IUMLClassifier::IsAbstract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLClassifier::IsFinalSpecialization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLClassifier::NestingInterface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLClassifier::OwnedUseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLUseCase.					

Operation **IUMLClassifier::Specifics**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

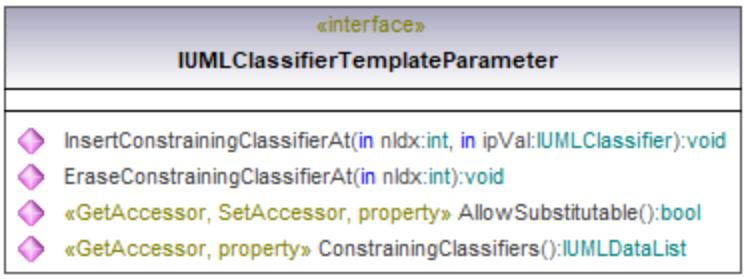
Operation **IUMLClassifier::UseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLUseCase.					

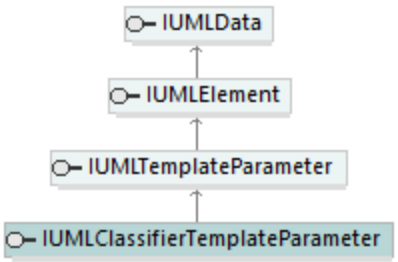
17.5.3.5.30 UModelAPI - IUMLClassifierTemplateParameter

Interface **IUMLClassifierTemplateParameter**

diagram



hierarchy



typedElements

Interface **IUMLDataAll**

Interface **IUMLTemplateSignature**

Operation **InsertOwnedTemplateParameterAt**

Operation **InsertOwnedTemplateParameterAt**

Operation **IUMLClassifierTemplateParameter::AllowSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLClassifierTemplateParameter::ConstrainingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLClassifierTemplateParameter::EraseConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

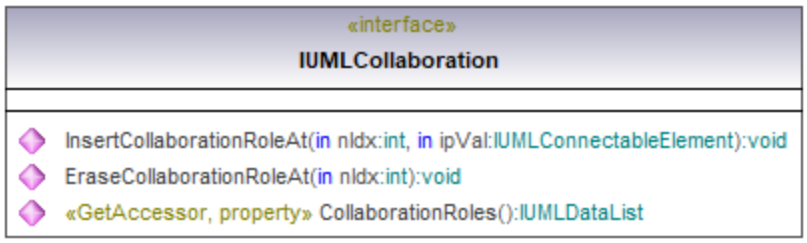
Operation **IUMLClassifierTemplateParameter::InsertConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

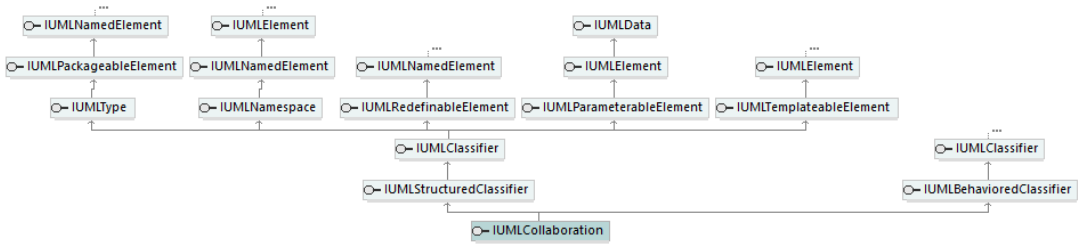
17.5.3.5.31 UModelAPI - IUMLCollaboration

Interface **IUMLCollaboration**

diagram



hierarchy



typedElements	Interface **IUMLCollaborationUse** Interface **IUMLDataAll**	Operation **CollaborationType** Operation **CollaborationType**

Operation **IUMLCollaboration::CollaborationRoles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLCollaboration.					

Operation **IUMLCollaboration::EraseCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLCollaboration::InsertCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnectableElement**			
	return	**return**	**void**			

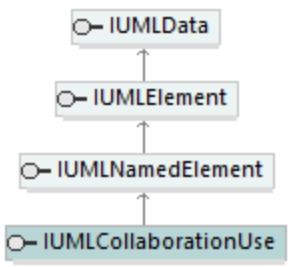
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.32 UModelAPI - IUMLCollaborationUse

Interface **IUMLCollaborationUse**

diagram

hierarchy



typedElements	Interface **IUMLClassifier** Interface **IUMLDataAll**	Operation **InsertCollaborationUseAt** Operation **InsertCollaborationUseAt**

Operation **IUMLCollaborationUse::CollaborationType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLCollaboration**			

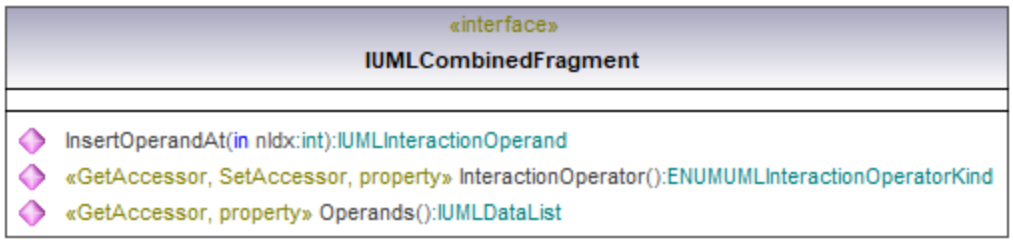
Operation **IUMLCollaborationUse::RoleBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLDependency.					

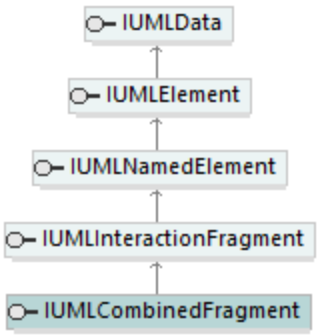
17.5.3.5.33 UModelAPI - IUMLCombinedFragment

Interface **IUMLCombinedFragment**

diagram



hierarchy



Operation **IUMLCombinedFragment::InsertOperandAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	return	**return**	**IUMLInteractionOperand**

Operation **IUMLCombinedFragment::InteractionOperator**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLInteractionOperatorKind**			

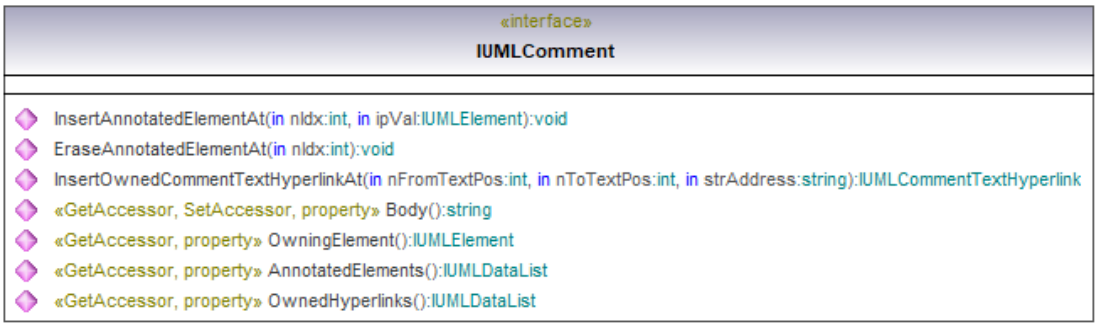
Operation **IUMLCombinedFragment::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInteractionOperand.					

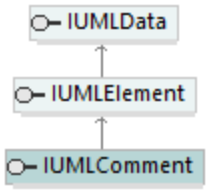
17.5.3.5.34 UModelAPI - IUMLComment

Interface **IUMLComment**

diagram



hierarchy



typedElements

Interface **IUMLDataAll**

Interface **IUMLElement**

Operation **InsertOwnedCommentAtOwnedDocComment**

Operation **InsertOwnedCommentAtOwnedDocComment**

Operation **IUMLComment::AnnotatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

Operation **IUMLComment::Body**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLComment::EraseAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLComment::InsertAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLElement**			
	return	**return**	**void**			

Operation **IUMLComment::InsertOwnedCommentTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLCommentTextHyperlink**			

Operation **IUMLComment::OwnedHyperlinks**

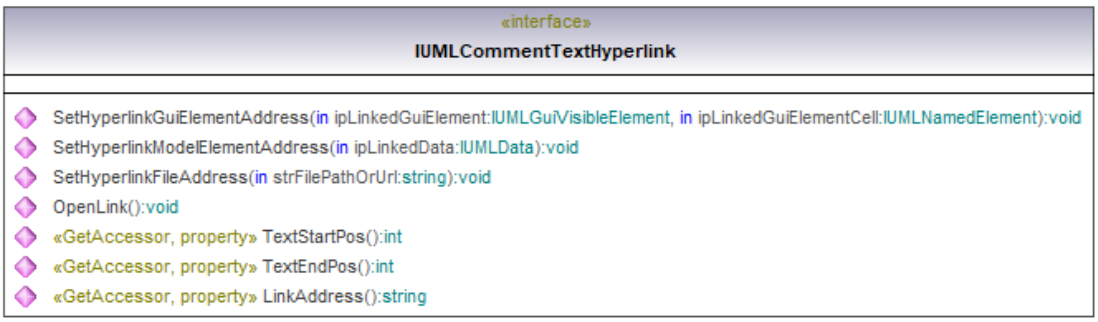
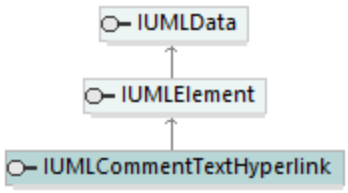
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLComment::OwningElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

17.5.3.5.35 UModelAPI - IUMLCommentTextHyperlink

Interface **IUMLCommentTextHyperlink**

diagram	
hierarchy	
typedElements	Interface **IUMLComment** Interface **IUMLDataAll** Operation **InsertOwnedCommentTextHyperlinkAt** Operation **InsertOwnedCommentTextHyperlinkAt**

Operation **IUMLCommentTextHyperlink::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLCommentTextHyperlink::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strFilePathOrUrl	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::SetHyperlinkGuiElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedGuiElement	**in**	**IUMLGuiVisibleElement**			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	**in**	**IUMLData**			
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::TextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

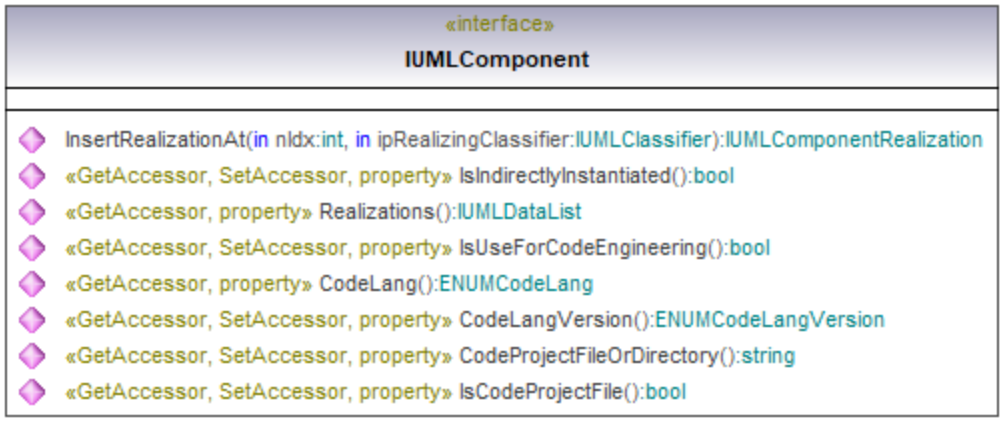
Operation **IUMLCommentTextHyperlink::TextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

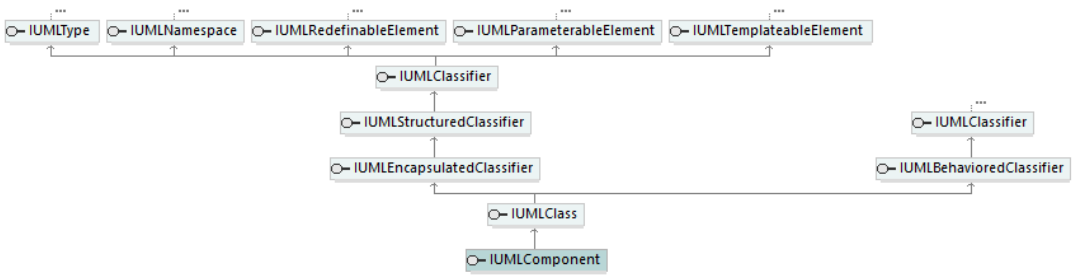
17.5.3.5.36 UModelAPI - IUMLComponent

Interface **IUMLComponent**

diagram



hierarchy



typedElements

Interface **IUMLComponentRealization** Operation **Abstraction**

Interface **IUMLDataAll** Operation **Abstraction**

Operation **IUMLComponent::CodeLang**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLang**			

Operation **IUMLComponent::CodeLangVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion**			

Operation **IUMLComponent::CodeProjectFileOrDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLComponent::InsertRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipRealizingClassifier	**in**	**IUMLClassifier**			
	return	**return**	**IUMLComponentRealization**			

Operation **IUMLComponent::IsCodeProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLComponent::IsIndirectlyInstantiated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLComponent::IsUseForCodeEngineering**

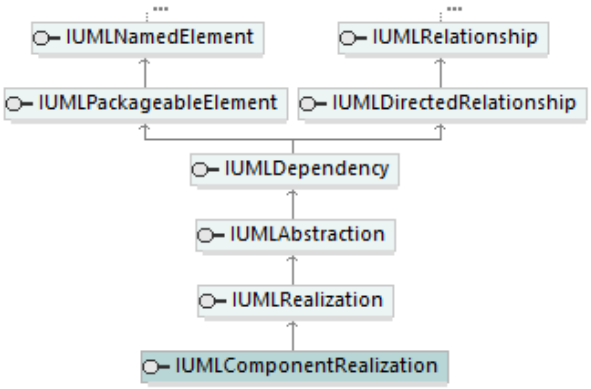
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLComponent::Realizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLComponentRealization.					

17.5.3.5.37 UModelAPI - IUMLComponentRealization

Interface **IUMLComponentRealization**

diagram	
hierarchy	
typedElements	Interface **IUMLComponent** Interface **IUMLDataAll** Operation **InsertRealizationAt** Operation **InsertRealizationAt**

Operation **IUMLComponentRealization::Abstraction**

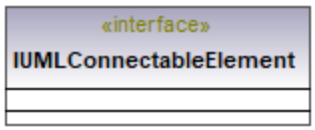
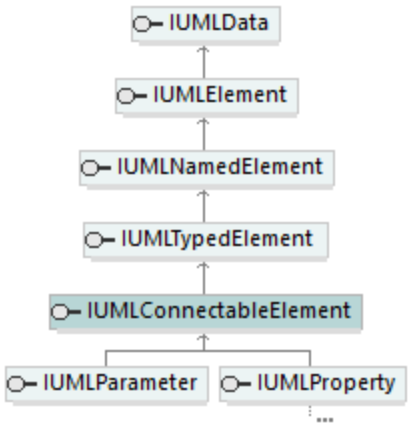
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComponent**			

Operation **IUMLComponentRealization::RealizingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

17.5.3.5.38 UModelAPI - IUMLConnectableElement

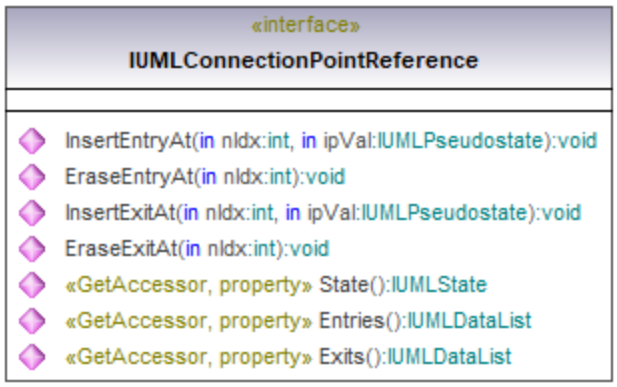
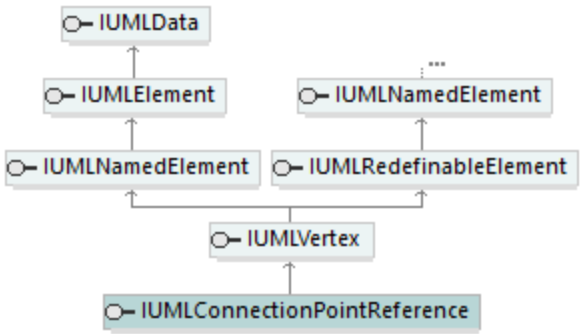
Interface **IUMLConnectableElement**

diagram	
hierarchy	
typedElements	Interface **IUMLCollaboration** Interface **IUMLConnectorEnd** Interface **IUMLDataAll** Interface **IUMLLifeline** Interface **IUMLStructuredClassifier** Operation **InsertCollaborationRoleAt** Operation **Role** Operation **InsertCollaborationRoleAt InsertOwnedConnectorAt Represents Role** Operation **Represents** Operation **InsertOwnedConnectorAt**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.39 UModelAPI - IUMLConnectionPointReference

Interface **IUMLConnectionPointReference**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLState** Operation **InsertConnectionAt** Operation **InsertConnectionAt**

Operation **IUMLConnectionPointReference::Entries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPseudostate.					

Operation **IUMLConnectionPointReference::EraseEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLConnectionPointReference::EraseExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLConnectionPointReference::Exits**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPseudostate.					

Operation **IUMLConnectionPointReference::InsertEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate**			
	return	**return**	**void**			

Operation **IUMLConnectionPointReference::InsertExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate**			
	return	**return**	**void**			

Operation **IUMLConnectionPointReference::State**

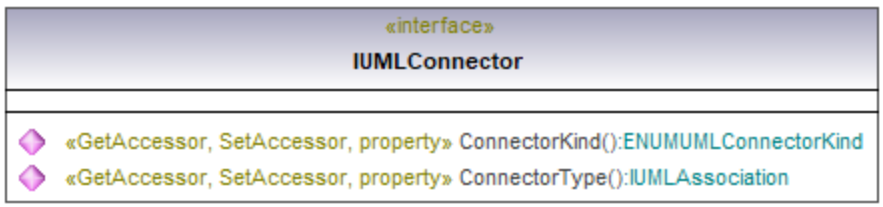
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

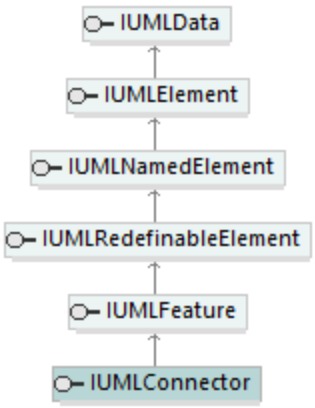
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.40 UModelAPI - IUMLConnector

Interface **IUMLConnector**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInformationFlow** Interface **IUMLStructuredClassifier** Operation **InsertOwnedConnectorAt** **InsertRealizingConnectorAt** Operation **InsertRealizingConnectorAt** Operation **InsertOwnedConnectorAt**

Operation **IUMLConnector::ConnectorKind**

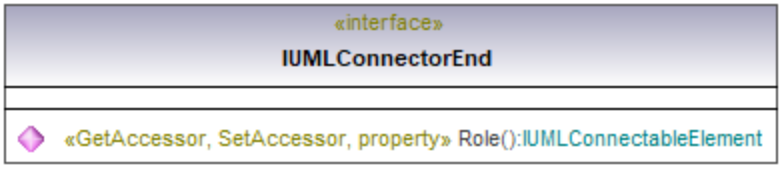
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLConnectorKind**			
documentation	**Deprecated:** Since UML2.3 (UModel2010r2) 'ConnectorKind' is derived and cannot be set anymore.					

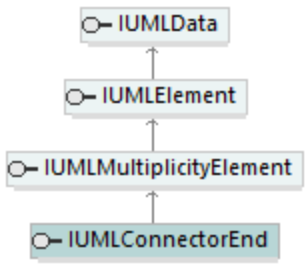
Operation **IUMLConnector::ConnectorType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

17.5.3.5.41 UModelAPI - IUMLConnectorEnd

Interface **IUMLConnectorEnd**

diagram	

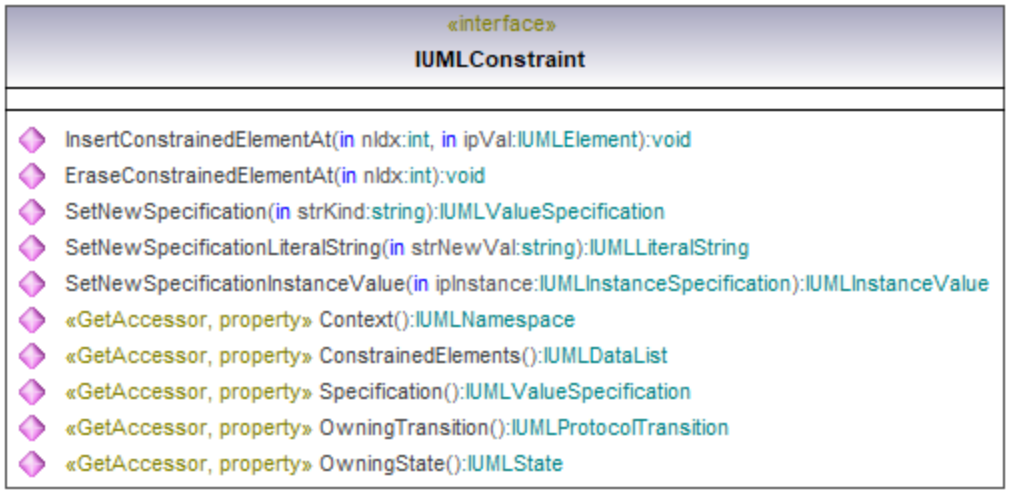
hierarchy	

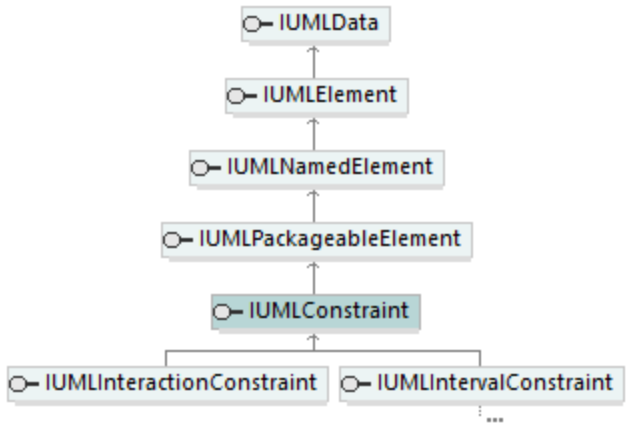
Operation **IUMLConnectorEnd::Role**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConnectableElement**			

17.5.3.5.42 UModelAPI - IUMLConstraint

Interface **IUMLConstraint**

diagram	

hierarchy	
typedElements	Interface **IUMLAction** Interface **IUMLBehavior** Interface **IUMLDataAll** Interface **IUMLNamespace** Interface **IUMLProtocolTransition** Interface **IUMLState** Interface **IUMLStateInvariant** Interface **IUMLTransition** Interface **IUMLValueSpecification** Operation **InsertLocalPostConditionAt** **InsertLocalPreConditionAt** Operation **InsertPostconditionAt** **InsertPreconditionAt** Operation **InsertLocalPostConditionAt** **InsertLocalPreConditionAt** **InsertOwnedRuleAt** **InsertPostconditionAt** **InsertPreconditionAt** **Invariant** **OwningConstraint** **PostCondition** **PreCondition** **SetNewInvariant** **SetNewPostCondition** **SetNewPreCondition** **SetNewStateInvariant** **SetNewTransitionGuard** **StateInvariant** **TransitionGuard** Operation **InsertOwnedRuleAt** Operation **PostCondition** **PreCondition** **SetNewPostCondition** **SetNewPreCondition** Operation **SetNewStateInvariant** **StateInvariant** Operation **Invariant** **SetNewInvariant** Operation **SetNewTransitionGuard** **TransitionGuard** Operation **OwningConstraint**

Operation **IUMLConstraint::ConstrainedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

Operation **IUMLConstraint::Context**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLConstraint::EraseConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLConstraint::InsertConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLElement**			
	return	**return**	**void**			

Operation **IUMLConstraint::OwningState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

Operation **IUMLConstraint::OwningTransition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolTransition**			

Operation **IUMLConstraint::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLConstraint::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLConstraint::SetNewSpecificationLiteralString**

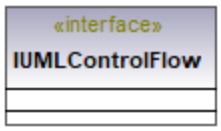
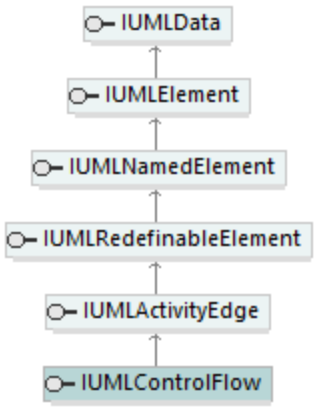
parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

Operation **IUMLConstraint::Specification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

17.5.3.5.43 UModelAPI - IUMLControlFlow

Interface **IUMLControlFlow**

diagram	
hierarchy	

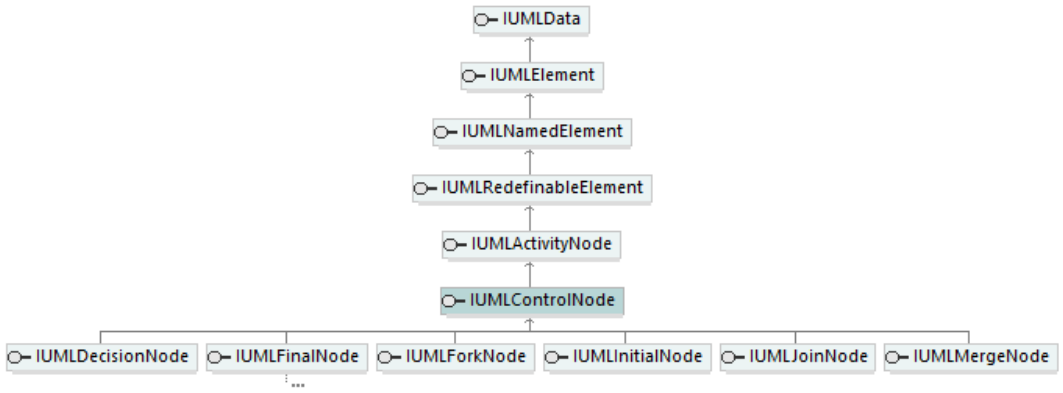
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.44 UModelAPI - IUMLControlNode

Interface **IUMLControlNode**

diagram	

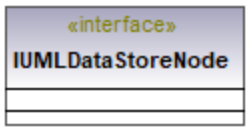
hierarchy



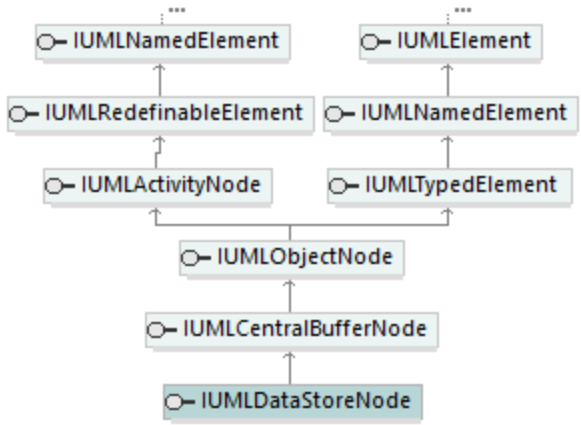
17.5.3.5.45 UModelAPI - IUMLDataStoreNode

Interface **IUMLDataStoreNode**

diagram

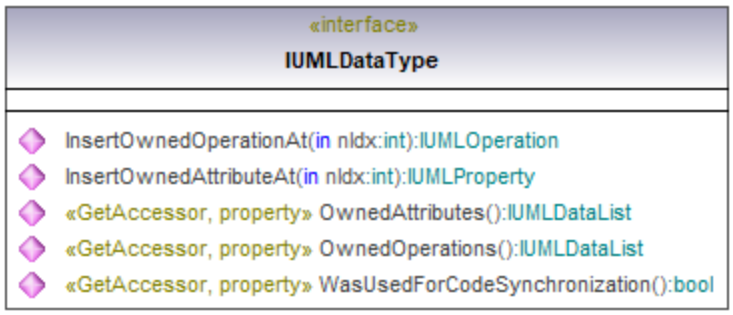
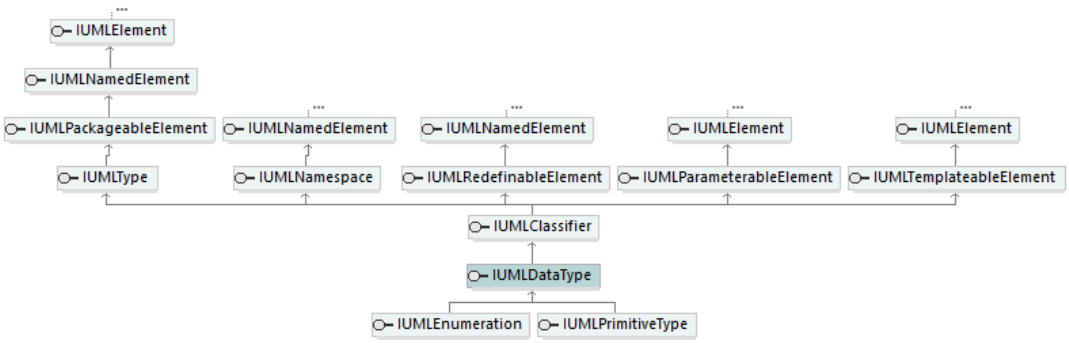


hierarchy



17.5.3.5.46 UModelAPI - IUMLDataType

Interface **IUMLDataType**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLOperation** Interface **IUMLProperty** Operation **Datatype** Operation **Datatype** Operation **Datatype**

Operation **IUMLDataType::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLDataType::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation**			

Operation **IUMLDataType::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLDataType::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList**
documentation	A list of elements of type IUMLOperation.		

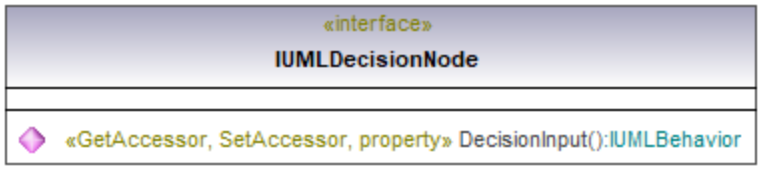
Operation **IUMLDataType::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

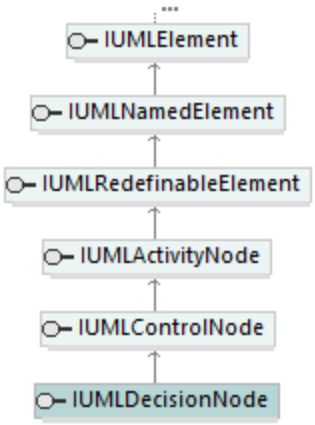
17.5.3.5.47 UModelAPI - IUMLDecisionNode

Interface **IUMLDecisionNode**

diagram



hierarchy

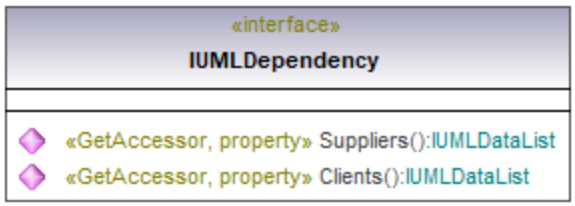
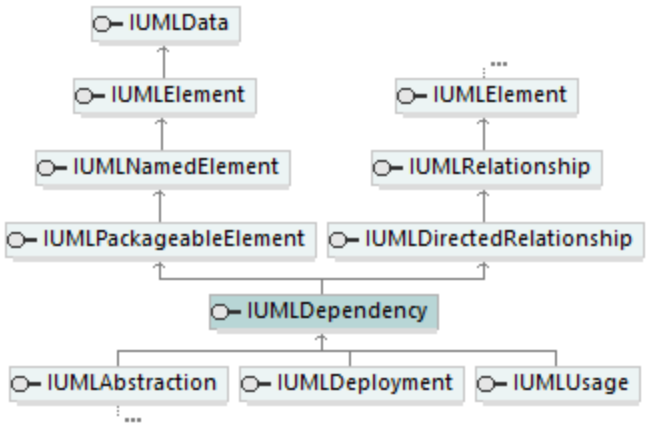


Operation **IUMLDecisionNode::DecisionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

17.5.3.5.48 UModelAPI - IUMLDependency

Interface **IUMLDependency**

diagram	
hierarchy	

Operation **IUMLDependency::Clients**

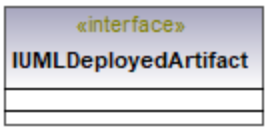
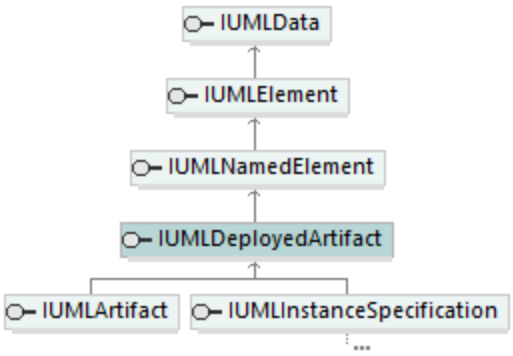
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

Operation **IUMLDependency::Suppliers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

17.5.3.5.49 UModelAPI - IUMLDeployedArtifact

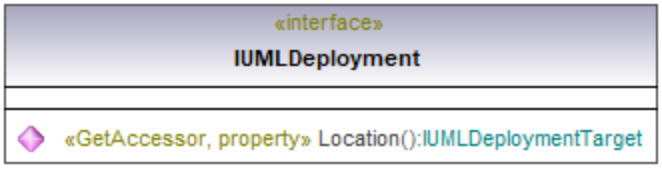
Interface **IUMLDeployedArtifact**

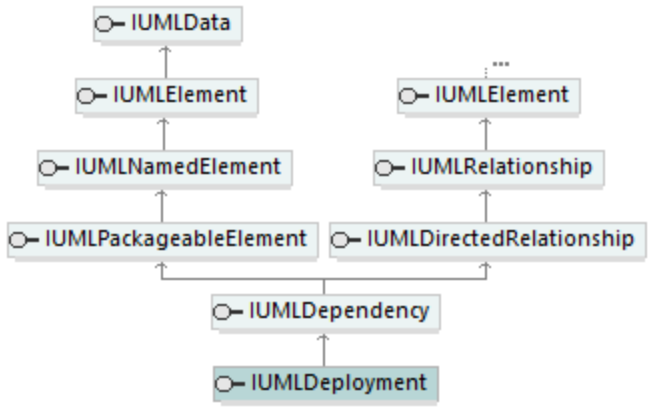
diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLDeploymentTarget** Operation **InsertDeploymentAt** Operation **InsertDeploymentAt**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.50 UModelAPI - IUMLDeployment

Interface **IUMLDeployment**

diagram	

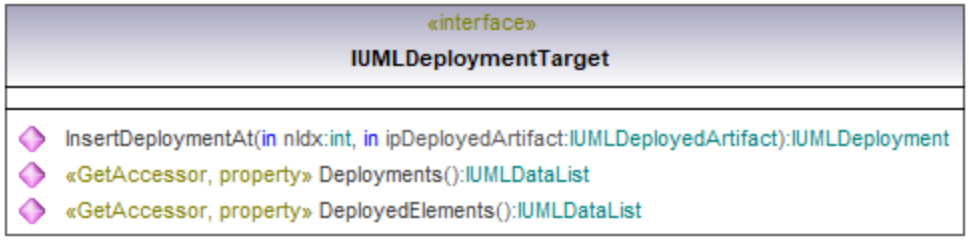
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLDeploymentTarget** Operation **InsertDeploymentAt** Operation **InsertDeploymentAt**

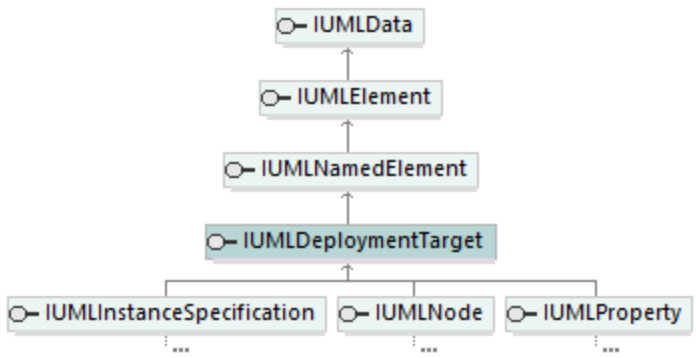
Operation **IUMLDeployment::Location**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDeployment Target**			

17.5.3.5.51 UModelAPI - IUMLDeploymentTarget

Interface **IUMLDeploymentTarget**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLDeployment** Operation **Location** Operation **Location**

Operation **IUMLDeploymentTarget::DeployedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackageableElement.					

Operation **IUMLDeploymentTarget::Deployments**

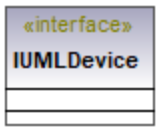
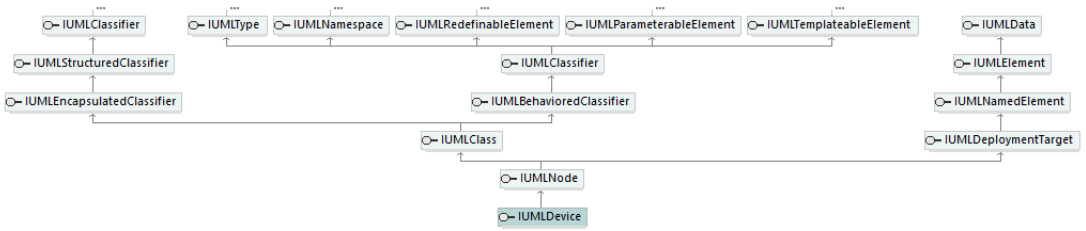
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLDeployment.					

Operation **IUMLDeploymentTarget::InsertDeploymentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDeployedArtifact	**in**	**IUMLDeployedArtifact**			
	return	**return**	**IUMLDeployment**			

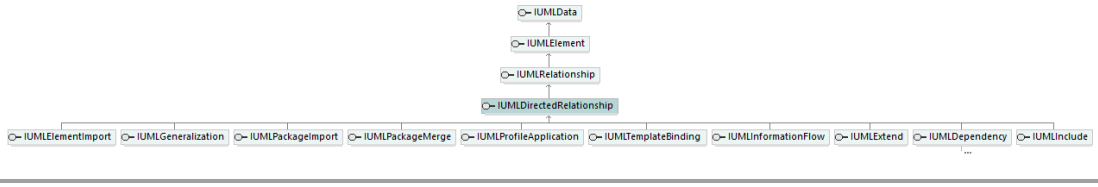
17.5.3.5.52 UModelAPI - IUMLDevice

Interface **IUMLDevice**

diagram	
hierarchy	

17.5.3.5.53 UModelAPI - IUMLDirectedRelationship

Interface **IUMLDirectedRelationship**

diagram	
hierarchy	

Operation **IUMLDirectedRelationship::Sources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

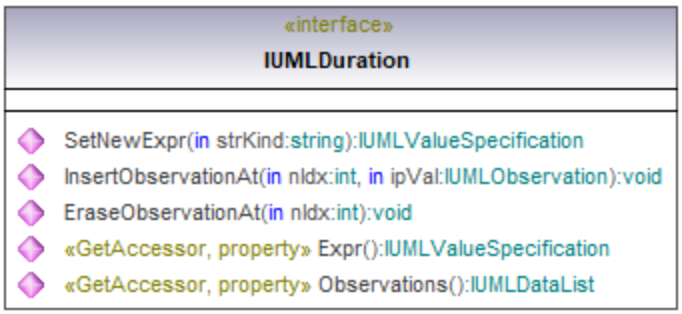
Operation **IUMLDirectedRelationship::Targets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

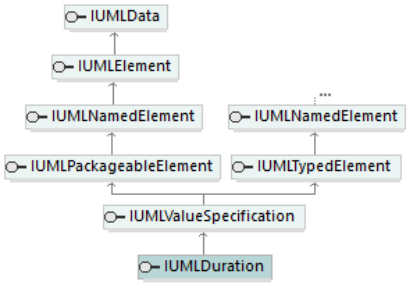
17.5.3.5.54 UModelAPI - IUMLDuration

Interface **IUMLDuration**

diagram



hierarchy



Operation **IUMLDuration::EraseObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDuration::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDuration::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	ipVal	**in**	**IUMLObservation**
	return	**return**	**void**

Operation **IUMLDuration::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLObservation.					

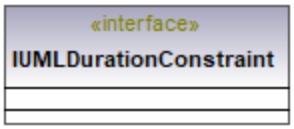
Operation **IUMLDuration::SetNewExpr**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

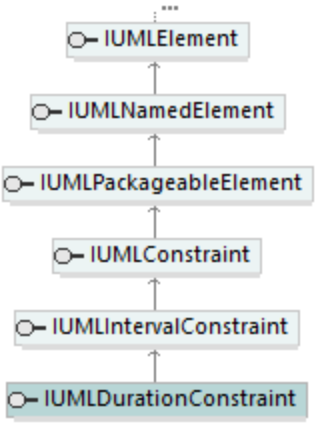
17.5.3.5.55 UModelAPI - IUMLDurationConstraint

Interface **IUMLDurationConstraint**

diagram

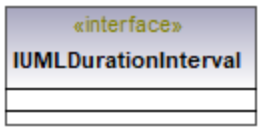
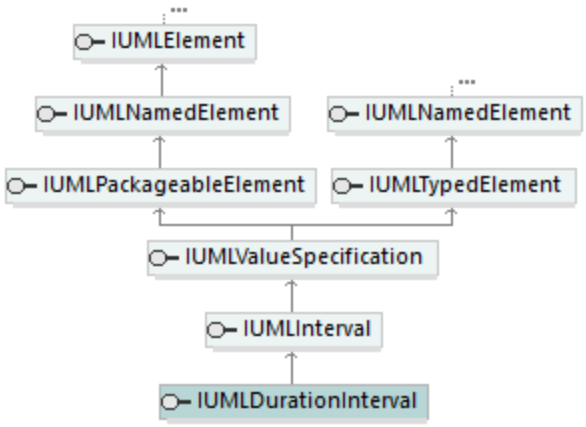


hierarchy



17.5.3.5.56 UModelAPI - IUMLDurationInterval

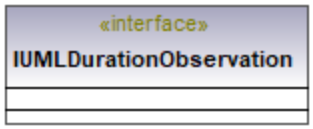
Interface **IUMLDurationInterval**

diagram	
hierarchy	

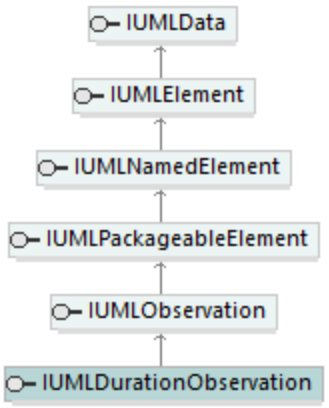
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.57 UModelAPI - IUMLDurationObservation

Interface **IUMLDurationObservation**

diagram	

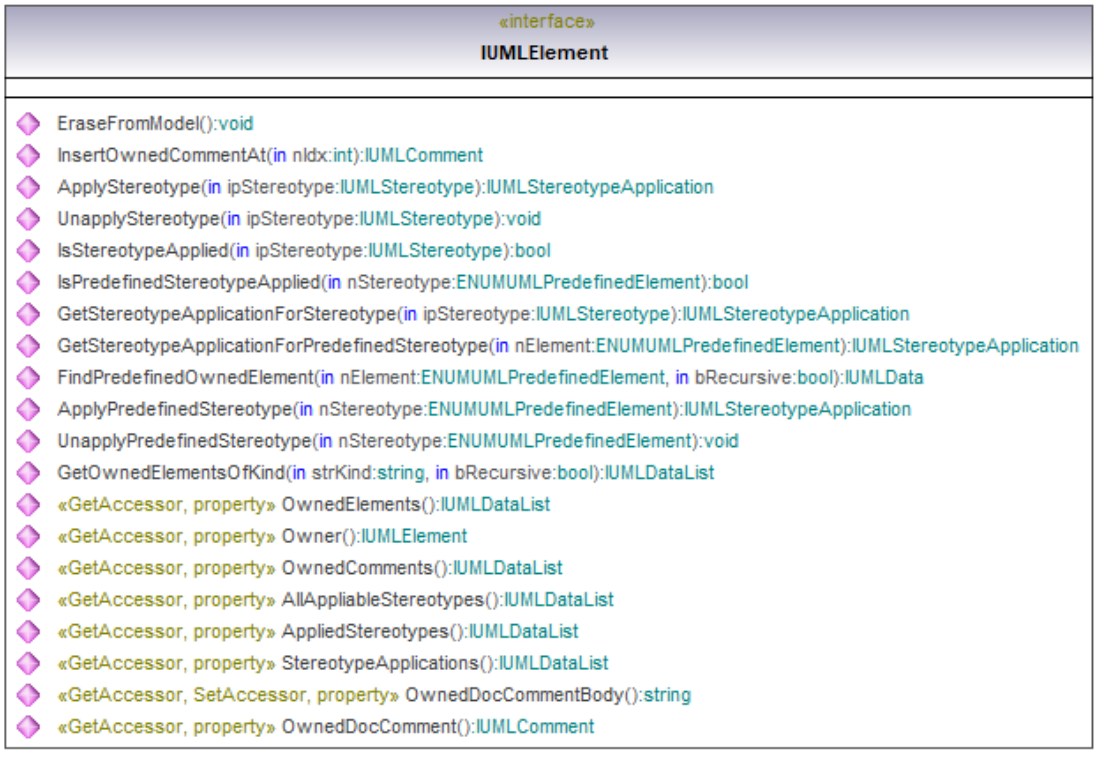
hierarchy

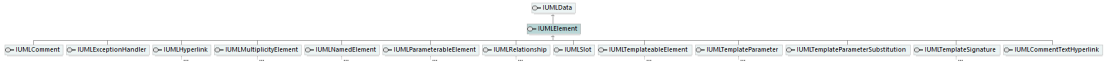


17.5.3.5.58 UModelAPI - IUMLElement

Interface **IUMLElement**

diagram



hierarchy	
typedElements	Interface **IGenerateSequenceDiagramDlg** Interface **IUMLComment** Interface **IUMLConstraint** Interface **IUMLDataAll** Interface **IUMLElement** Interface **IUMLGuiDiagram** Interface **IUMLGuiLink** Interface **IUMLGuiNodeLink** Interface **IUMLGuiTextLabel** Interface **IUMLPackage** Interface **IUMLStereotypeApplication** Operation **DiagramOwner** Operation **InsertAnnotatedElementAt OwningElement** Operation **InsertConstrainedElementAt** Operation **AppliedElement Element InsertAnnotatedElementAt InsertConstrainedElementAt InsertPackagedElementRelationshipAt IsElementVisible LinkedOwner Owner OwningElement SetElementVisible TextLabelElement** Operation **Owner** Operation **LinkedOwner** Operation **Element** Operation **IsElementVisible SetElementVisible** Operation **TextLabelElement** Operation **InsertPackagedElementRelationshipAt** Operation **AppliedElement**

Operation **IUMLElement::AllAppliableStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLStereotype.					

Operation **IUMLElement::AppliedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLStereotype.					

Operation **IUMLElement::ApplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLElement::ApplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			

	return	**return**	**IUMLStereotype Application**

Operation **IUMLElement::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			
documentation	Use this function to erase the element from the model and all diagrams. Use IUMLGuiDiagram::EraseFromDiagram to erase from diagram only.					

Operation **IUMLElement::FindPredefinedOwnedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	**ENUMUMLPredefinedElement**			
	bRecursive	**in**	**bool**			
	return	**return**	**IUMLData**			

Operation **IUMLElement::GetOwnedElementsOfKind**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	bRecursive	**in**	**bool**			
	return	**return**	**IUMLDataList**			
documentation	get all owned elements of the specified kind (*strKind*)					

Operation **IUMLElement::GetStereotypeApplicationForPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**IUMLStereotype Application**			

Operation **IUMLElement::GetStereotypeApplicationForStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**IUMLStereotype Application**			

Operation **IUMLElement::InsertOwnedCommentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLComment**			

Operation **IUMLElement::IsPredefinedStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**bool**			

Operation **IUMLElement::IsStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**bool**			

Operation **IUMLElement::OwnedComments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLComment.					

Operation **IUMLElement::OwnedDocComment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComment**			

Operation **IUMLElement::OwnedDocCommentBody**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLElement::OwnedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

Operation **IUMLElement::Owner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLElement::StereotypeApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLStereotypeApplication.					

Operation **IUMLElement::UnapplyPredefinedStereotype**

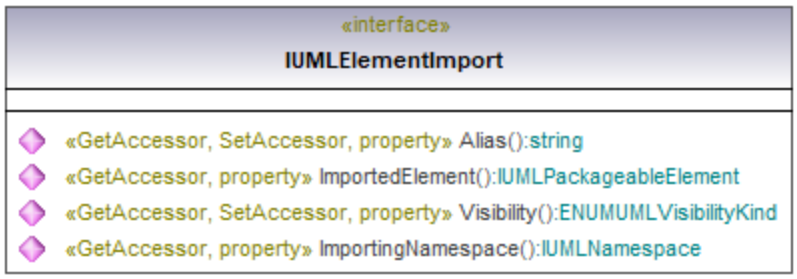
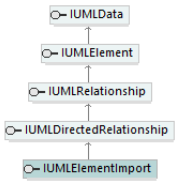
parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**void**			

Operation **IUMLElement::UnapplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**void**			

17.5.3.5.59 UModelAPI - IUMLElementImport

Interface **IUMLElementImport**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamespace** Operation **InsertElementImportAt** Operation **InsertElementImportAt**

Operation **IUMLElementImport::Alias**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLElementImport::ImportedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLElementImport::ImportingNamespace**

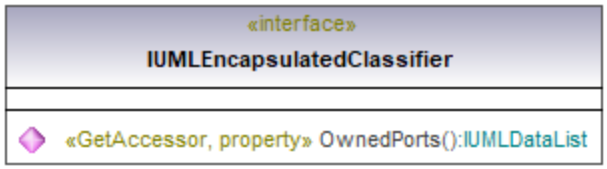
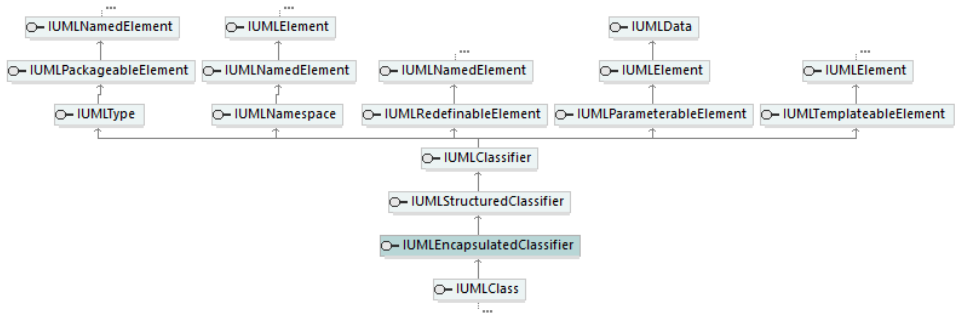
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLElementImport::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

17.5.3.5.60 UModelAPI - IUMLEncapsulatedClassifier

Interface **IUMLEncapsulatedClassifier**

diagram	
hierarchy	

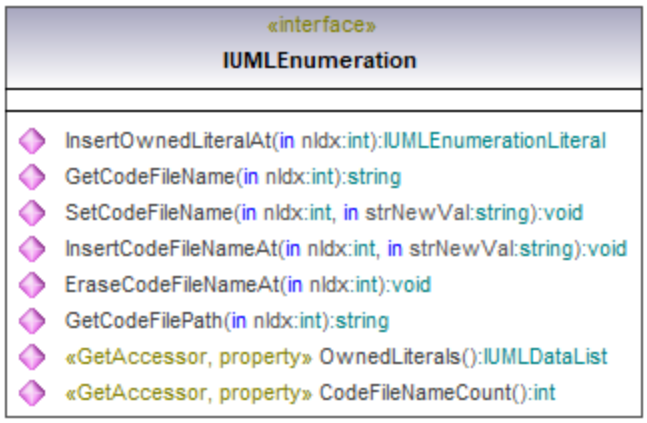
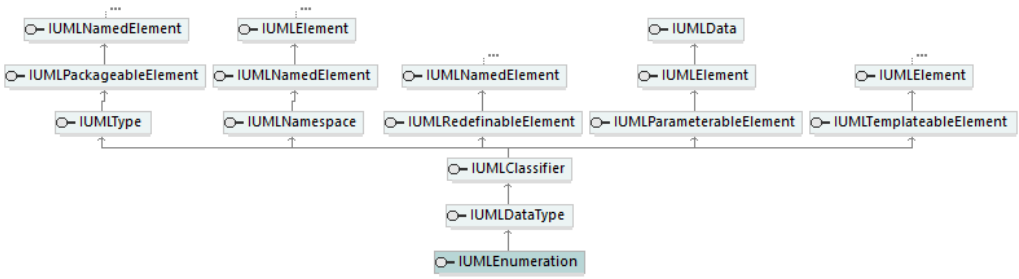
Operation **IUMLEncapsulatedClassifier::OwnedPorts**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.61 UModelAPI - IUMLEnumeration

Interface **IUMLEnumeration**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLEnumerationLiteral** Operation **Enumeration** Operation **Enumeration**

Operation **IUMLEnumeration::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLEnumeration::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLEnumeration::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IUMLEnumeration::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	return	**return**	**string**
documentation	get the full code file path		

Operation **IUMLEnumeration::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLEnumeration::InsertOwnedLiteralAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLEnumerationLiteral**			

Operation **IUMLEnumeration::OwnedLiterals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLEnumerationLiteral.					

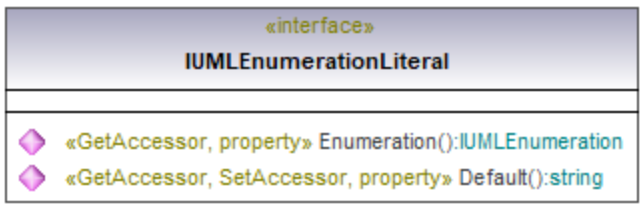
Operation **IUMLEnumeration::SetCodeFileName**

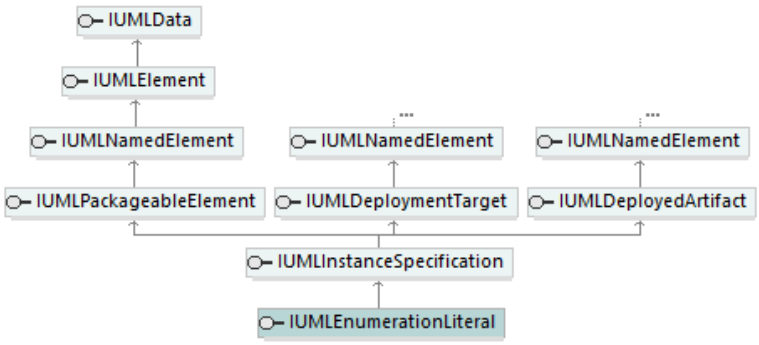
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.62 UModelAPI - IUMLEnumerationLiteral

Interface **IUMLEnumerationLiteral**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLEnumeration** Operation **InsertOwnedLiteralAt** Operation **InsertOwnedLiteralAt**

Operation **IUMLEnumerationLiteral::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

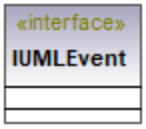
Operation **IUMLEnumerationLiteral::Enumeration**

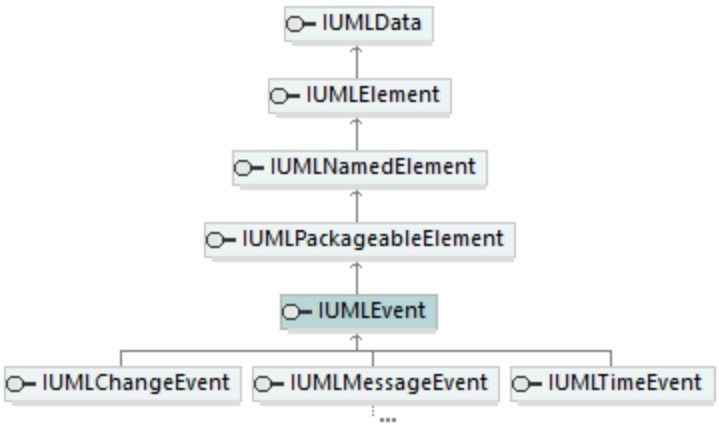
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEnumeration**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.63 UModelAPI - IUMLEvent

Interface **IUMLEvent**

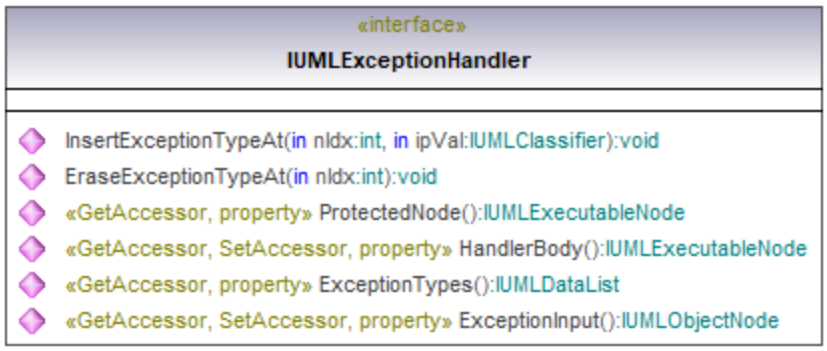
diagram	

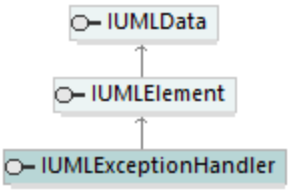
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLOccurrenceSpecification** Interface **IUMLTrigger** Operation **Event** **OccurringEvent** Operation **OccurringEvent** Operation **Event**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.64 UModelAPI - IUMLExceptionHandler

Interface **IUMLExceptionHandler**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLExecutableNode** Operation **InsertHandlerAt** Operation **InsertHandlerAt**

Operation **IUMLExceptionHandler::EraseExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLExceptionHandler::ExceptionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLObjectNode**			

Operation **IUMLExceptionHandler::ExceptionTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLExceptionHandler::HandlerBody**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutableNode**			

Operation **IUMLExceptionHandler::InsertExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

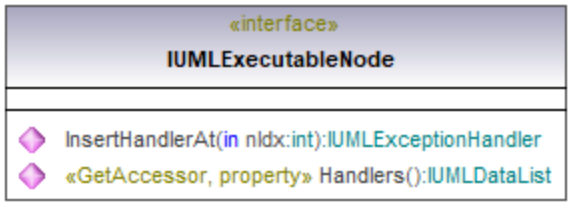
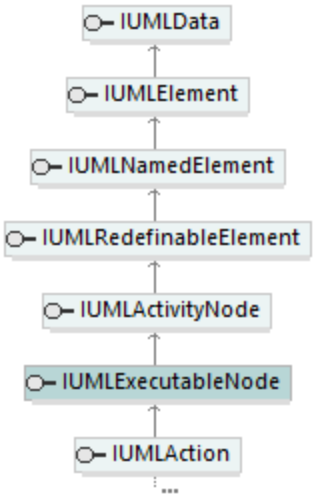
Operation **IUMLExceptionHandler::ProtectedNode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutableNode**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.65 UModelAPI - IUMLExecutableNode

Interface **IUMLExecutableNode**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLExceptionHandler** Operation **HandlerBody** **ProtectedNode** Operation **HandlerBody** **ProtectedNode**

Operation **IUMLExecutableNode::Handlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLExceptionHandler.					

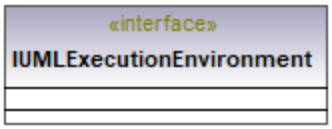
Operation **IUMLExecutableNode::InsertHandlerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExceptionHandler**			

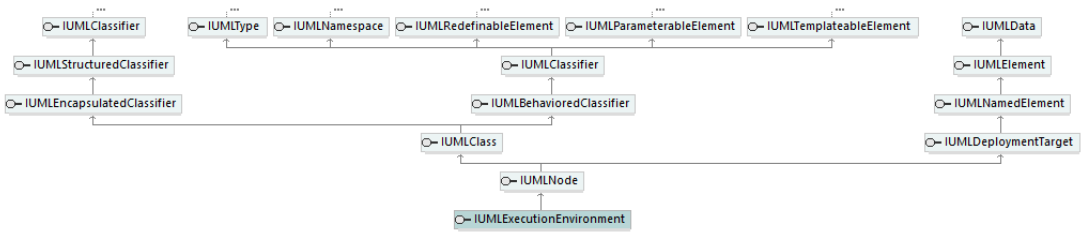
17.5.3.5.66 UModelAPI - IUMLExecutionEnvironment

Interface **IUMLExecutionEnvironment**

diagram



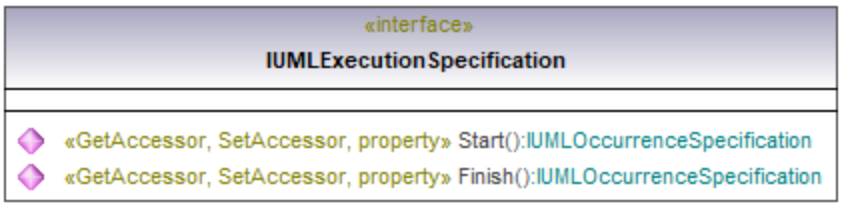
hierarchy



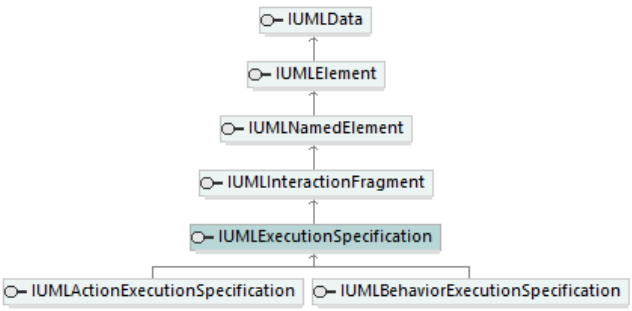
17.5.3.5.67 UModelAPI - IUMLExecutionSpecification

Interface **IUMLExecutionSpecification**

diagram



hierarchy



typedElements	Interface **IUMLDataAll** Interface **IUMLOccurrenceSpecification**	Operation **ExecutionSpecificationFinish** **ExecutionSpecificationStart** Operation **ExecutionSpecificationFinish** **ExecutionSpecificationStart**

Operation **IUMLExecutionSpecification::Finish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOccurrenceSpecification**			

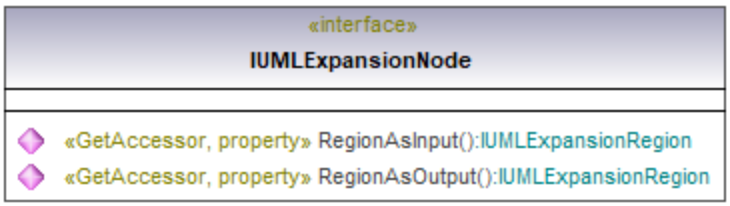
Operation **IUMLExecutionSpecification::Start**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOccurrenceSpecification**			

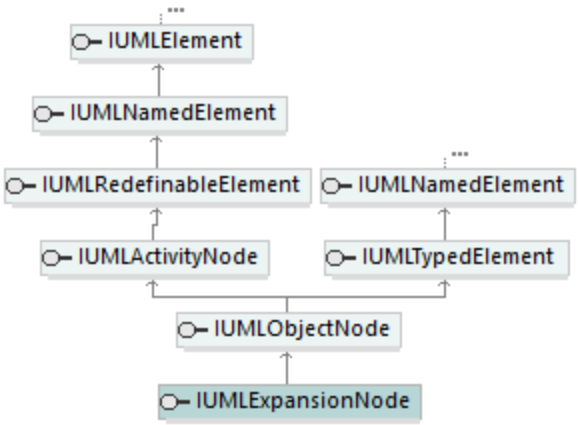
17.5.3.5.68 UModelAPI - IUMLExpansionNode

Interface **IUMLExpansionNode**

diagram



hierarchy



typedElements	Interface **IUMLDataAll**	Operation **InsertInputElementAt** **InsertOutputElementAt**

	Interface **IUMLExpansionRegion**	Operation **InsertInputElementAt** **InsertOutputElementAt**

Operation **IUMLExpansionNode::RegionAsInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpansionRegion**			

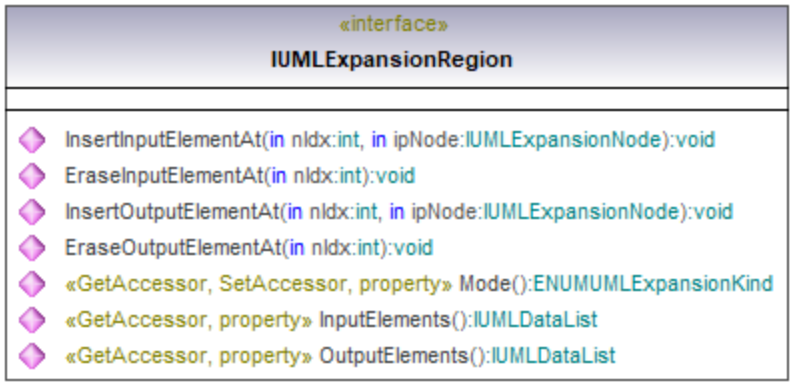
Operation **IUMLExpansionNode::RegionAsOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpansionRegion**			

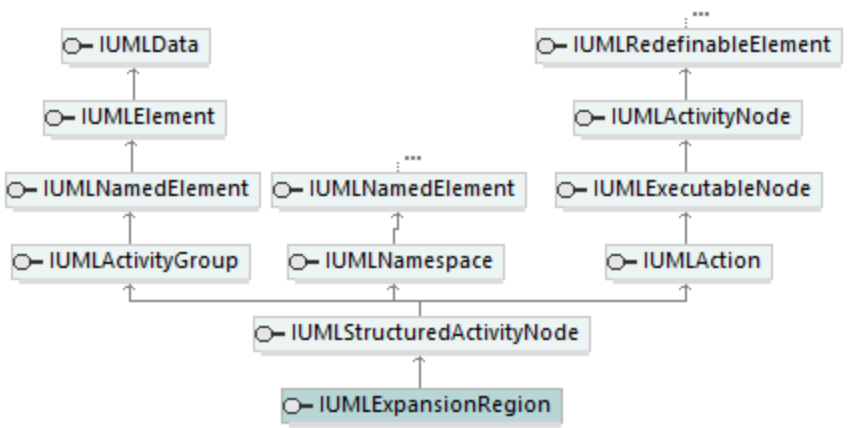
17.5.3.5.69 UModelAPI - IUMLExpansionRegion

Interface **IUMLExpansionRegion**

diagram



hierarchy



typedElements	Interface **IUMLDataAll** Interface **IUMLExpansionNode**	Operation **RegionAsInput** **RegionAsOutput** Operation **RegionAsInput** **RegionAsOutput**

Operation **IUMLExpansionRegion::EraseInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLExpansionRegion::EraseOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLExpansionRegion::InputElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLExpansionNode.					

Operation **IUMLExpansionRegion::InsertInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode**			
	return	**return**	**void**			

Operation **IUMLExpansionRegion::InsertOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode**			
	return	**return**	**void**			

Operation **IUMLExpansionRegion::Mode**

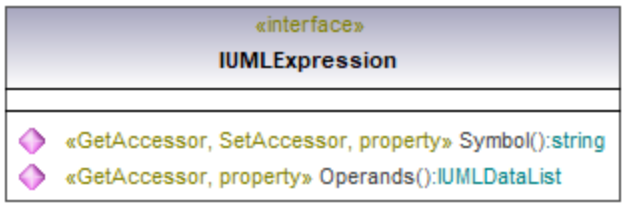
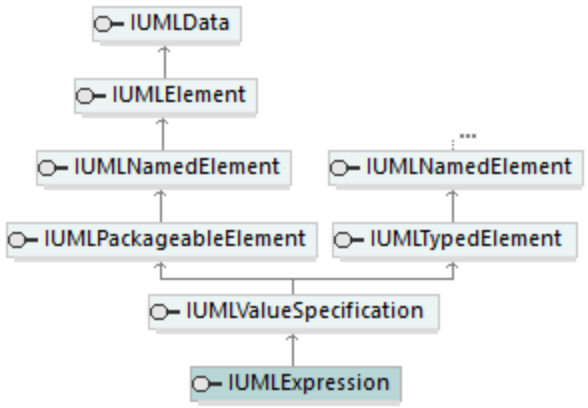
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLExpansionKind**			

Operation **IUMLExpansionRegion::OutputElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLExpansionNode.					

17.5.3.5.70 UModelAPI - IUMLExpression

Interface **IUMLExpression**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLValueSpecification** Operation **Expression** Operation **Expression**

Operation **IUMLExpression::Operands**

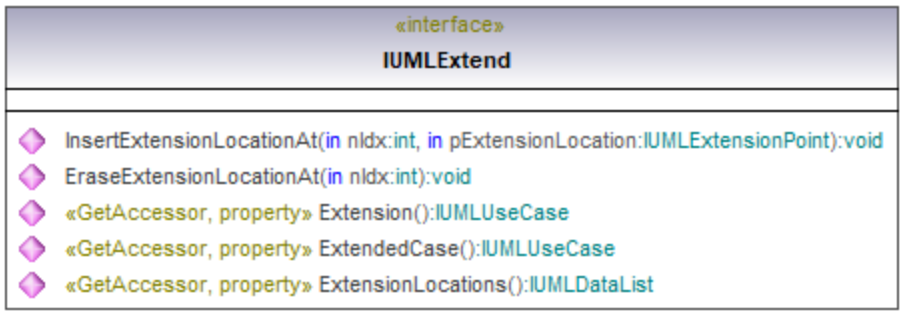
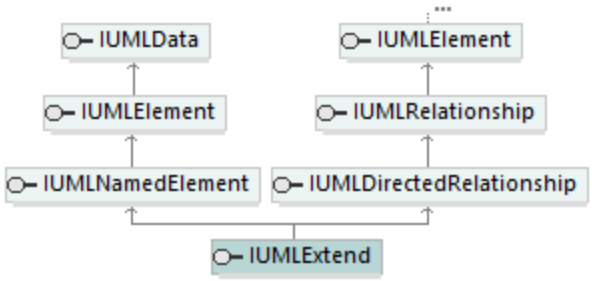
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLValueSpecification.					

Operation **IUMLExpression::Symbol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.3.5.71 UModelAPI - IUMLExtend

Interface **IUMLExtend**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLUseCase** Operation **InsertExtendAt** Operation **InsertExtendAt**

Operation **IUMLExtend::EraseExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLExtend::ExtendedCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLExtend::Extension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLExtend::ExtensionLocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLExtensionPoint.					

Operation **IUMLExtend::InsertExtensionLocationAt**

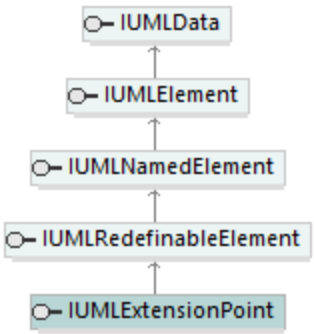
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pExtensionLocation	**in**	**IUMLExtensionPoint**			
	return	**return**	**void**			

17.5.3.5.72 UModelAPI - IUMLExtensionPoint

Interface **IUMLExtensionPoint**

diagram

hierarchy



typedElements

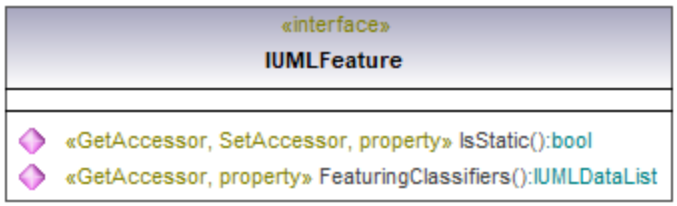
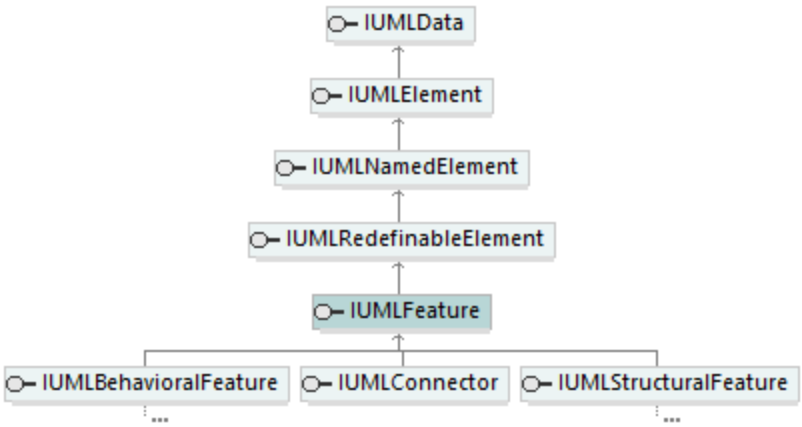
Interface IUMLDataAll	Operation InsertExtensionLocationAt
	Operation InsertExtensionPointAt
Interface IUMLExtend	Operation InsertExtensionLocationAt
Interface IUMLUseCase	Operation InsertExtensionPointAt

Operation **IUMLExtensionPoint::UseCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

17.5.3.5.73 UModelAPI - IUMLFeature

Interface **IUMLFeature**

diagram	
hierarchy	

Operation **IUMLFeature::FeaturingClassifiers**

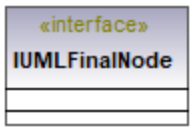
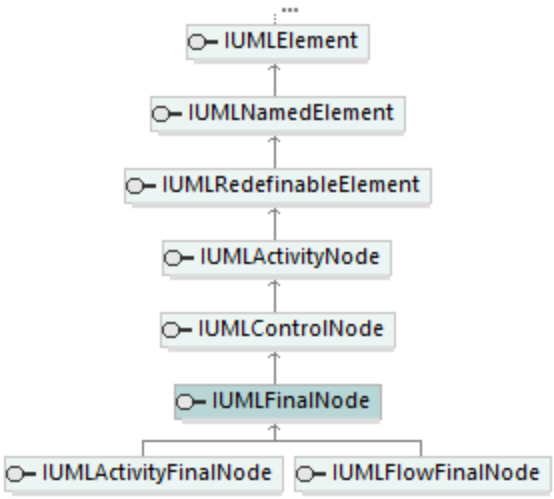
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLFeature::IsStatic**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.3.5.74 UModelAPI - IUMLFinalNode

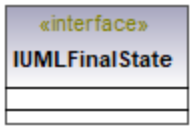
Interface **IUMLFinalNode**

diagram	
hierarchy	

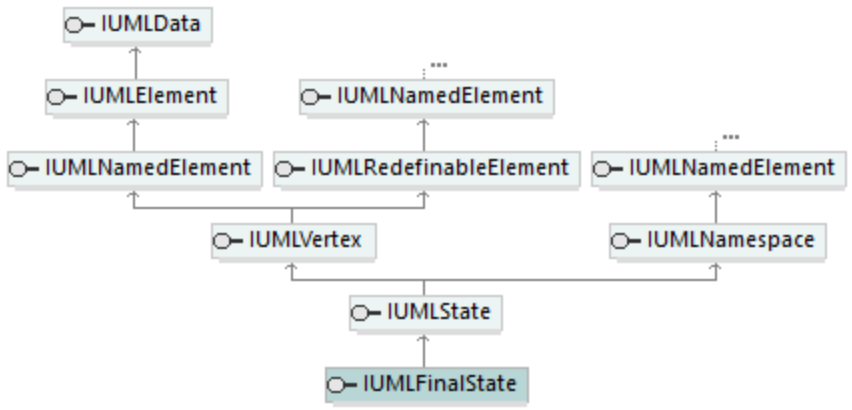
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.75 UModelAPI - IUMLFinalState

Interface **IUMLFinalState**

diagram	

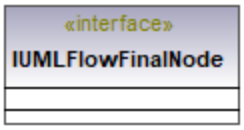
hierarchy



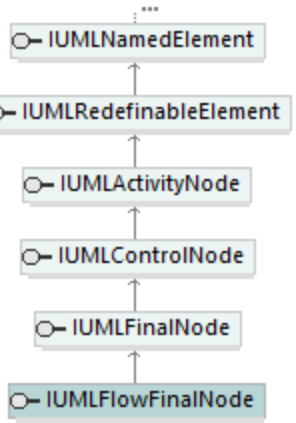
17.5.3.5.76 UModelAPI - IUMLFlowFinalNode

Interface **IUMLFlowFinalNode**

diagram

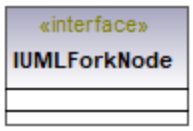
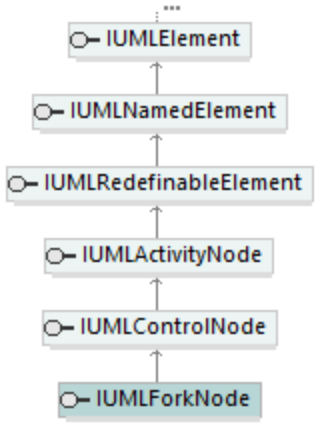


hierarchy



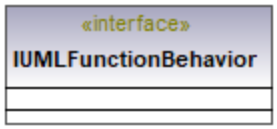
17.5.3.5.77 UModelAPI - IUMLForkNode

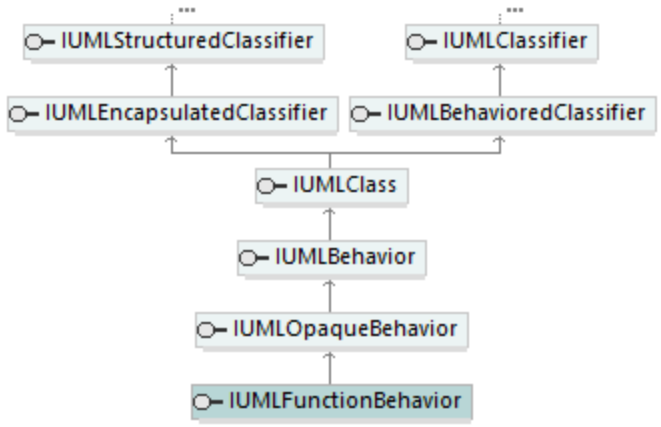
Interface **IUMLForkNode**

diagram	
hierarchy	

17.5.3.5.78 UModelAPI - IUMLFunctionBehavior

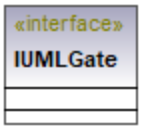
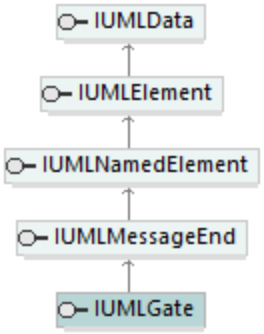
Interface **IUMLFunctionBehavior**

diagram	

hierarchy	

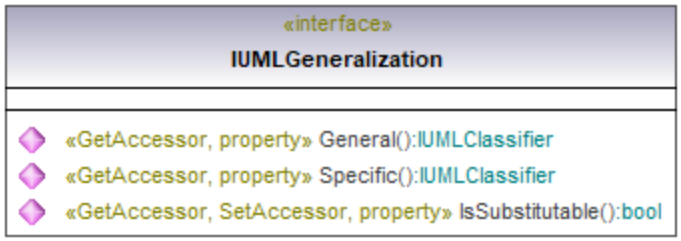
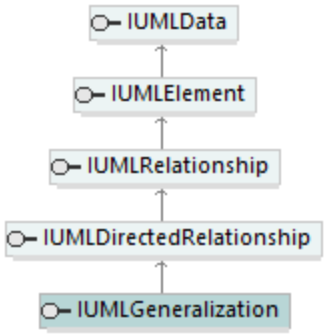
17.5.3.5.79 UModelAPI - IUMLGate

Interface **IUMLGate**

diagram		
hierarchy		
typedElements	Interface **IUMLDataAll** Interface **IUMLInteraction** Interface **IUMLInteractionUse**	Operation **InsertActualGateAt** **InsertFormalGateAt** Operation **InsertFormalGateAt** Operation **InsertActualGateAt**

17.5.3.5.80 UModelAPI - IUMLGeneralization

Interface **IUMLGeneralization**

diagram	
hierarchy	
typedElements	Interface IUMLClassifier Interface IUMLDataAll Operation InsertGeneralizationAt Operation InsertGeneralizationAt

Operation **IUMLGeneralization::General**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLClassifier			

Operation **IUMLGeneralization::IsSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

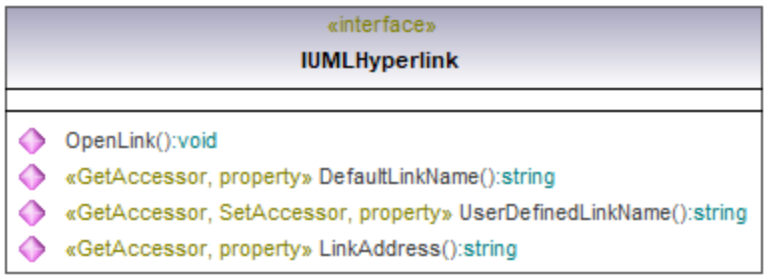
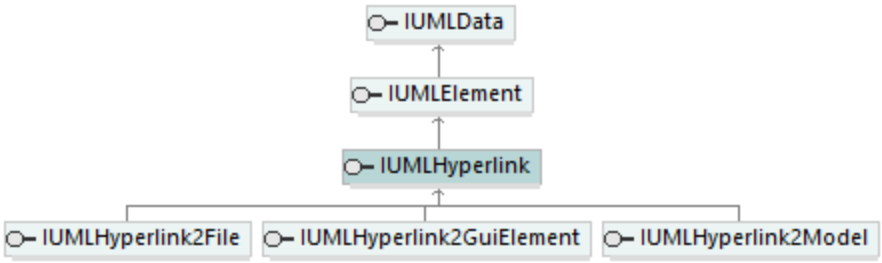
Operation **IUMLGeneralization::Specific**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLClassifier			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.3.5.81 UModelAPI - IUMLHyperlink

Interface **IUMLHyperlink**

diagram	
hierarchy	

Operation **IUMLHyperlink::DefaultLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLHyperlink::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLHyperlink::OpenLink**

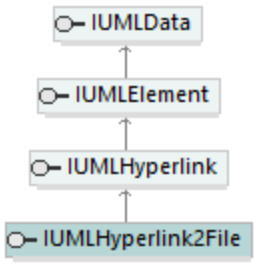
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLHyperlink::UserDefinedLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.3.5.82 UModelAPI - IUMLHyperlink2File

Interface **IUMLHyperlink2File**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamedElement** Operation **InsertOwnedHyperlink2FileAt** Operation **InsertOwnedHyperlink2FileAt**

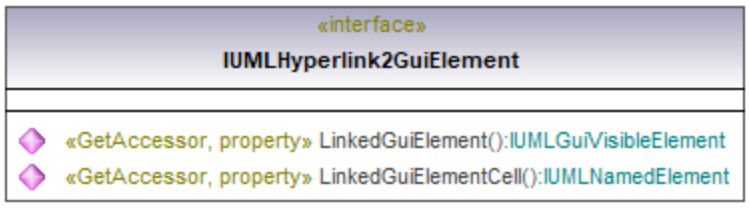
Operation **IUMLHyperlink2File::SetHyperlinkFileAddress**

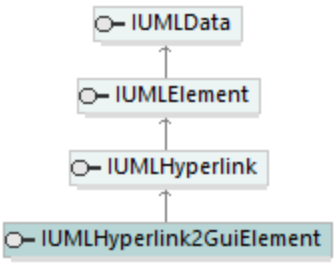
parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.83 UModelAPI - IUMLHyperlink2GuiElement

Interface **IUMLHyperlink2GuiElement**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamedElement** Operation **InsertOwnedHyperlink2GuiElementAt** Operation **InsertOwnedHyperlink2GuiElementAt**

Operation **IUMLHyperlink2GuiElement::LinkedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiVisibleElement**			

Operation **IUMLHyperlink2GuiElement::LinkedGuiElementCell**

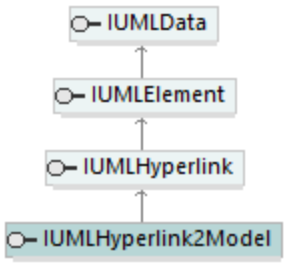
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamedElement**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.84 UModelAPI - IUMLHyperlink2Model

Interface **IUMLHyperlink2Model**

diagram	

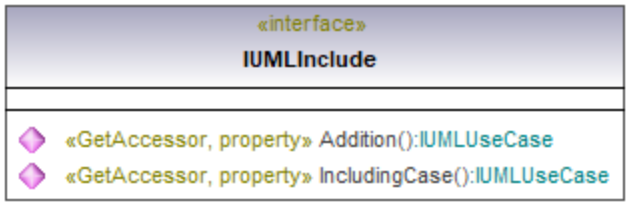
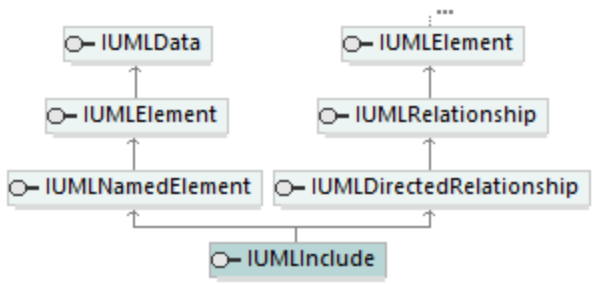
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamedElement** Operation **InsertOwnedHyperlink2ModelAt** Operation **InsertOwnedHyperlink2ModelAt**

Operation **IUMLHyperlink2Model::LinkedModelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData**			

17.5.3.5.85 UModelAPI - IUMLInclude

Interface **IUMLInclude**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLUseCase** Operation **InsertIncludeAt** Operation **InsertIncludeAt**

Operation **IUMLInclude::Addition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

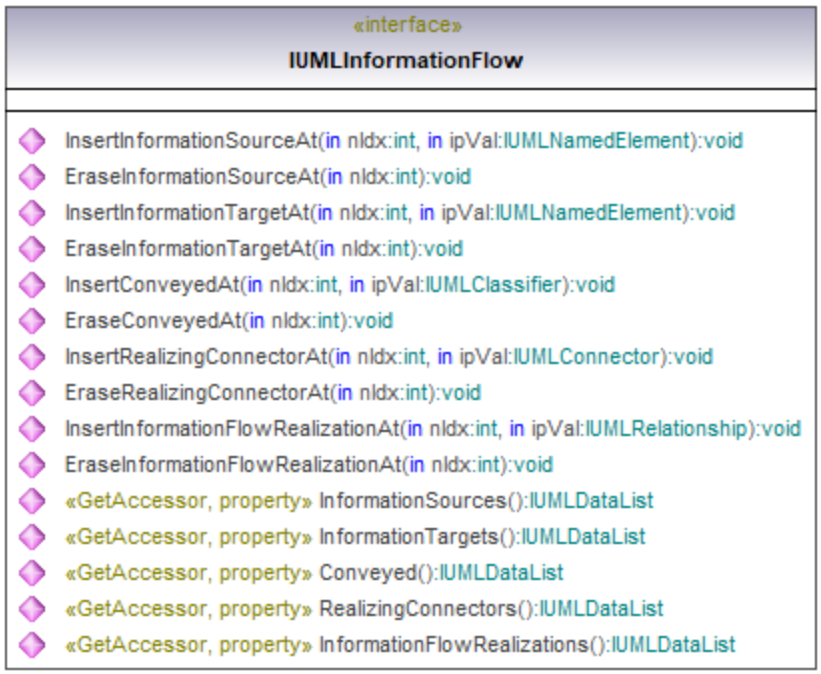
Operation **IUMLInclude::IncludingCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

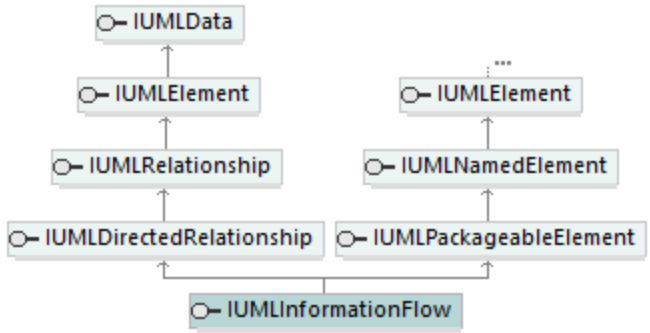
17.5.3.5.86 UModelAPI - IUMLInformationFlow

Interface **IUMLInformationFlow**

diagram



hierarchy



Operation **IUMLInformationFlow::Conveyed**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLInformationFlow::EraseConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::EraseInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::EraseInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::EraseInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::EraseRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InformationFlowRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLInformationFlow::InformationSources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLInformationFlow::InformationTargets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLInformationFlow::InsertConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLRelationship**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnector**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::RealizingConnectors**

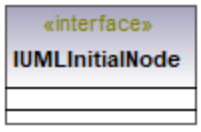
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

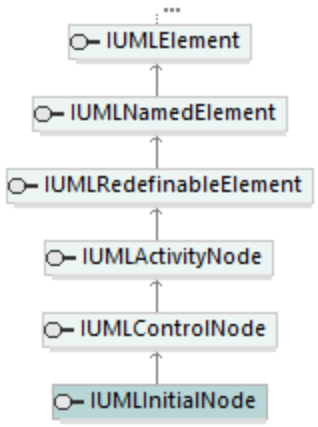
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.87 UModelAPI - IUMLInitialNode

Interface **IUMLInitialNode**

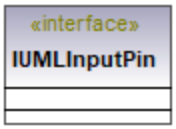
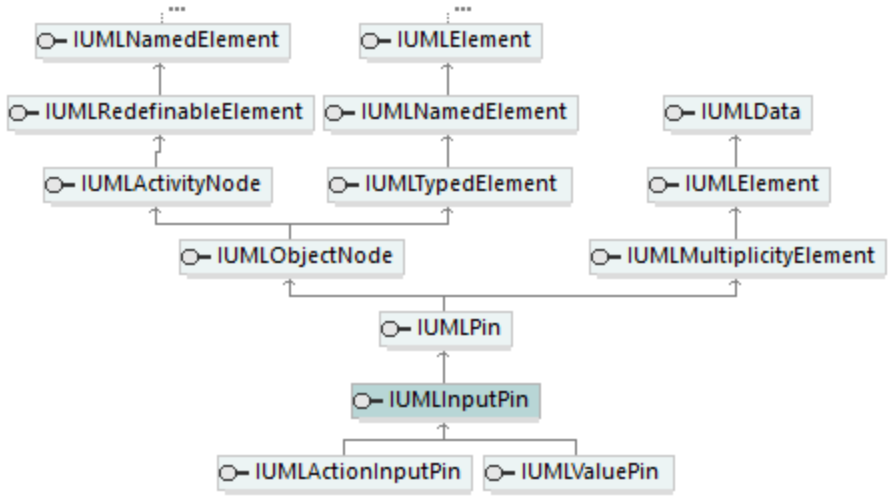
diagram	



hierarchy	

17.5.3.5.88 UModelAPI - IUMLInputPin

Interface **IUMLInputPin**

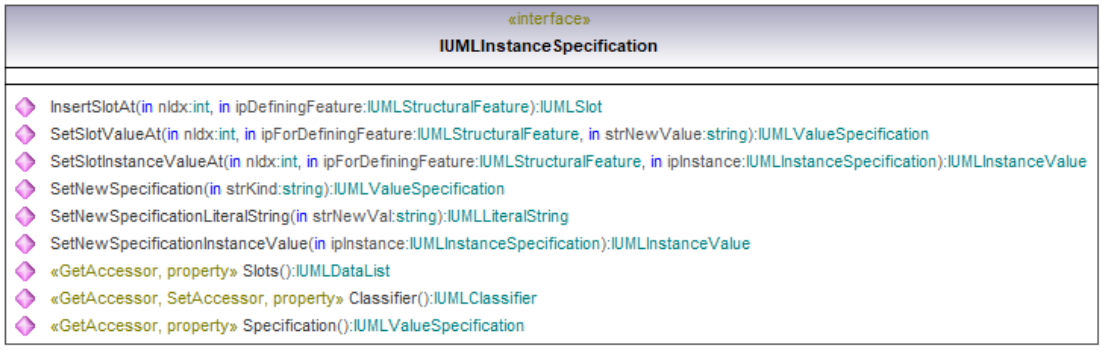
diagram	
hierarchy	
typedElements	Interface **IUMLCallOperationAction** Interface **IUMLDataAll** Operation **CallTarget** **SetNewCallTarget** Operation **CallTarget** **InsertArgumentAt** **InsertArgumentOfKindAt**

	Interface **IUMLInvocationAction**	Operation **InsertInputValueAt** **SetNewCallTarget** **SetNewSignalTarget** **SignalTarget** **InsertArgumentAt** **InsertArgumentOfKindAt**
	Interface **IUMLOpaqueAction**	Operation **InsertInputValueAt**
	Interface **IUMLSendSignalAction**	Operation **SetNewSignalTarget** **SignalTarget**

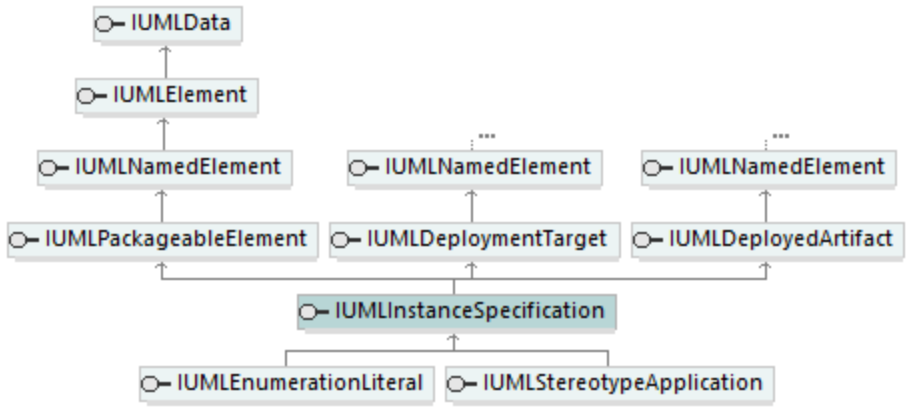
17.5.3.5.89 UModelAPI - IUMLInstanceSpecification

Interface **IUMLInstanceSpecification**

diagram



hierarchy



typedElements	Interface **IUMLConstraint**	Operation **SetNewSpecificationInstanceValue**
	Interface **IUMLDataAll**	Operation **InsertSlotInstanceValueAt** **Instance** **OwningInstance** **OwningInstanceSpec** **SetNewDefaultValueInstanceValue**

		SetNewSpecificationInstanceValue SetSlotInstanceValueAt
	Interface IUMLInstanceSpecification	Operation SetNewSpecificationInstanceValue SetSlotInstanceValueAt
	Interface IUMLInstanceValue	Operation Instance
	Interface IUMLParameter	Operation SetNewDefaultValueInstanceValue
	Interface IUMLProperty	Operation SetNewDefaultValueInstanceValue
	Interface IUMLSlot	Operation InsertSlotInstanceValueAt OwningInstance
	Interface IUMLValueSpecification	Operation OwningInstanceSpec

Operation **IUMLInstanceSpecification::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLClassifier			

Operation **IUMLInstanceSpecification::InsertSlotAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	IUMLStructuralFeature			
	return	**return**	IUMLSlot			

Operation **IUMLInstanceSpecification::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	IUMLValueSpecification			

Operation **IUMLInstanceSpecification::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	IUMLInstanceSpecification			
	return	**return**	IUMLInstanceValue			

Operation **IUMLInstanceSpecification::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	IUMLLiteralString			

Operation **IUMLInstanceSpecification::SetSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	IUMLStructuralFeature			
	ipInstance	**in**	IUMLInstanceSpecification			

	return	**return**	**IUMLInstanceValue**

Operation **IUMLInstanceSpecification::SetSlotValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	**IUMLStructuralFeature**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInstanceSpecification::Slots**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLSlot.					

Operation **IUMLInstanceSpecification::Specification**

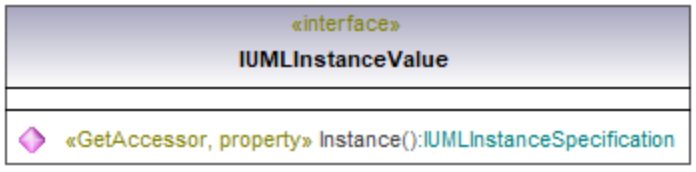
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

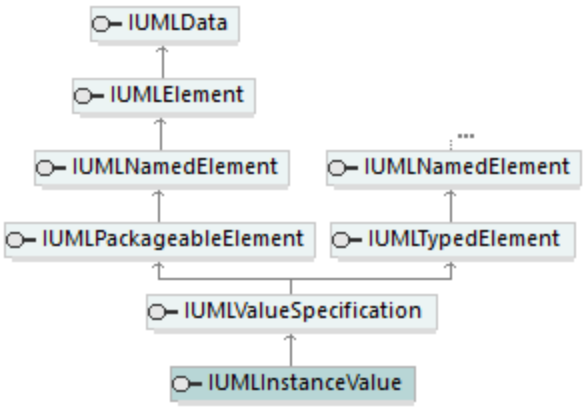
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.90 UModelAPI - IUMLInstanceValue

Interface **IUMLInstanceValue**

diagram



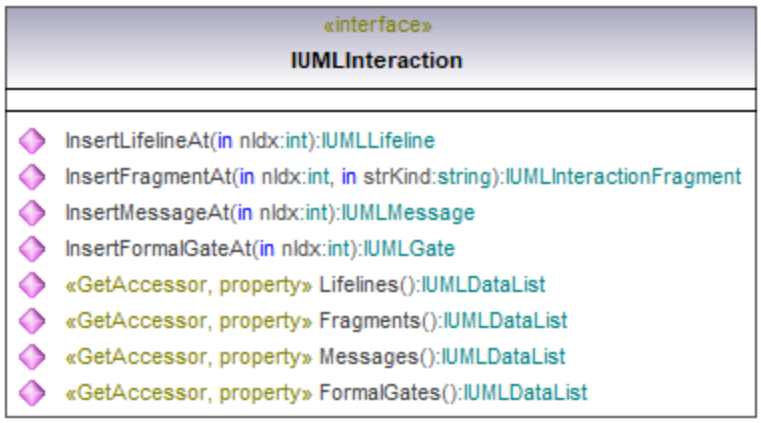
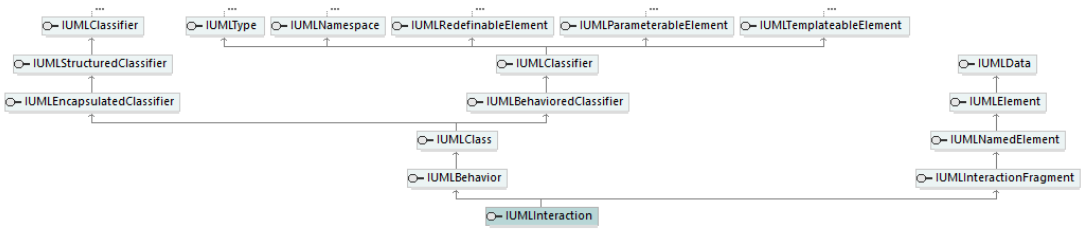
hierarchy	
typedElements	Interface **IUMLConstraint** Interface **IUMLDataAll** Interface **IUMLInstanceSpecification** Interface **IUMLParameter** Interface **IUMLProperty** Interface **IUMLSlot** Operation **SetNewSpecificationInstanceValue** Operation **InsertSlotInstanceValueAt SetNewDefaultValueInstanceValue SetNewSpecificationInstanceValue SetSlotInstanceValueAt** Operation **SetNewSpecificationInstanceValue SetSlotInstanceValueAt** Operation **SetNewDefaultValueInstanceValue** Operation **SetNewDefaultValueInstanceValue** Operation **InsertSlotInstanceValueAt**

Operation **IUMLInstanceValue::Instance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

17.5.3.5.91 UModelAPI - IUMLInteraction

Interface **IUMLInteraction**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInteractionUse** Operation **RefersTo** Operation **RefersTo**

Operation **IUMLInteraction::FormalGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLGate.					

Operation **IUMLInteraction::Fragments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInteractionFragment.					

Operation **IUMLInteraction::InsertFormalGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGate**			

Operation **IUMLInteraction::InsertFragmentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLInteractionFragment**			

Operation **IUMLInteraction::InsertLifelineAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLLifeline**			

Operation **IUMLInteraction::InsertMessageAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLMessage**			

Operation **IUMLInteraction::Lifelines**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLLifeline.					

Operation **IUMLInteraction::Messages**

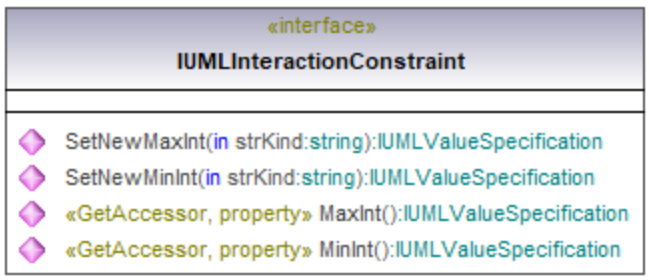
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLMessage.					

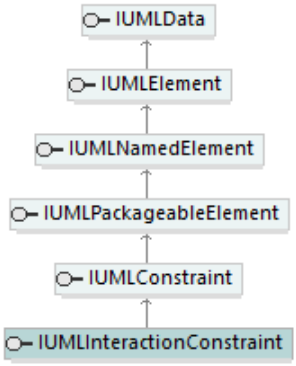
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.92 UModelAPI - IUMLInteractionConstraint

Interface **IUMLInteractionConstraint**

diagram	



hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInteractionOperand** Operation **OperandGuard** **SetNewOperandGuard** Operation **OperandGuard** **SetNewOperandGuard**

Operation **IUMLInteractionConstraint::MaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInteractionConstraint::MinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInteractionConstraint::SetNewMaxInt**

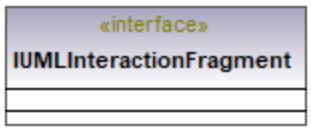
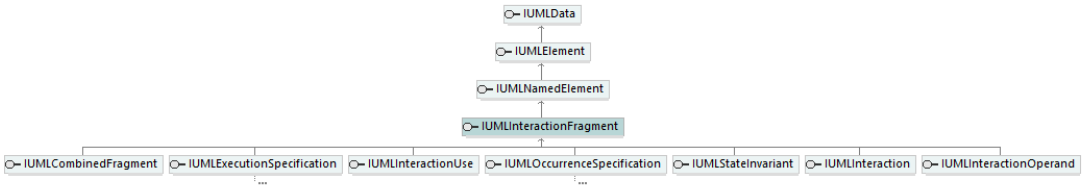
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInteractionConstraint::SetNewMinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

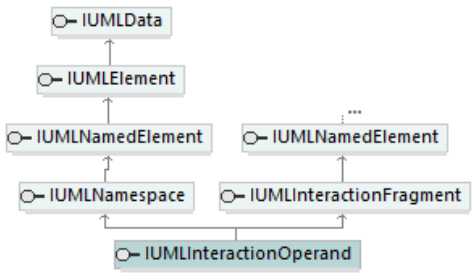
17.5.3.5.93 UModelAPI - IUMLInteractionFragment

Interface **IUMLInteractionFragment**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInteraction** Interface **IUMLLifeline** Operation **InsertCoveredByAt** **InsertFragmentAt** Operation **InsertFragmentAt** Operation **InsertCoveredByAt**

17.5.3.5.94 UModelAPI - IUMLInteractionOperand

Interface **IUMLInteractionOperand**

diagram	
hierarchy	
typedElements	Interface **IUMLCombinedFragment** Interface **IUMLDataAll** Operation **InsertOperandAt** Operation **InsertOperandAt**

Operation **IUMLInteractionOperand::OperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteractionConstraint**			

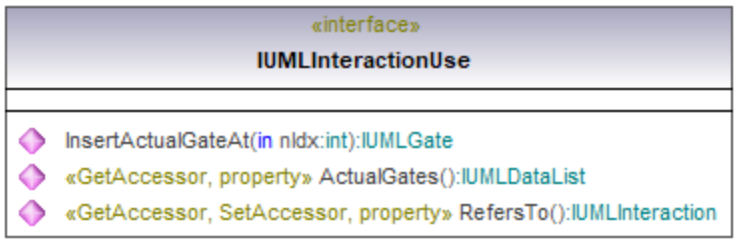
Operation **IUMLInteractionOperand::SetNewOperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteractionConstraint**			

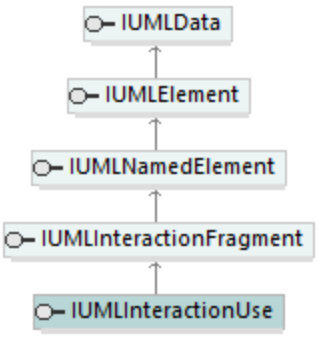
17.5.3.5.95 UModelAPI - IUMLInteractionUse

Interface **IUMLInteractionUse**

diagram



hierarchy



Operation **IUMLInteractionUse::ActualGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLGate.					

Operation **IUMLInteractionUse::InsertActualGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGate**			

Operation **IUMLInteractionUse::RefersTo**

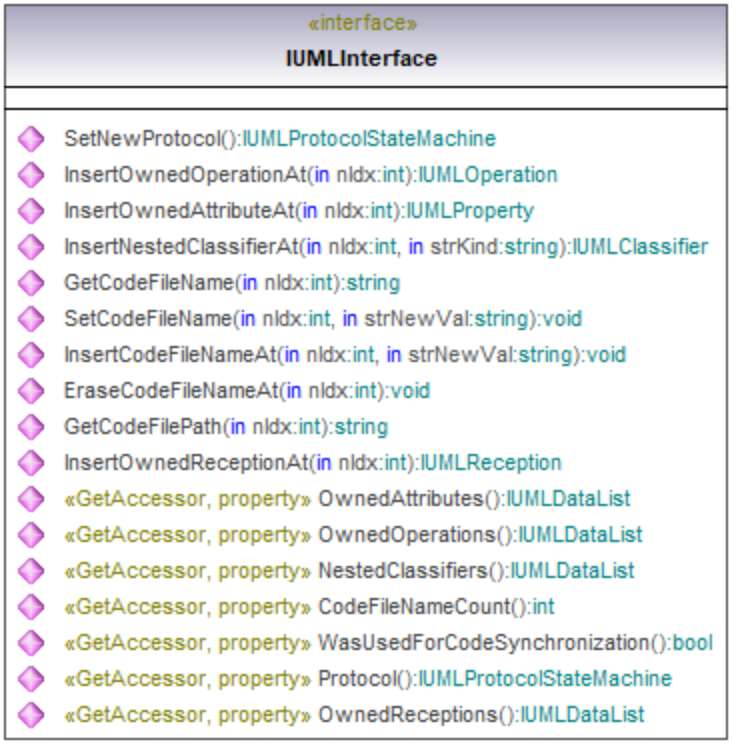
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteraction**			

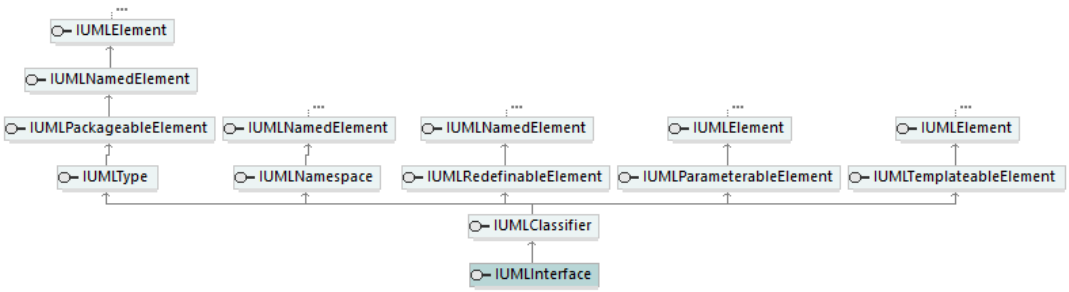
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.96 UModelAPI - IUMLInterface

Interface **IUMLInterface**

diagram



hierarchy	
typedElements	Interface **IUMLBehavioredClassifier** Interface **IUMLClassifier** Interface **IUMLDataAll** Interface **IUMLInterfaceRealization** Interface **IUMLOperation** Interface **IUMLProperty** Interface **IUMLProtocolStateMachine** Interface **IUMLReception** Operation **InsertInterfaceRealizationAt** Operation **NestingInterface** Operation **Contract** **InsertInterfaceRealizationAt** **Interface** **NestingInterface** Operation **Contract** Operation **Interface** Operation **Interface** Operation **Interface** Operation **Interface**

Operation **IUMLInterface::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLInterface::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInterface::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IUMLInterface::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			
documentation	get the full code file path					

Operation **IUMLInterface::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLInterface::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLClassifier**			

Operation **IUMLInterface::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLInterface::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation**			

Operation **IUMLInterface::InsertOwnedReceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLReception**			

Operation **IUMLInterface::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLInterface::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLInterface::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOperation.					

Operation **IUMLInterface::OwnedReceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLInterface::Protocol**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLProtocolStateMachine**

Operation **IUMLInterface::SetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLInterface::SetNewProtocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine**			

Operation **IUMLInterface::WasUsedForCodeSynchronization**

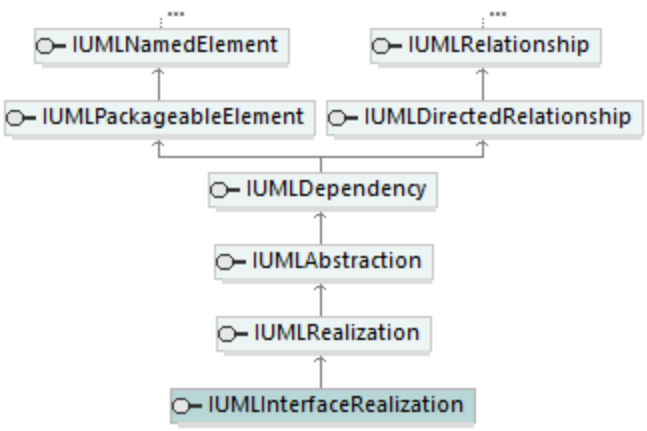
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.3.5.97 UModelAPI - IUMLInterfaceRealization

Interface **IUMLInterfaceRealization**

diagram

hierarchy



typed Elements	Interface IUMLBehavioredClassifier Interface IUMLDataAll	Operation InsertInterfaceRealizationAt Operation InsertInterfaceRealizationAt

Operation **IUMLInterfaceRealization::Contract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInterface			

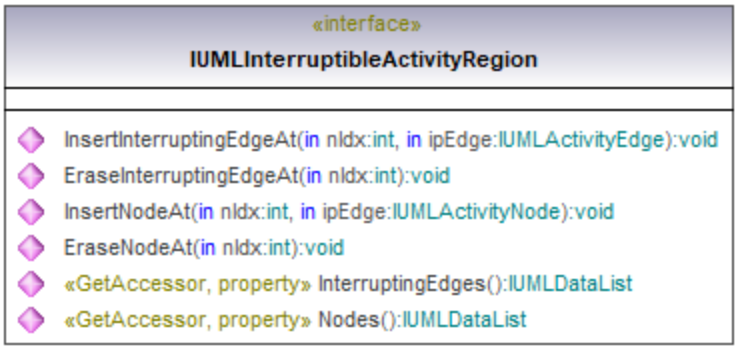
Operation **IUMLInterfaceRealization::ImplementingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavioredClassifier			

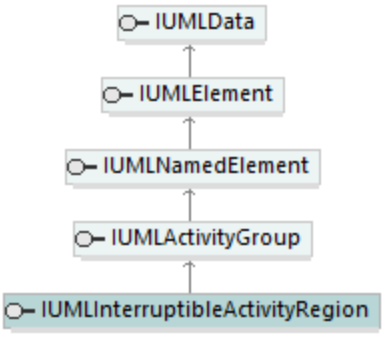
17.5.3.5.98 UModelAPI - IUMLInterruptibleActivityRegion

Interface **IUMLInterruptibleActivityRegion**

diagram



hierarchy



Operation **IUMLInterruptibleActivityRegion::EraseInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::InsertInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge**			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityNode**			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::InterruptingEdges**

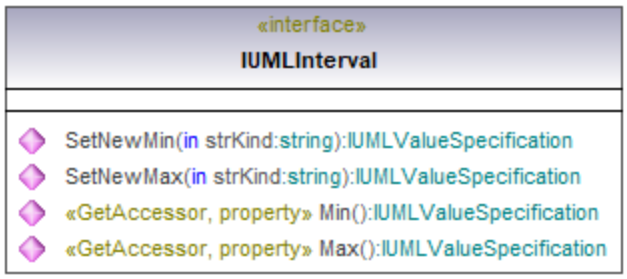
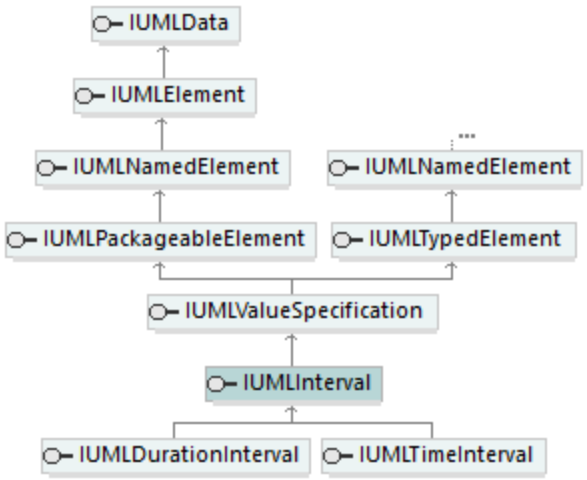
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

Operation **IUMLInterruptibleActivityRegion::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityNode.					

17.5.3.5.99 UModelAPI - IUMLInterval

Interface **IUMLInterval**

diagram	
hierarchy	

Operation **IUMLInterval::Max**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInterval::Min**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInterval::SetNewMax**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInterval::SetNewMin**

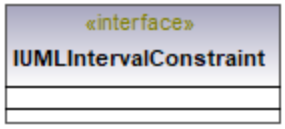
parameter	name	direction	type	type modifier	multiplicity	default

	strKind **return**	**in** **return**	**string** **IUMLValueSpecification**

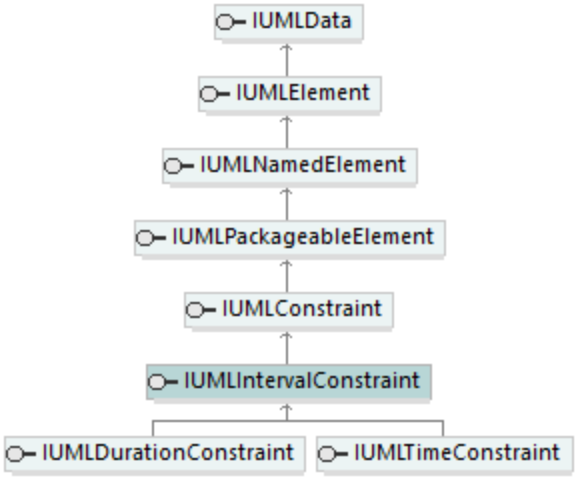
17.5.3.5.100 UModelAPI - IUMLIntervalConstraint

Interface **IUMLIntervalConstraint**

diagram

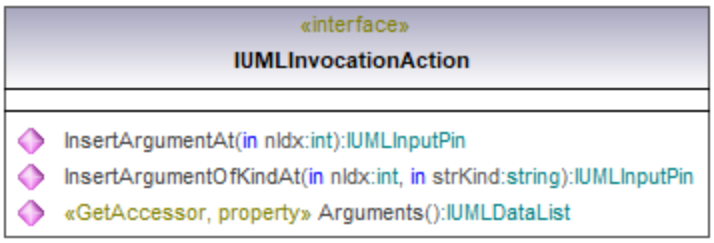
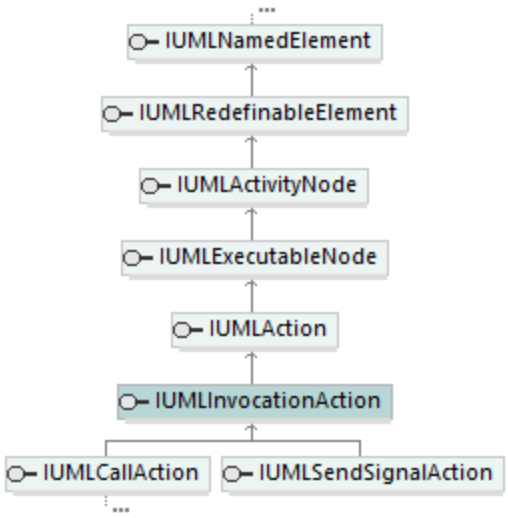


hierarchy



17.5.3.5.101 UModelAPI - IUMLInvocationAction

Interface **IUMLInvocationAction**

diagram	
hierarchy	

Operation **IUMLInvocationAction::Arguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInputPin.					

Operation **IUMLInvocationAction::InsertArgumentAt**

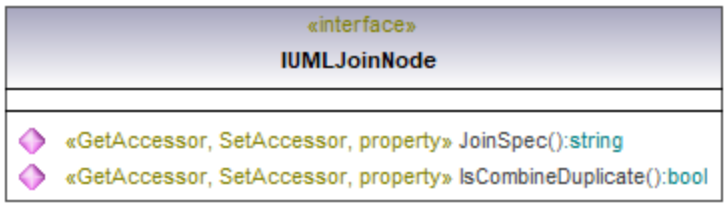
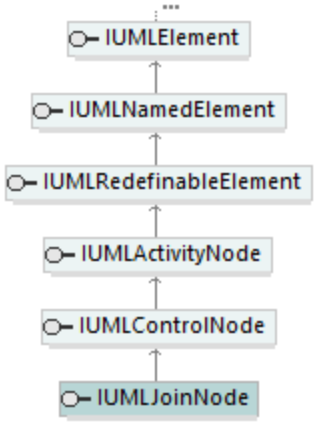
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin**			

Operation **IUMLInvocationAction::InsertArgumentOfKindAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin**			

17.5.3.5.102 UModelAPI - IUMLJoinNode

Interface **IUMLJoinNode**

diagram	
hierarchy	

Operation **IUMLJoinNode::IsCombineDuplicate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

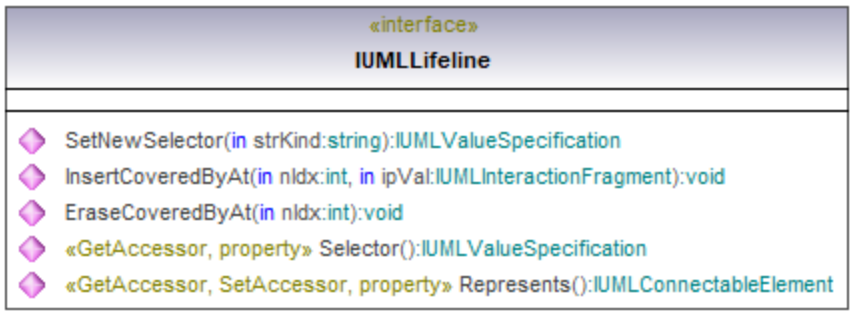
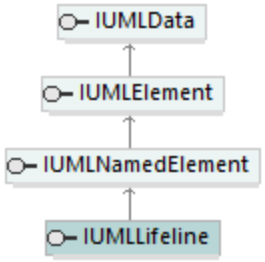
Operation **IUMLJoinNode::JoinSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.103 UModelAPI - IUMLLifeline

Interface **IUMLLifeline**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInteraction** Interface **IUMLMessage** Interface **IUMLOccurrenceSpecification** Interface **IUMLStateInvariant** Operation **Covered** **GetSourceLifeline** **GetTargetLifeline** **InsertLifelineAt** Operation **InsertLifelineAt** Operation **GetSourceLifeline** **GetTargetLifeline** Operation **Covered** Operation **Covered**

Operation **IUMLLifeline::EraseCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLLifeline::InsertCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLInteractionFragment**			
	return	**return**	**void**			

Operation **IUMLLifeline::Represents**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLConnectableElement**

Operation **IUMLLifeline::Selector**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

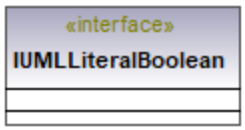
Operation **IUMLLifeline::SetNewSelector**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

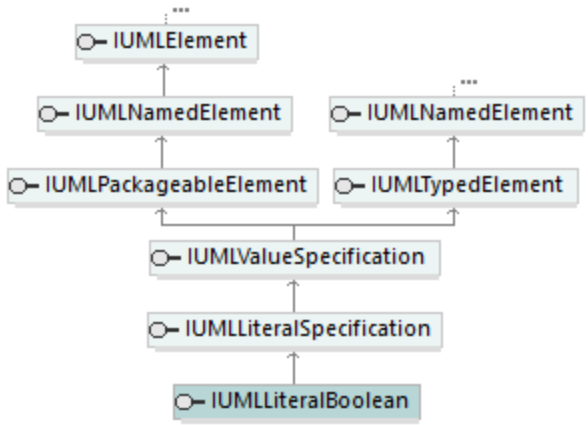
17.5.3.5.104 UModelAPI - IUMLLiteralBoolean

Interface **IUMLLiteralBoolean**

diagram

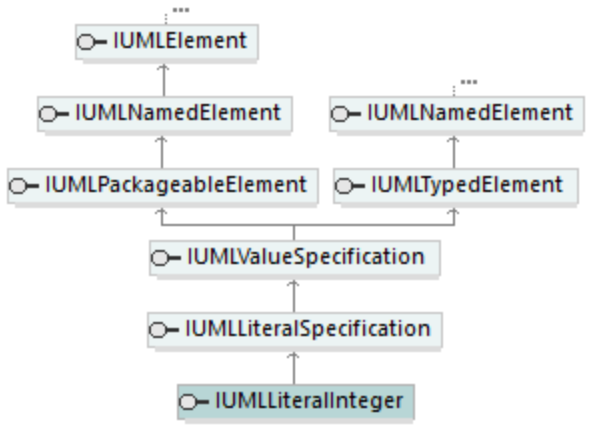


hierarchy



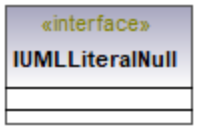
17.5.3.5.105 UModelAPI - IUMLLiteralInteger

Interface **IUMLLiteralInteger**

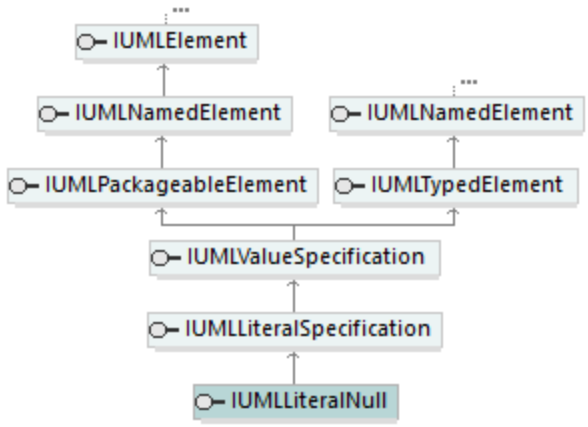
diagram	
hierarchy	

17.5.3.5.106 UModelAPI - IUMLLiteralNull

Interface **IUMLLiteralNull**

diagram	

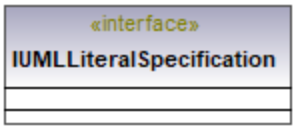
hierarchy



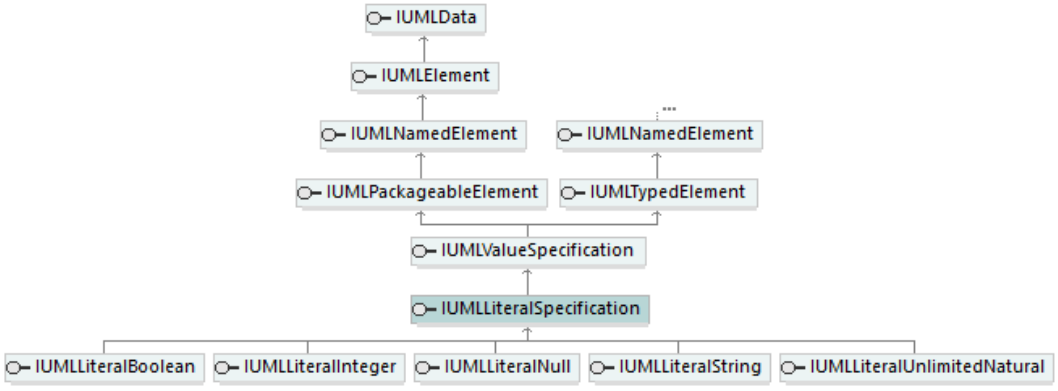
17.5.3.5.107 UModelAPI - IUMLLiteralSpecification

Interface **IUMLLiteralSpecification**

diagram

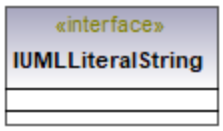
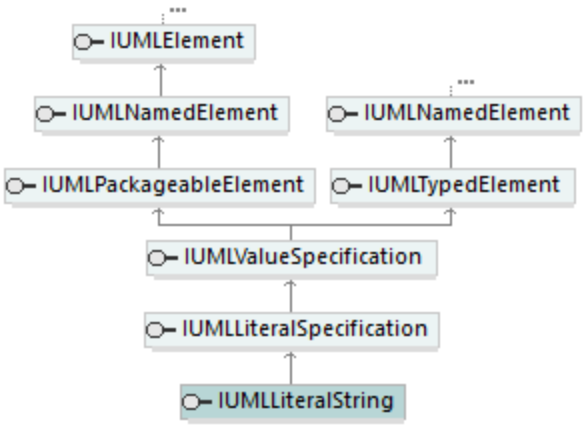


hierarchy



17.5.3.5.108 UModelAPI - IUMLLiteralString

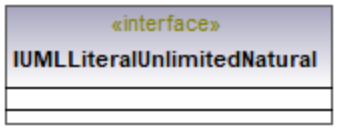
Interface **IUMLLiteralString**

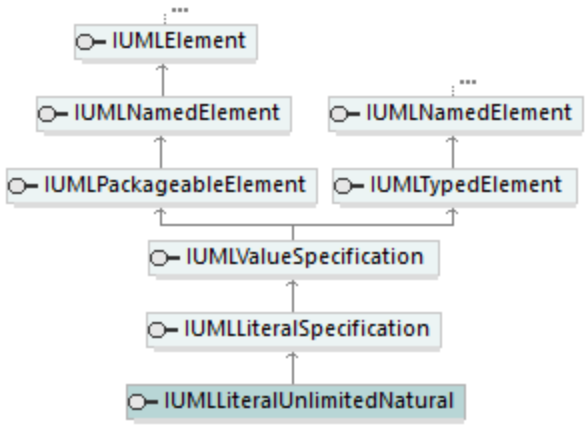
diagram	
hierarchy	
typedElements	Interface **IUMLConstraint** Interface **IUMLDataAll** Interface **IUMLInstanceSpecification** Interface **IUMLParameter** Interface **IUMLProperty** Operation **SetNewSpecificationLiteralString** Operation **SetNewDefaultValueLiteralString** **SetNewSpecificationLiteralString** Operation **SetNewSpecificationLiteralString** Operation **SetNewDefaultValueLiteralString** Operation **SetNewDefaultValueLiteralString**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.109 UModelAPI - IUMLLiteralUnlimitedNatural

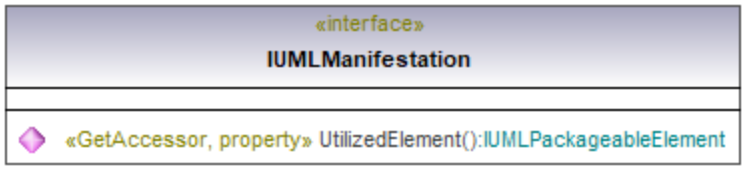
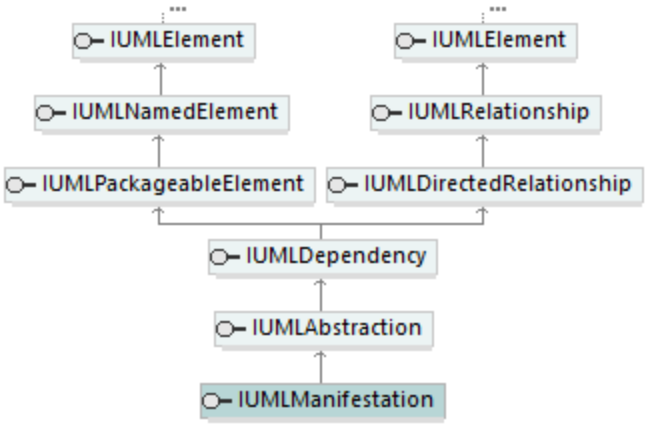
Interface **IUMLLiteralUnlimitedNatural**

diagram	

hierarchy	

17.5.3.5.110 UModelAPI - IUMLManifestation

Interface **IUMLManifestation**

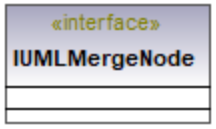
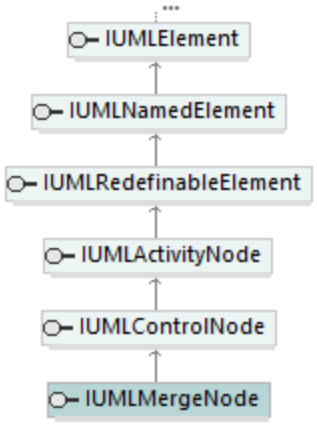
diagram		
hierarchy		
typedElements	Interface **IUMLArtifact** Interface **IUMLDataAll**	Operation **InsertManifestationAt** Operation **InsertManifestationAt**

Operation **IUMLManifestation::UtilizedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackageableElement**			

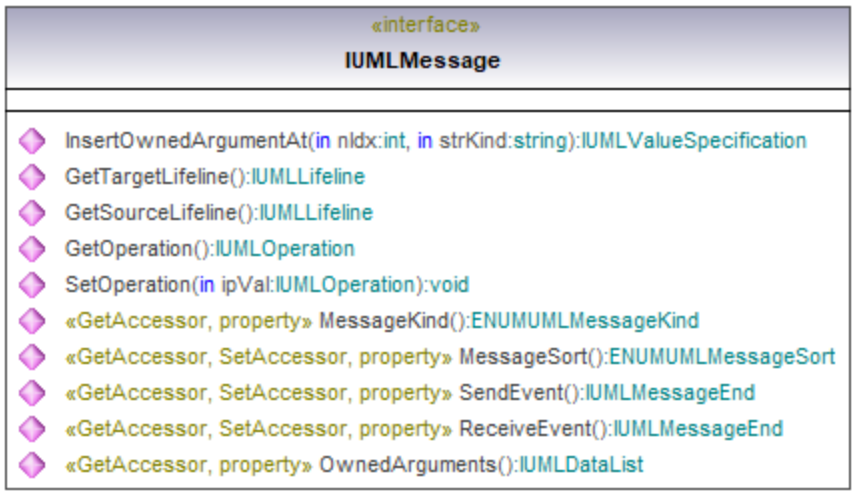
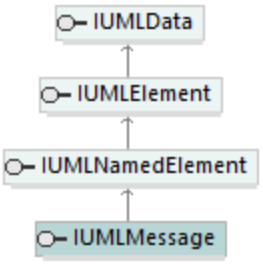
17.5.3.5.111 UModelAPI - IUMLMergeNode

Interface **IUMLMergeNode**

diagram	
hierarchy	

17.5.3.5.112 UModelAPI - IUMLMessage

Interface **IUMLMessage**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInteraction** Interface **IUMLMessageEnd** Operation **InsertMessageAt Message** Operation **InsertMessageAt** Operation **Message**

Operation **IUMLMessage::GetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLMessage::GetSourceLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLMessage::GetTargetLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLMessage::InsertOwnedArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLMessage::MessageKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLMessageKind**			

Operation **IUMLMessage::MessageSort**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLMessageSort**			

Operation **IUMLMessage::OwnedArguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLValueSpecification.					

Operation **IUMLMessage::ReceiveEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessageEnd**			

Operation **IUMLMessage::SendEvent**

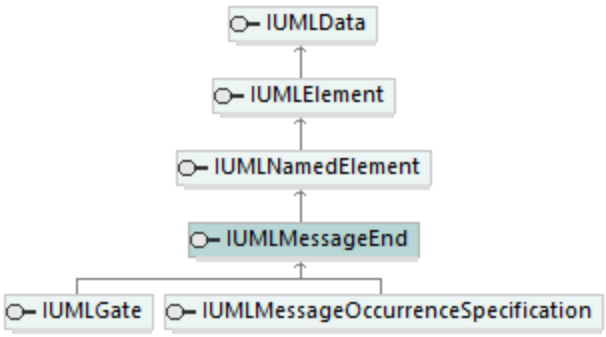
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessageEnd**			

Operation **IUMLMessage::SetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	**IUMLOperation**			
	return	**return**	**void**			

17.5.3.5.113 UModelAPI - IUMLMessageEnd

Interface **IUMLMessageEnd**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLMessage**
	Operation **ReceiveEvent** **SendEvent** Operation **ReceiveEvent** **SendEvent**

Operation **IUMLMessageEnd::Message**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessage**			

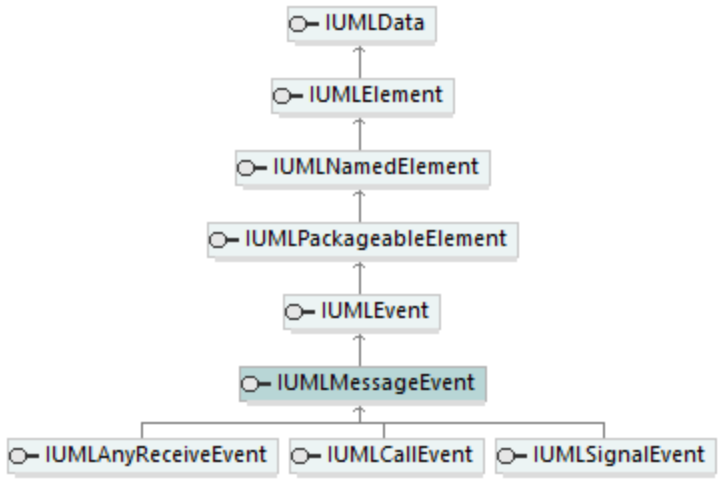
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.114 UModelAPI - IUMLMessageEvent

Interface **IUMLMessageEvent**

diagram	

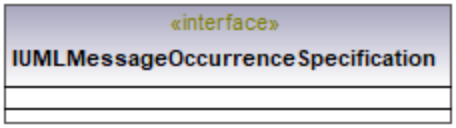
hierarchy



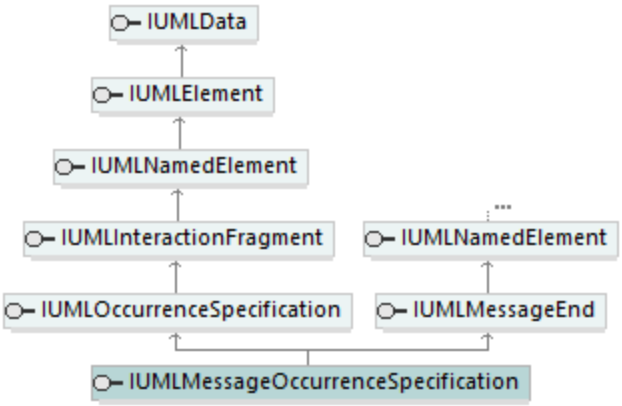
17.5.3.5.115 UModelAPI - IUMLMessageOccurrenceSpecification

Interface **IUMLMessageOccurrenceSpecification**

diagram

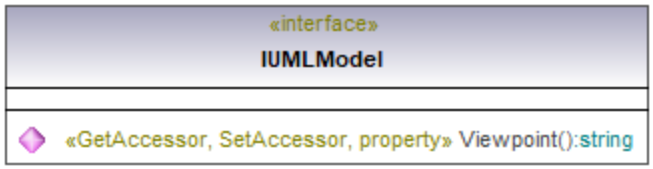
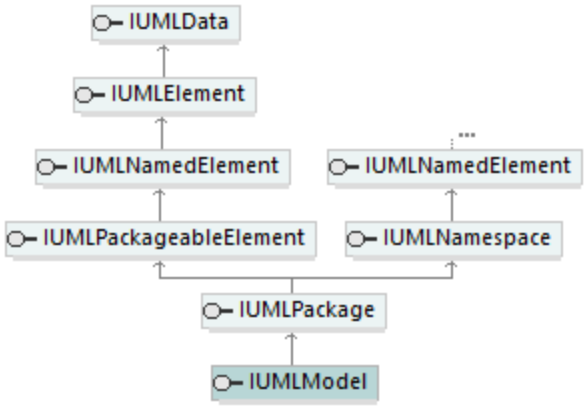


hierarchy



17.5.3.5.116 UModelAPI - IUMLModel

Interface **IUMLModel**

diagram	
hierarchy	

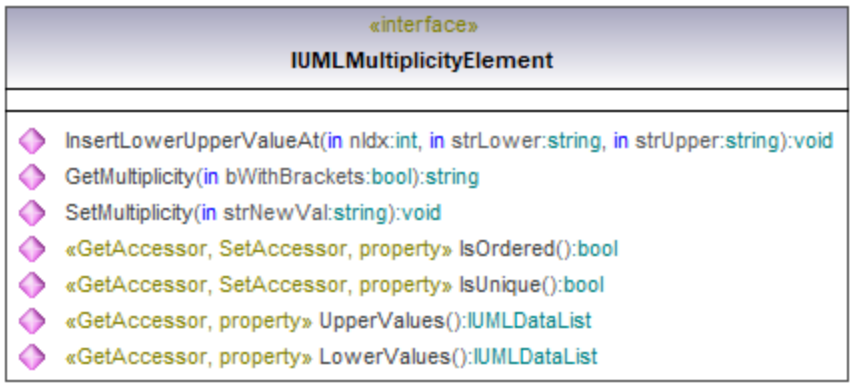
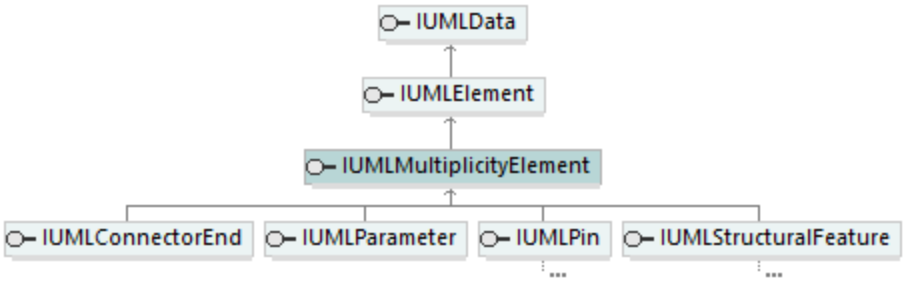
Operation **IUMLModel::Viewpoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.117 UModelAPI - IUMLMultiplicityElement

Interface **IUMLMultiplicityElement**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLValueSpecification** Operation **OwningLower** **OwningUpper** Operation **OwningLower** **OwningUpper**

Operation **IUMLMultiplicityElement::GetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	bWithBrackets	**in**	**bool**			
	return	**return**	**string**			

Operation **IUMLMultiplicityElement::InsertLowerUpperValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strLower	**in**	**string**			
	strUpper	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLMultiplicityElement::IsOrdered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLMultiplicityElement::IsUnique**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLMultiplicityElement::LowerValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLValueSpecification.					

Operation **IUMLMultiplicityElement::SetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLMultiplicityElement::UpperValues**

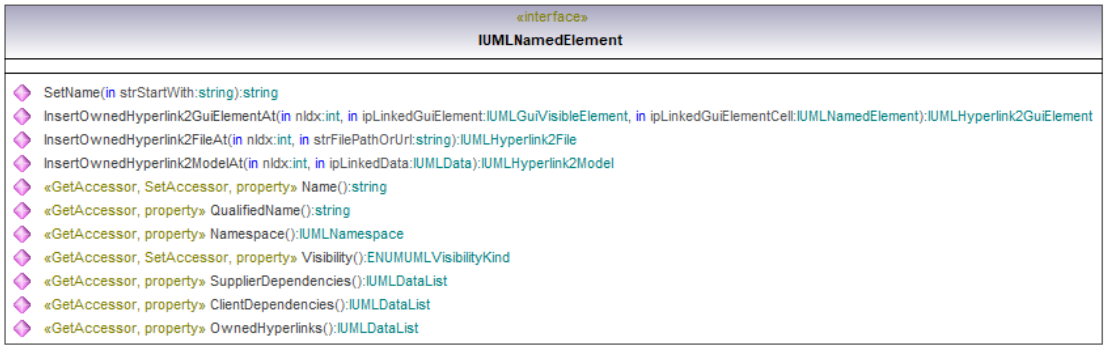
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLValueSpecification.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

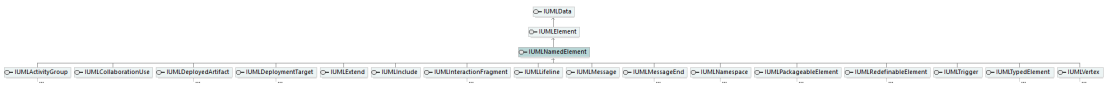
17.5.3.5.118 UModelAPI - IUMLNamedElement

Interface **IUMLNamedElement**

diagram



hierarchy



typedElements

Interface **IUMLCommentTextHyperlink**
Interface **IUMLDataAll**
Operation **SetHyperlinkGuiElementAddress**
Operation **FindOwnedMemberWithQualifiedName** **InsertInformationSourceAt**

		InsertInformationTargetAt InsertOwnedHyperlink2GuiElementAt LinkedGuiElementCell SetHyperlinkGuiElementAddress TimeObservationEvent
	Interface IUMLGuiTextHyperlink	Operation SetHyperlinkGuiElementAddress
	Interface IUMLHyperlink2GuiElement	Operation LinkedGuiElementCell
	Interface IUMLInformationFlow	Operation InsertInformationSourceAt InsertInformationTargetAt
	Interface IUMLNamedElement	Operation InsertOwnedHyperlink2GuiElementAt
	Interface IUMLNamespace	Operation FindOwnedMemberWithQualifiedName
	Interface IUMLTimeObservation	Operation TimeObservationEvent

Operation **IUMLNamedElement::ClientDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			
documentation	A list of elements of type IUMLDependency.					

Operation **IUMLNamedElement::InsertOwnedHyperlink2FileAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strFilePathOrUrl	**in**	**string**			
	return	**return**	IUMLHyperlink2File			

Operation **IUMLNamedElement::InsertOwnedHyperlink2GuiElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedGuiElement	**in**	IUMLGuiVisibleElement			
	ipLinkedGuiElementCell	**in**	IUMLNamedElement			
	return	**return**	IUMLHyperlink2GuiElement			

Operation **IUMLNamedElement::InsertOwnedHyperlink2ModelAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedData	**in**	IUMLData			
	return	**return**	IUMLHyperlink2Model			

Operation **IUMLNamedElement::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLNamedElement::Namespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLNamedElement::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLHyperlink.					

Operation **IUMLNamedElement::QualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLNamedElement::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			
documentation	This function w ill find and set a unique name (starting w ith 'strStartWith') that the element is distinguishable in its parent namespace.					

Operation **IUMLNamedElement::SupplierDependencies**

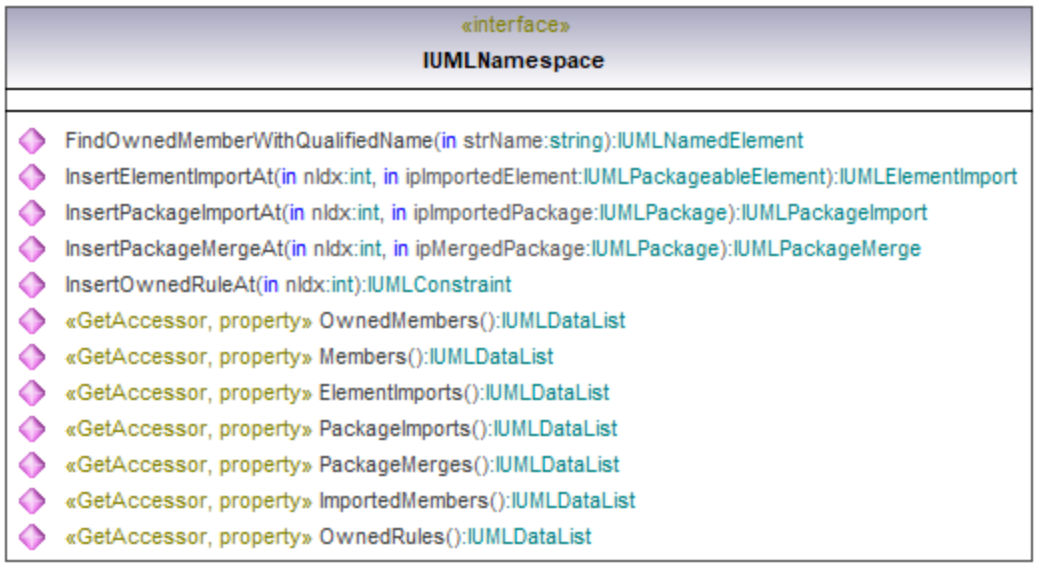
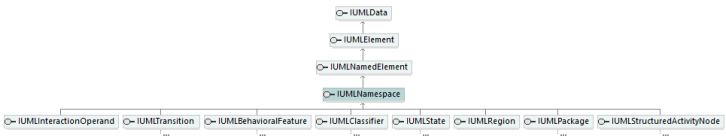
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLDependency.					

Operation **IUMLNamedElement::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

17.5.3.5.119 UModelAPI - IUMLNamespace

Interface **IUMLNamespace**

diagram	
hierarchy	
typedElements	Interface **IUMLConstraint** Interface **IUMLDataAll** Interface **IUMLElementImport** Interface **IUMLNamedElement** Interface **IUMLPackageImport** Operation **Context** Operation **Context ImportingNamespace Namespace** Operation **ImportingNamespace** Operation **Namespace** Operation **ImportingNamespace**

Operation **IUMLNamespace::ElementImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElementImport.					

Operation **IUMLNamespace::FindOwnedMemberWithQualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
	strName	**in**	**string**			
	return	**return**	**IUMLNamedElement**			

Operation **IUMLNamespace::ImportedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

documentation	A list of elements of type IUMLPackageableElement.

Operation **IUMLNamespace::InsertElementImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedElement	**in**	**IUMLPackageableElement**			
	return	**return**	**IUMLElementImport**			

Operation **IUMLNamespace::InsertOwnedRuleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLNamespace::InsertPackageImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedPackage	**in**	**IUMLPackage**			
	return	**return**	**IUMLPackageImport**			

Operation **IUMLNamespace::InsertPackageMergeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipMergedPackage	**in**	**IUMLPackage**			
	return	**return**	**IUMLPackageMerge**			

Operation **IUMLNamespace::Members**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

Operation **IUMLNamespace::OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

Operation **IUMLNamespace::OwnedRules**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

documentation	A list of elements of type IUMLConstraint.

Operation **IUMLNamespace::PackageImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackageImport.					

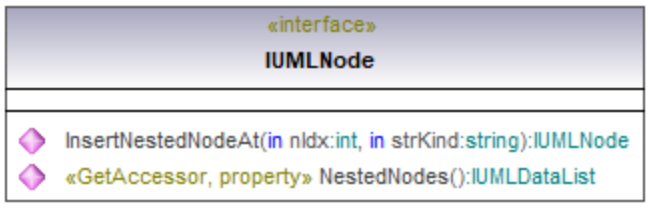
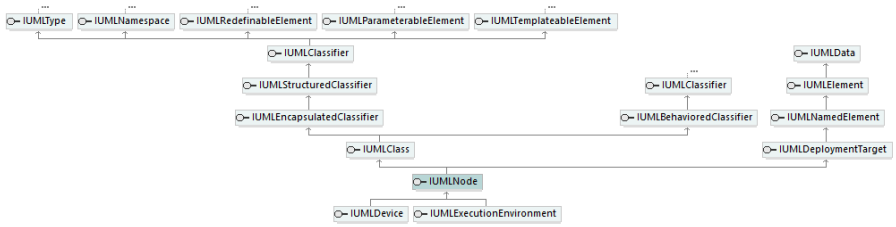
Operation **IUMLNamespace::PackageMerges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackageMerge.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.120 UModelAPI - IUMLNode

Interface **IUMLNode**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNode**		Operation **InsertNestedNodeAt** Operation **InsertNestedNodeAt**			

Operation **IUMLNode::InsertNestedNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLNode**			

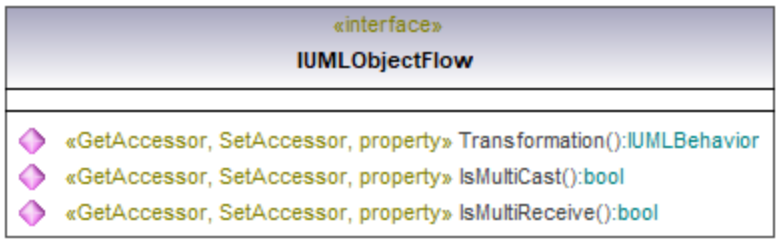
Operation **IUMLNode::NestedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNode.					

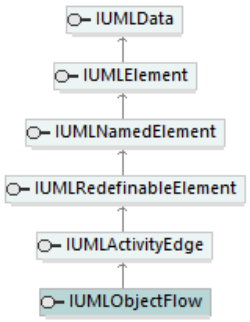
17.5.3.5.121 UModelAPI - IUMLObjectFlow

Interface **IUMLObjectFlow**

diagram



hierarchy



Operation **IUMLObjectFlow::IsMultiCast**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLObjectFlow::IsMultiReceive**

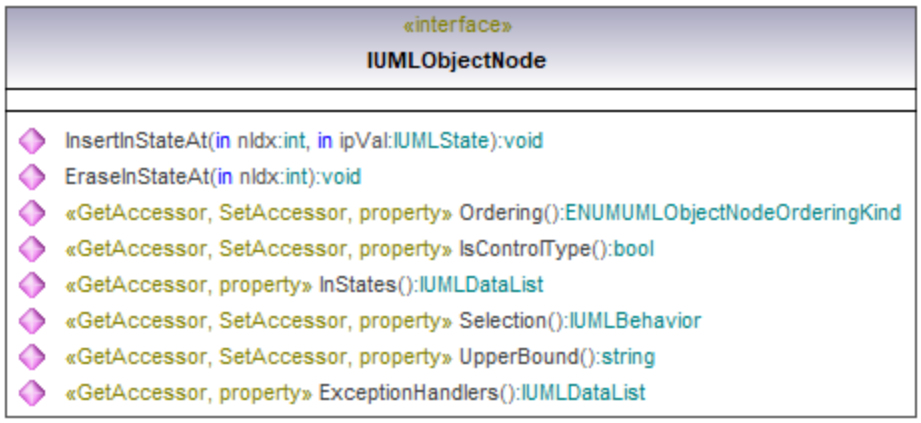
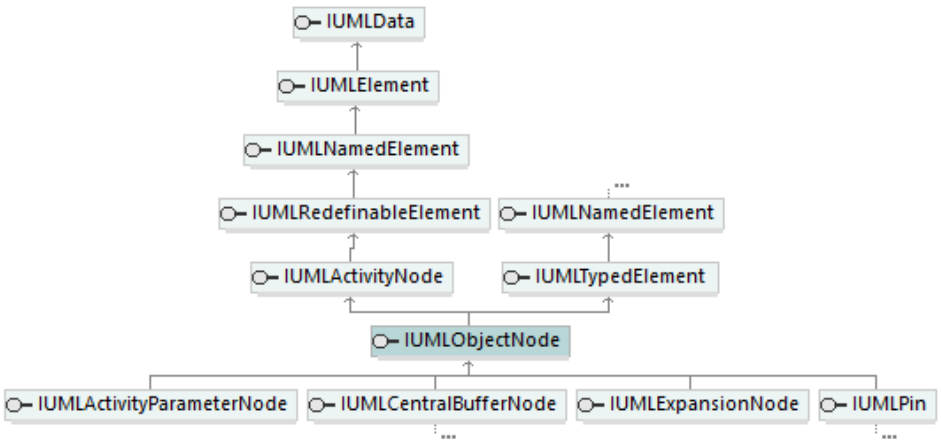
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLObjectFlow::Transformation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

17.5.3.5.122 UModelAPI - IUMLObjectNode

Interface **IUMLObjectNode**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLExceptionHandler** Operation **ExceptionInput** Operation **ExceptionInput**

Operation **IUMLObjectNode::EraseInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLObjectNode::ExceptionHandlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

documentation	A list of elements of type IUMLExceptionHandler.

Operation **IUMLObjectNode::InsertInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLState**			
	return	**return**	**void**			

Operation **IUMLObjectNode::InStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLState.					

Operation **IUMLObjectNode::IsControlType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLObjectNode::Ordering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLObjectNodeOrderingKind**			

Operation **IUMLObjectNode::Selection**

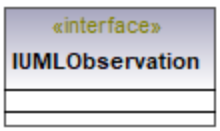
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

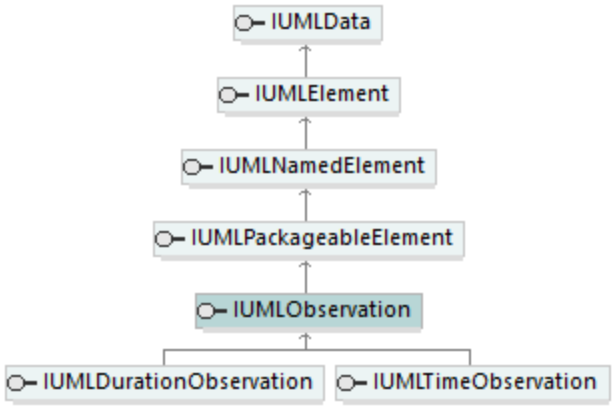
Operation **IUMLObjectNode::UpperBound**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.3.5.123 UModelAPI - IUMLObservation

Interface **IUMLObservation**

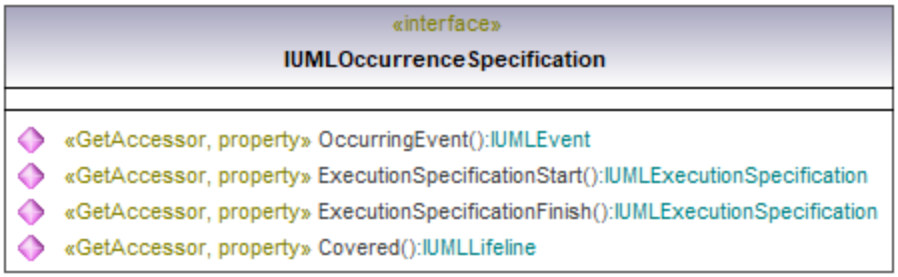
diagram	

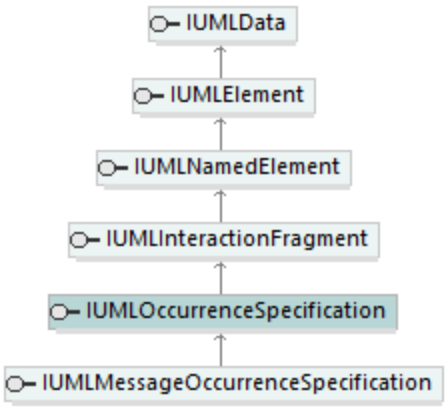
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLDuration** Interface **IUMLTimeExpression** Operation **InsertObservationAt** Operation **InsertObservationAt** Operation **InsertObservationAt**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.124 UModelAPI - IUMLOccurrenceSpecification

Interface **IUMLOccurrenceSpecification**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Operation **Finish** **Start** Interface **IUMLExecutionSpecification** Operation **Finish** **Start**

Operation **IUMLOccurrenceSpecification::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLOccurrenceSpecification::ExecutionSpecificationFinish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutionSpecification**			

Operation **IUMLOccurrenceSpecification::ExecutionSpecificationStart**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutionSpecification**			

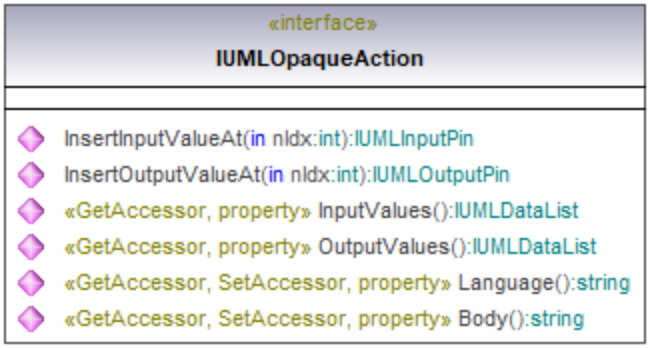
Operation **IUMLOccurrenceSpecification::OccurringEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEvent**			

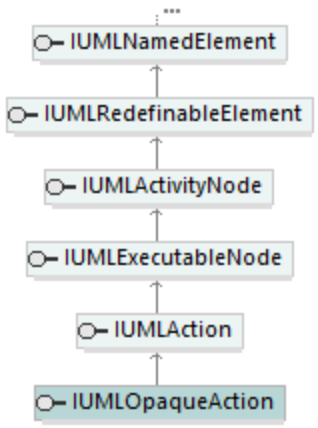
17.5.3.5.125 UModelAPI - IUMLOpaqueAction

Interface **IUMLOpaqueAction**

diagram



hierarchy



Operation **IUMLOpaqueAction::Body**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLOpaqueAction::InputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLOpaqueAction::InsertInputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin**			

Operation **IUMLOpaqueAction::InsertOutputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLOpaqueAction::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

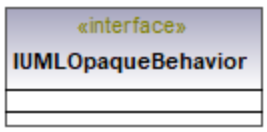
Operation **IUMLOpaqueAction::OutputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

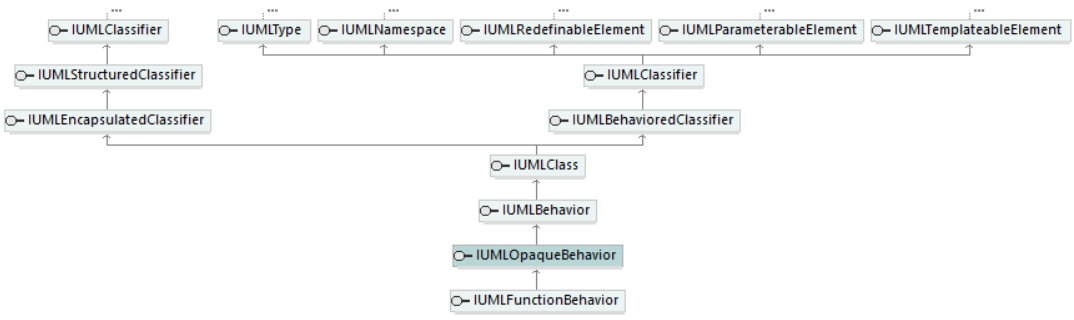
17.5.3.5.126 UModelAPI - IUMLOpaqueBehavior

Interface **IUMLOpaqueBehavior**

diagram



hierarchy



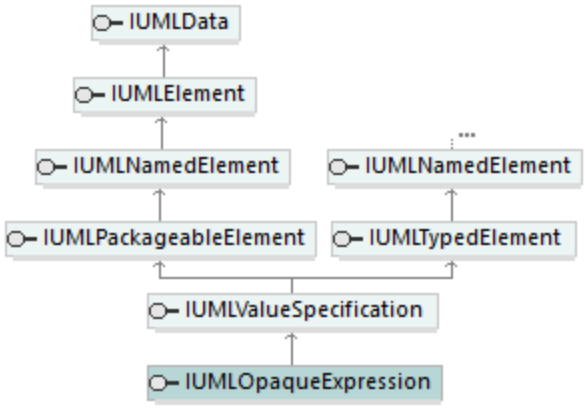
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.127 UModelAPI - IUMLOpaqueExpression

Interface **IUMLOpaqueExpression**

diagram

hierarchy



typedElements

Interface **IUMLAbstraction**
Interface **IUMLDataAll**
Operation **Mapping SetNewMapping**
Operation **Mapping SetNewMapping**

Operation **IUMLOpaqueExpression::Body**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLOpaqueExpression::Language**

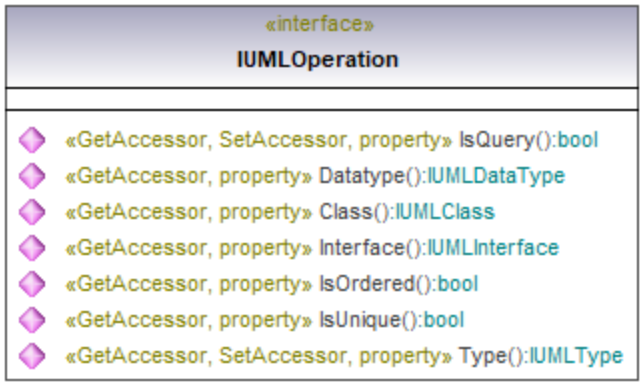
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

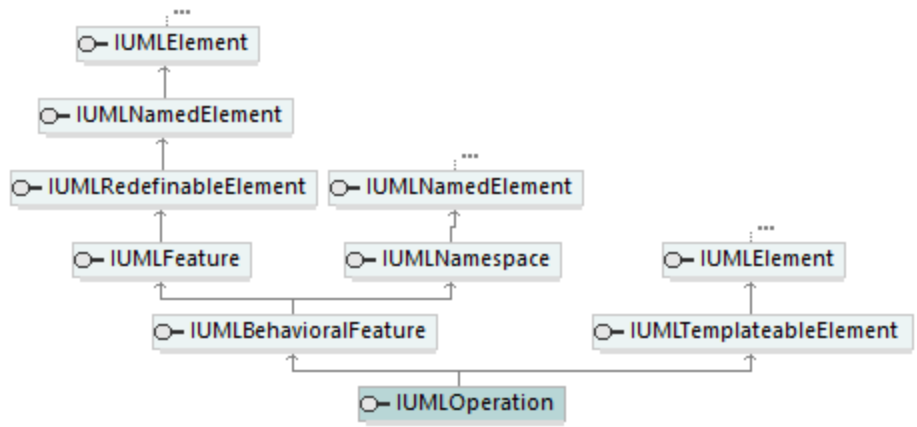
17.5.3.5.128 UModelAPI - IUMLOperation

Interface **IUMLOperation**

diagram



hierarchy



typedElements

Interface **IDocument**
Interface **IUMLArtifact**
Interface **IUMLCallEvent**
Interface **IUMLCallOperationAction**
Interface **IUMLClass**
Interface **IUMLDataAll**

Interface **IUMLDataType**
Interface **IUMLGuiSequenceDiagram**
Interface **IUMLInterface**
Interface **IUMLMessage**
Interface **IUMLParameter**

Operation **GenerateSequenceDiagram**
Operation **InsertOwnedOperationAt**
Operation **Operation**
Operation **CallOperation**
Operation **InsertOwnedOperationAt**
Operation **CallOperation** **CodeOperation** **GetOperation** **InsertOwnedOperationAt** **Operation** **SetOperation**
Operation **InsertOwnedOperationAt**
Operation **CodeOperation**

Operation **InsertOwnedOperationAt**
Operation **GetOperation** **SetOperation**
Operation **Operation**

Operation **IUMLOperation::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass**			

Operation **IUMLOperation::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataType**			

Operation **IUMLOperation::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLOperation::IsOrdered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLOperation::IsQuery**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLOperation::IsUnique**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

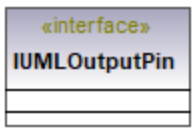
Operation **IUMLOperation::Type**

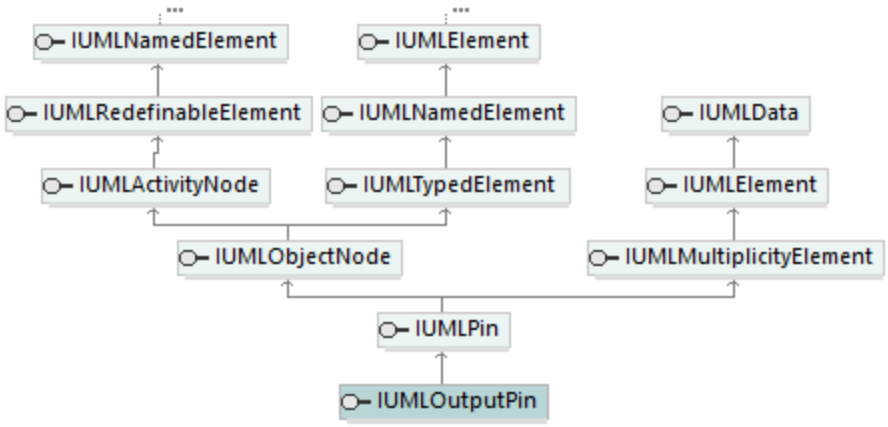
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLType**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.129 UModelAPI - IUMLOutputPin

Interface **IUMLOutputPin**

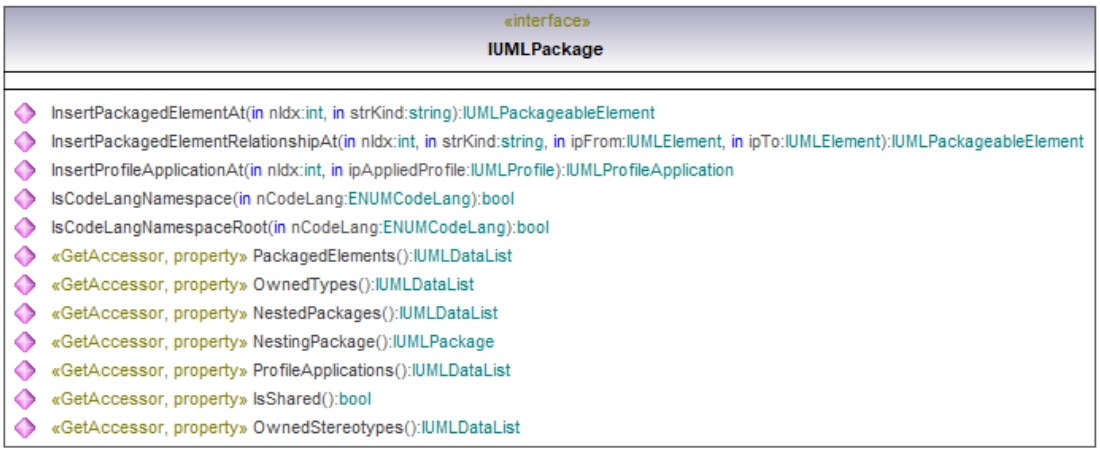
diagram	

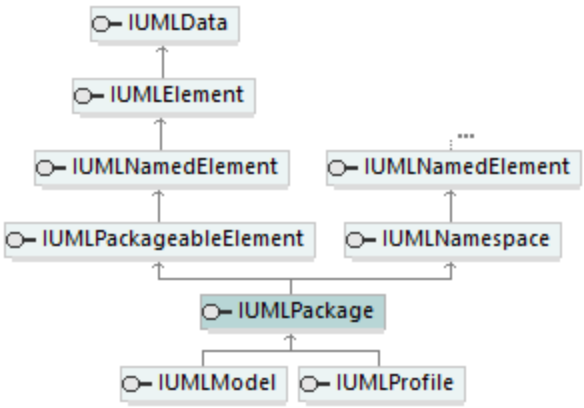
hierarchy	
typedElements	Interface **IUMLAcceptEventAction** Interface **IUMLCallAction** Interface **IUMLDataAll** Interface **IUMLOpaqueAction** Interface **IUMLValueSpecificationAction** Operation **InsertEventActionResultAt** Operation **InsertResultAt** Operation **InsertEventActionResultAt** **InsertOutputValueAt** **InsertResultAt** **Result** Operation **InsertOutputValueAt** Operation **Result**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.130 UModelAPI - IUMLPackage

Interface **IUMLPackage**

diagram	

hierarchy	
typedElements	Interface **IDocument** Interface **IImportSourceDlg** Interface **IModelTransformationDlg** Interface **IUMLDataAll** Interface **IUMLNamespace** Interface **IUMLPackage** Interface **IUMLPackageableElement** Interface **IUMLPackageImport** Interface **IUMLPackageMerge** Interface **IUMLProfileApplication** Interface **IUMLType** Operation **RootPackage** Operation **ImportTarget** Operation **SourcePackage** **TargetPackage** Operation **ApplyingPackage** **ImportedPackage** **InsertPackageImportAt** **InsertPackageMergeAt** **MergedPackage** **NestingPackage** **OwningPackage** **Package** **ReceivingPackage** Operation **InsertPackageImportAt** **InsertPackageMergeAt** Operation **NestingPackage** Operation **OwningPackage** Operation **ImportedPackage** Operation **MergedPackage** **ReceivingPackage** Operation **ApplyingPackage** Operation **Package**

Operation **IUMLPackage::InsertPackagedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLPackage::InsertPackagedElementRelationshipAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLElement**			
	ipTo	**in**	**IUMLElement**			
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLPackage::InsertProfileApplicationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipAppliedProfile	**in**	**IUMLProfile**			
	return	**return**	**IUMLProfileApplication**			

Operation **IUMLPackage::IsCodeLangNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	**ENUMCodeLang**			
	return	**return**	**bool**			

Operation **IUMLPackage::IsCodeLangNamespaceRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	**ENUMCodeLang**			
	return	**return**	**bool**			

Operation **IUMLPackage::IsShared**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLPackage::NestedPackages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackage.					

Operation **IUMLPackage::NestingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLPackage::OwnedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLPackage::OwnedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLType.					

Operation **IUMLPackage::PackagedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackageableElement.					

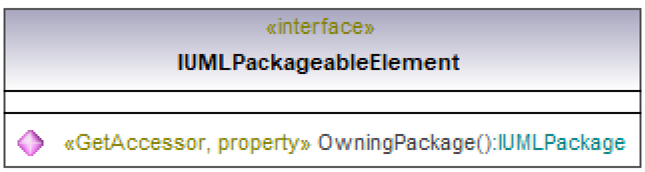
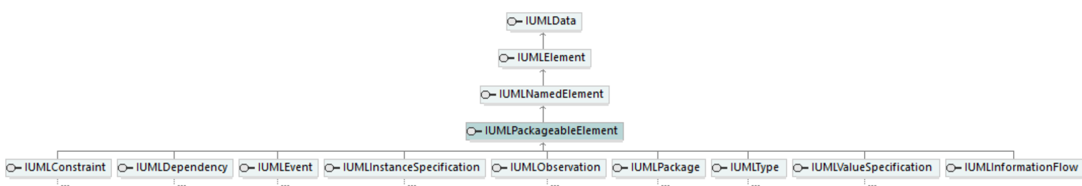
Operation **IUMLPackage::ProfileApplications**

parameter	name **return**	direction **return**	type [IUMLDataList]	type modifier	multiplicity	default
documentation	A list of elements of type IUMLProfileApplication.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.131 UModelAPI - IUMLPackageableElement

Interface **IUMLPackageableElement**

diagram	
hierarchy	
typedElements	Interface **IUMLArtifact** — Operation **InsertManifestationAt** Interface **IUMLDataAll** — Operation **ImportedElement** **InsertElementImportAt** **InsertManifestationAt** **InsertPackagedElementAt** **InsertPackagedElementRelationshipAt** **UtilizedElement** Interface **IUMLElementImport** — Operation **ImportedElement** Interface **IUMLManifestation** — Operation **UtilizedElement** Interface **IUMLNamespace** — Operation **InsertElementImportAt** Interface **IUMLPackage** — Operation **InsertPackagedElementAt** **InsertPackagedElementRelationshipAt**

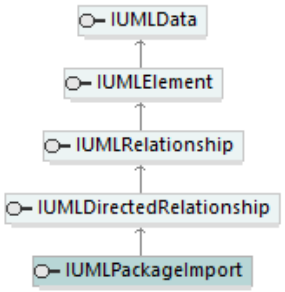
Operation **IUMLPackageableElement::OwningPackage**

parameter	name **return**	direction **return**	type **IUMLPackage**	type modifier	multiplicity	default

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.132 UModelAPI - IUMLPackageImport

Interface **IUMLPackageImport**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamespace** Operation **InsertPackageImportAt** Operation **InsertPackageImportAt**

Operation **IUMLPackageImport::ImportedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLPackageImport::ImportingNamespace**

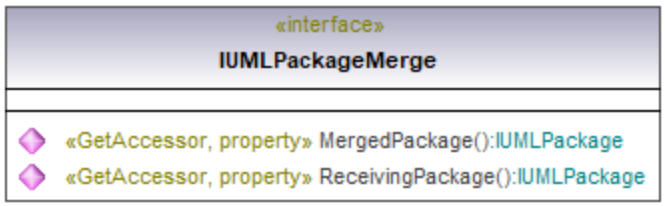
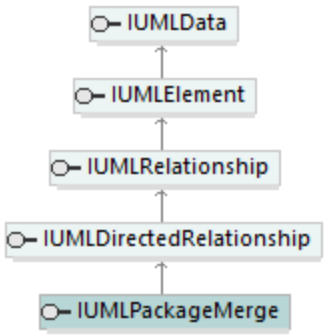
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLPackageImport::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

17.5.3.5.133 UModelAPI - IUMLPackageMerge

Interface **IUMLPackageMerge**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamespace** Operation **InsertPackageMergeAt** Operation **InsertPackageMergeAt**

Operation **IUMLPackageMerge::MergedPackage**

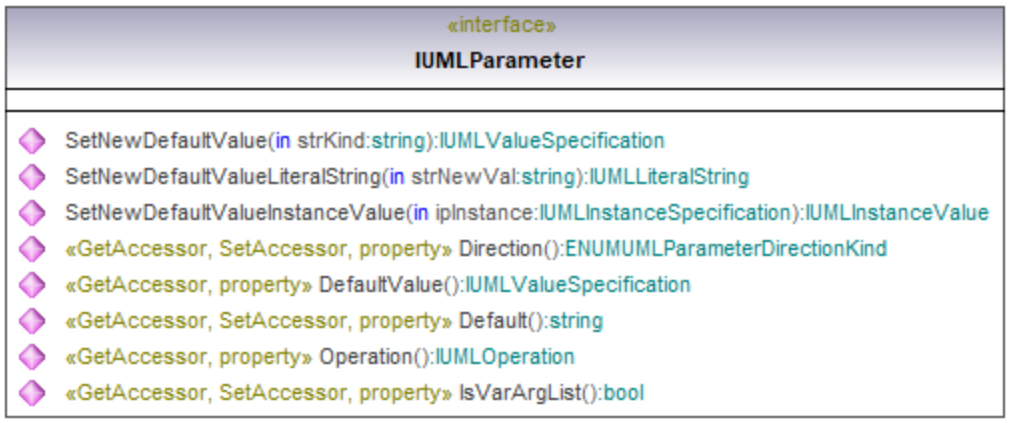
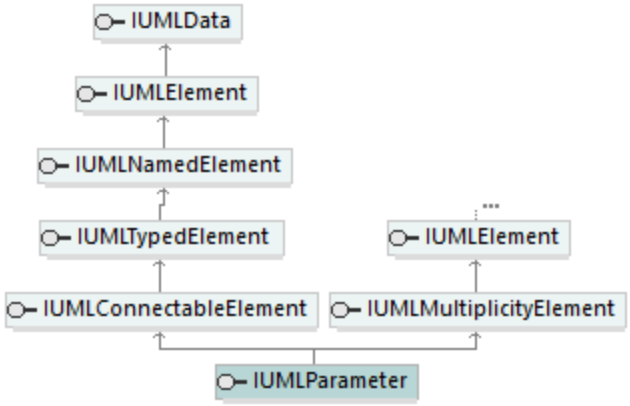
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLPackageMerge::ReceivingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

17.5.3.5.134 UModelAPI - IUMLParameter

Interface **IUMLParameter**

diagram	
hierarchy	
typedElements	Interface **IUMLActivityParameterNode** Interface **IUMLBehavior** Interface **IUMLBehavioralFeature** Interface **IUMLDataAll** Interface **IUMLValueSpecification** Operation **Parameter** Operation **InsertOwnedParameterAt** Operation **InsertOwnedParameterAt** Operation **InsertOwnedParameterAt** **OwningParameter** **Parameter** Operation **OwningParameter**

Operation **IUMLParameter::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLParameter::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLParameter::Direction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLParameterDirectionKind**			

Operation **IUMLParameter::IsVarArgList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLParameter::Operation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLParameter::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLParameter::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLParameter::SetNewDefaultValueLiteralString**

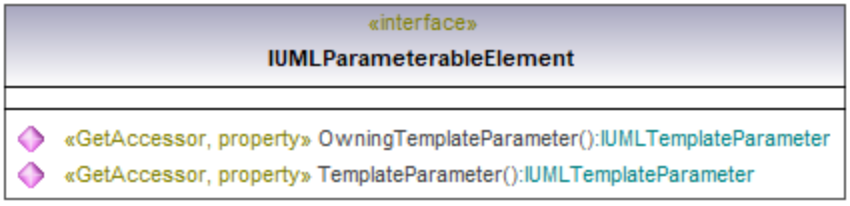
parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

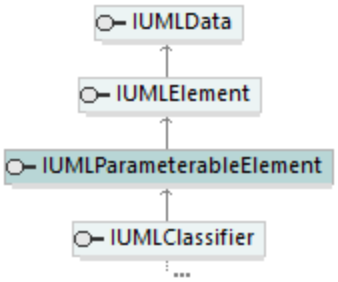
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.135 UModelAPI - IUMLParameterableElement

Interface **IUMLParameterableElement**

diagram	



hierarchy	
typedElements	Interface **IUMLDataAll** Operation **Actual** **InsertParameterSubstitutionAt** **OwnedActual** **OwnedParameteredElement** **ParameteredElement** **SetNewOwnedParameteredElement** Interface **IUMLTemplateBinding** Operation **InsertParameterSubstitutionAt** Interface **IUMLTemplateParameter** Operation **OwnedParameteredElement** **ParameteredElement** **SetNewOwnedParameteredElement** Interface **IUMLTemplateParameterSubstitution** Operation **Actual** **OwnedActual**

Operation **IUMLParameterableElement::OwningTemplateParameter**

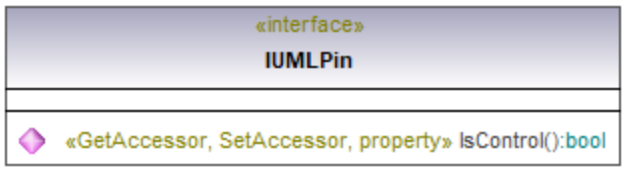
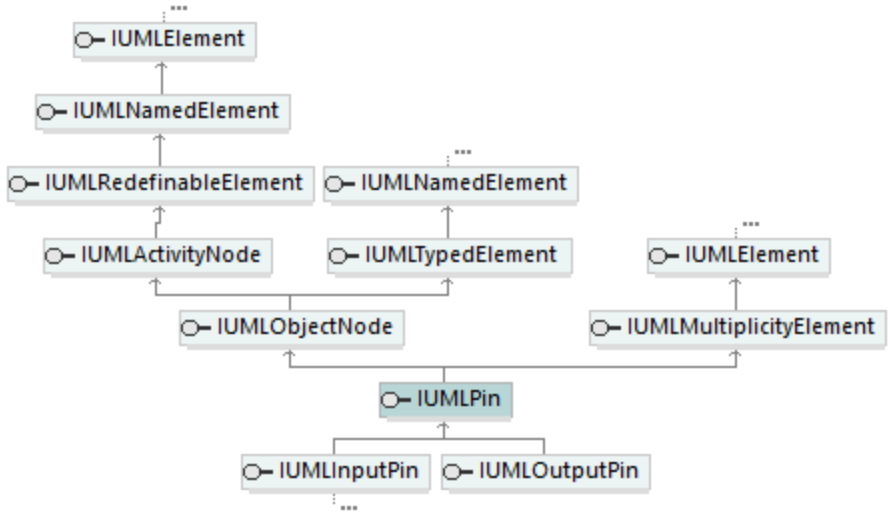
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

Operation **IUMLParameterableElement::TemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

17.5.3.5.136 UModelAPI - IUMLPin

Interface **IUMLPin**

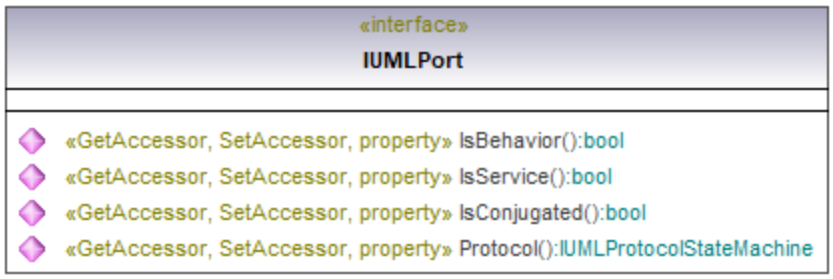
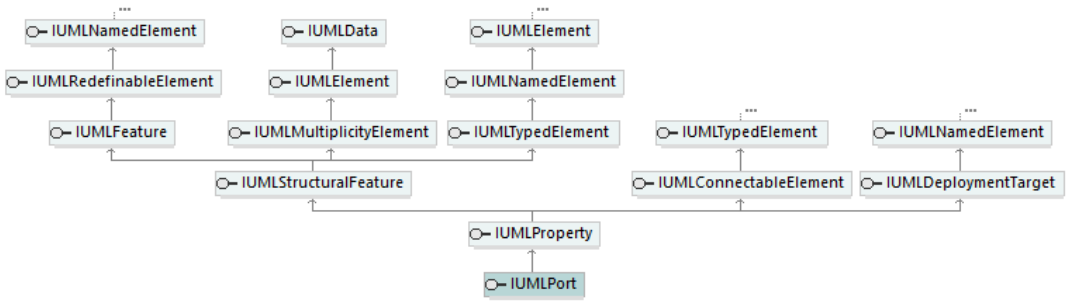
diagram	
hierarchy	

Operation **IUMLPin::IsControl**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.3.5.137 UModelAPI - IUMLPort

Interface **IUMLPort**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLStructuredClassifier** Operation **InsertOwnedPortAt** Operation **InsertOwnedPortAt**

Operation **IUMLPort::IsBehavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLPort::IsConjugated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLPort::IsService**

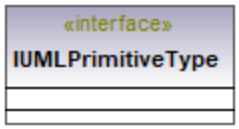
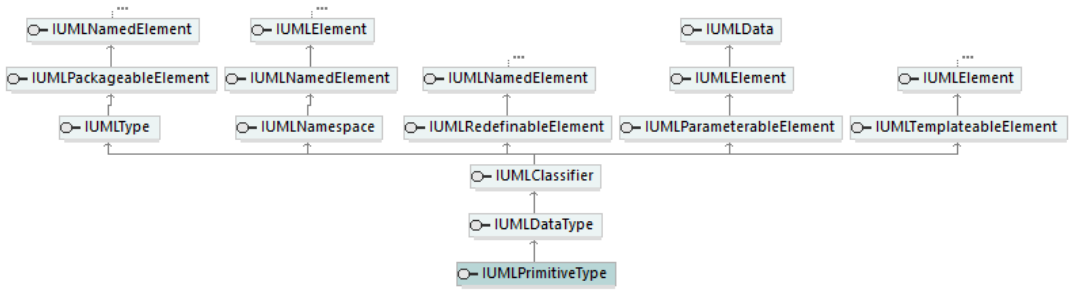
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLPort::Protocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine**			

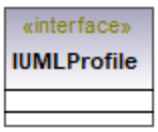
17.5.3.5.138 UModelAPI - IUMLPrimitiveType

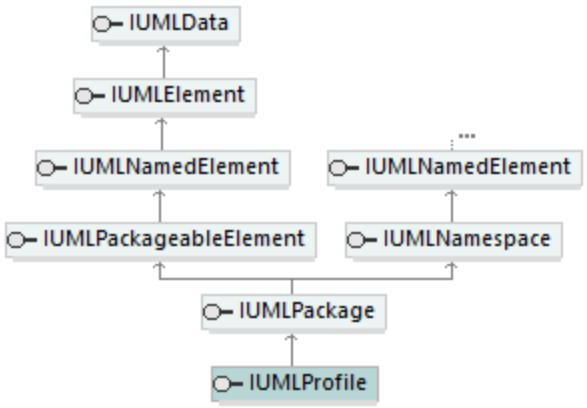
Interface **IUMLPrimitiveType**

diagram	
hierarchy	

17.5.3.5.139 UModelAPI - IUMLProfile

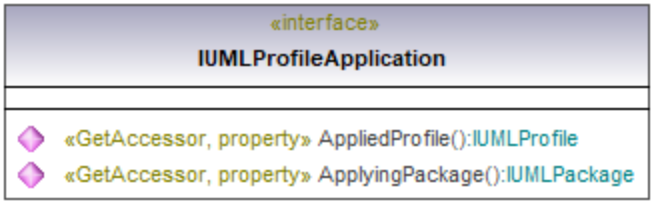
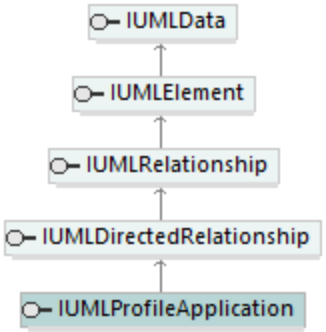
Interface **IUMLProfile**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLPackage** Interface **IUMLProfileApplication** Operation **AppliedProfile** **InsertProfileApplicationAt** Operation **InsertProfileApplicationAt** Operation **AppliedProfile**

17.5.3.5.140 UModelAPI - IUMLProfileApplication

Interface **IUMLProfileApplication**

diagram	
hierarchy	

typedElements	Interface **IUMLDataAll** Interface **IUMLPackage**	Operation **InsertProfileApplicationAt** Operation **InsertProfileApplicationAt**

Operation **IUMLProfileApplication::AppliedProfile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProfile**			

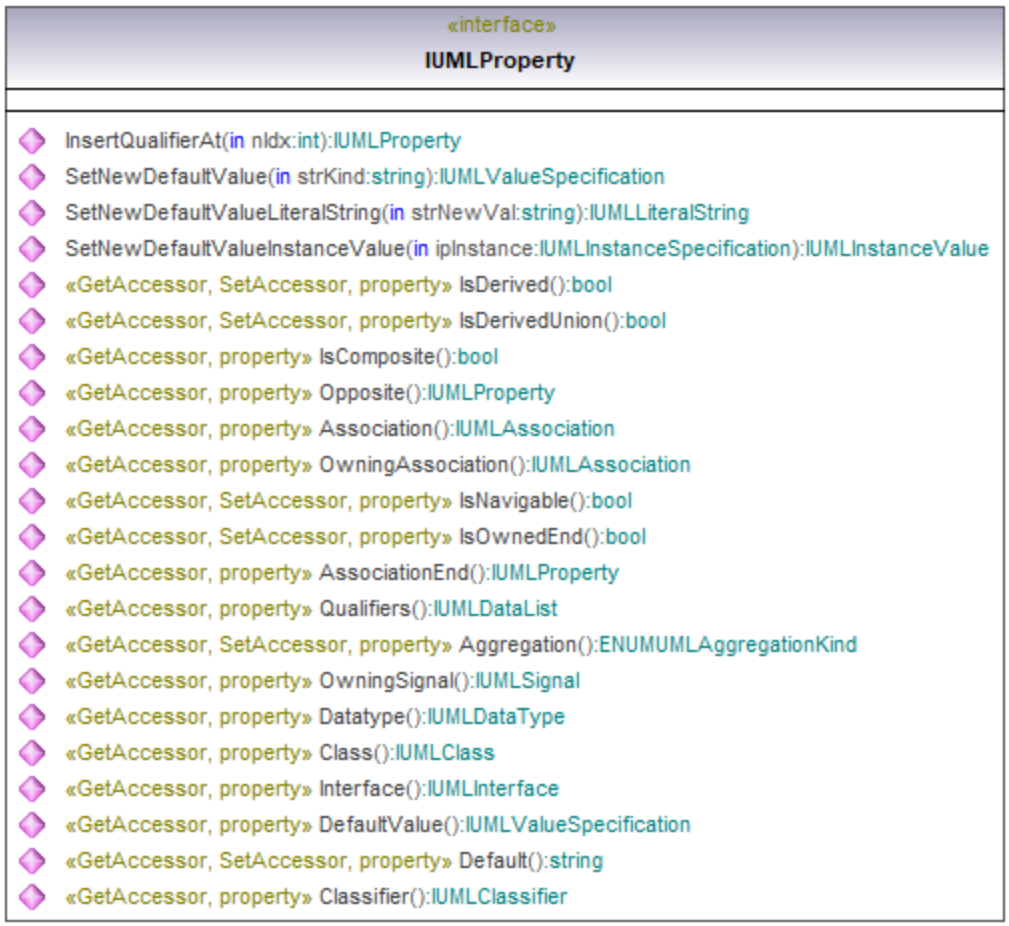
Operation **IUMLProfileApplication::ApplyingPackage**

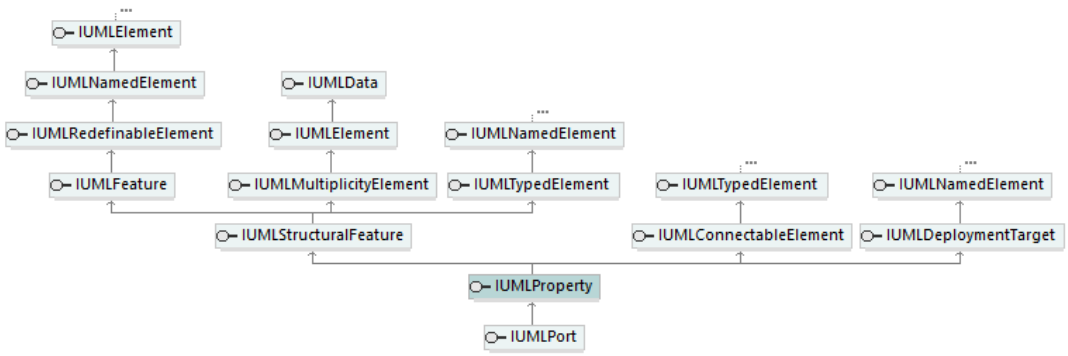
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

17.5.3.5.141 UModelAPI - IUMLProperty

Interface **IUMLProperty**

diagram



hierarchy	
typedElements	Interface **IUMLArtifact** Interface **IUMLDataAll** Interface **IUMLDataType** Interface **IUMLInterface** Interface **IUMLProperty** Interface **IUMLSignal** Interface **IUMLStructuredClassifier** Interface **IUMLValueSpecification**
	Operation **InsertOwnedAttributeAt** Operation **AssociationEnd InsertOwnedAttributeAt InsertQualifierAt Opposite OwningProperty** Operation **InsertOwnedAttributeAt** Operation **InsertOwnedAttributeAt** Operation **AssociationEnd InsertQualifierAt Opposite** Operation **InsertOwnedAttributeAt** Operation **InsertOwnedAttributeAt** Operation **OwningProperty**

Operation **IUMLProperty::Aggregation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLAggregationKind**			

Operation **IUMLProperty::Association**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

Operation **IUMLProperty::AssociationEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLProperty::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass**			

Operation **IUMLProperty::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLProperty::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataType**			

Operation **IUMLProperty::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLProperty::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLProperty::InsertQualifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLProperty::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLProperty::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLProperty::IsDerived**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLProperty::IsDerivedUnion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLProperty::IsNavigable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLProperty::IsOwnedEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLProperty::Opposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLProperty::OwningAssociation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

Operation **IUMLProperty::OwningSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

Operation **IUMLProperty::Qualifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLProperty::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLProperty::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLProperty::SetNewDefaultValueLiteralString**

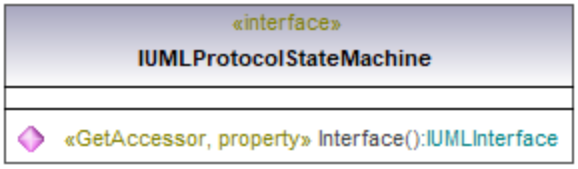
parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

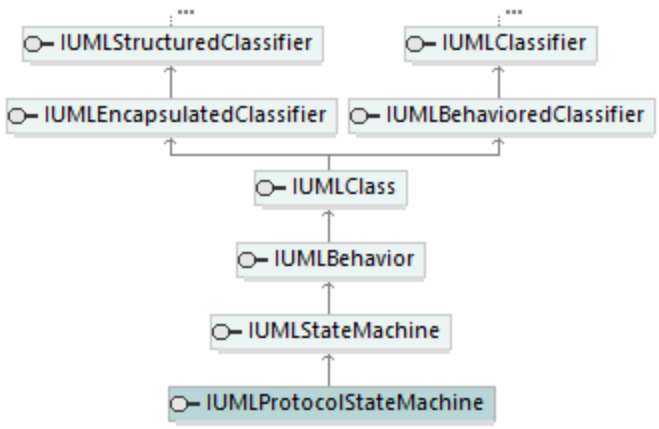
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.142 UModelAPI - IUMLProtocolStateMachine

Interface **IUMLProtocolStateMachine**

diagram	



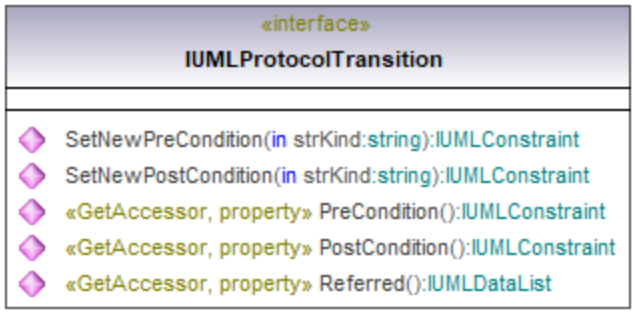
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInterface** Interface **IUMLPort** Operation **Protocol** **SetNewProtocol** Operation **Protocol** **SetNewProtocol** Operation **Protocol**

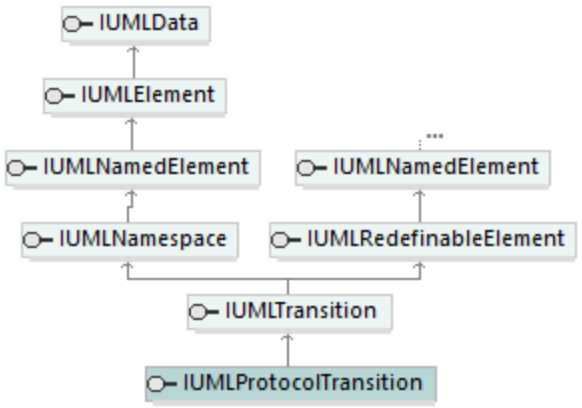
Operation **IUMLProtocolStateMachine::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

17.5.3.5.143 UModelAPI - IUMLProtocolTransition

Interface **IUMLProtocolTransition**

diagram	

hierarchy	
typedElements	Interface **IUMLConstraint** Interface **IUMLDataAll** Operation **OwningTransition** Operation **OwningTransition**

Operation **IUMLProtocolTransition::PostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLProtocolTransition::PreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLProtocolTransition::Referred**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLProtocolTransition::SetNewPostCondition**

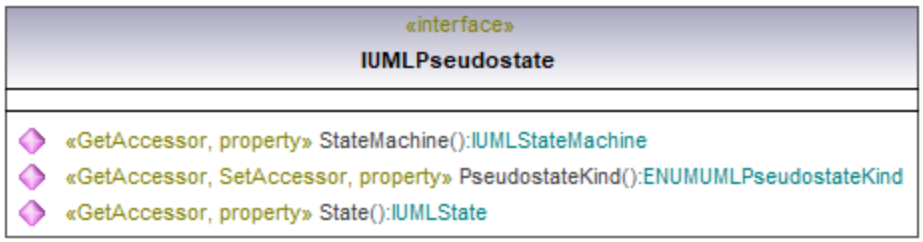
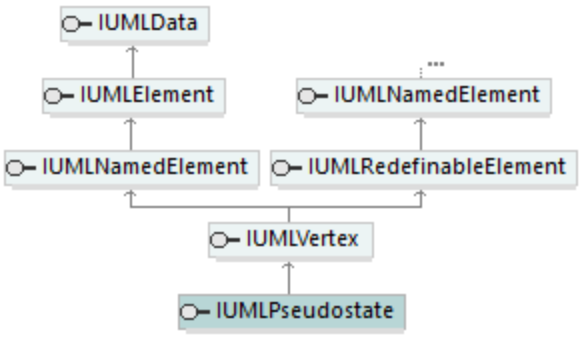
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLProtocolTransition::SetNewPreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

17.5.3.5.144 UModelAPI - IUMLPseudostate

Interface **IUMLPseudostate**

diagram	
hierarchy	
typedElements	Interface **IUMLConnectionPointReference** Interface **IUMLDataAll** Interface **IUMLState** Interface **IUMLStateMachine** Operation **InsertEntryAt** **InsertExitAt** Operation **InsertConnectionPointAt** **InsertEntryAt** **InsertExitAt** Operation **InsertConnectionPointAt** Operation **InsertConnectionPointAt**

Operation **IUMLPseudostate::PseudostateKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLPseudostateKind**			

Operation **IUMLPseudostate::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

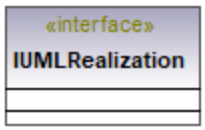
Operation **IUMLPseudostate::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine**			

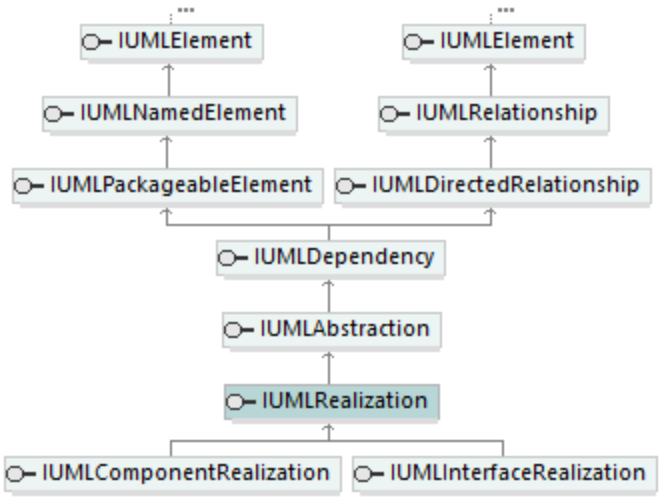
17.5.3.5.145 UModelAPI - IUMLRealization

Interface **IUMLRealization**

diagram



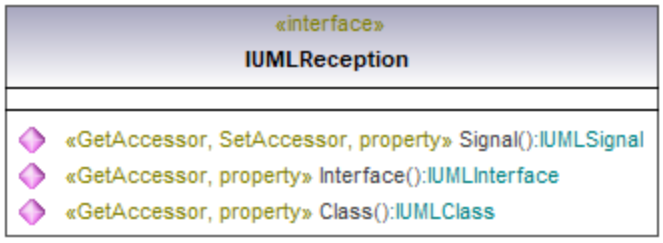
hierarchy

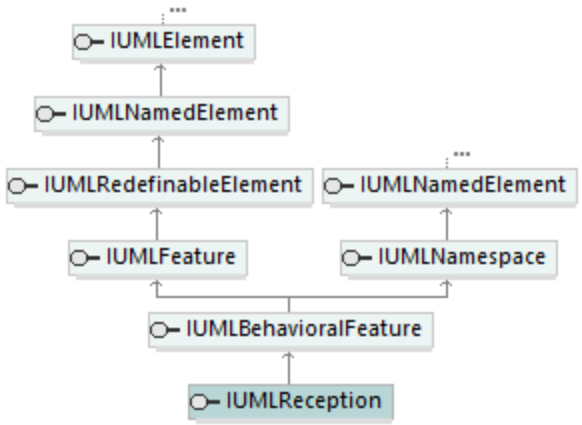


17.5.3.5.146 UModelAPI - IUMLReception

Interface **IUMLReception**

diagram



hierarchy	
typedElements	Interface **IUMLClass** Interface **IUMLDataAll** Interface **IUMLInterface** Operation **InsertOwnedReceptionAt** Operation **InsertOwnedReceptionAt** Operation **InsertOwnedReceptionAt**

Operation **IUMLReception::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass**			

Operation **IUMLReception::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLReception::Signal**

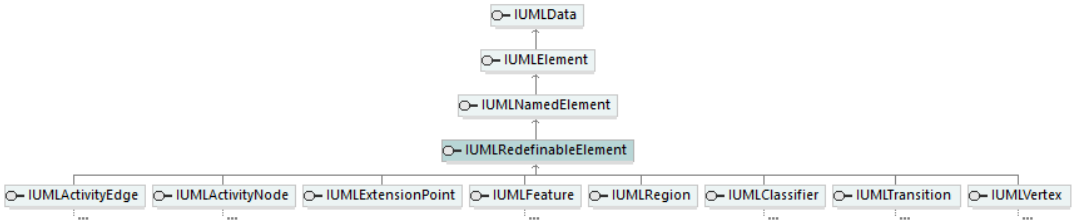
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.147 UModelAPI - IUMLRedefinableElement

Interface **IUMLRedefinableElement**

diagram	

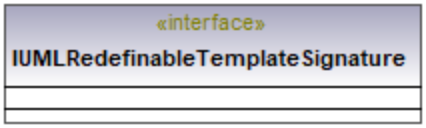
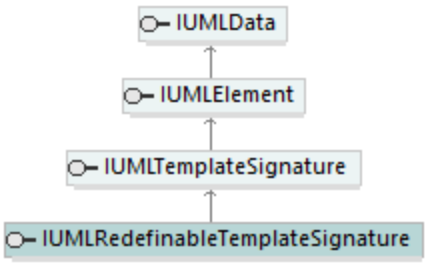
hierarchy	

Operation **IUMLRedefinableElement::IsLeaf**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

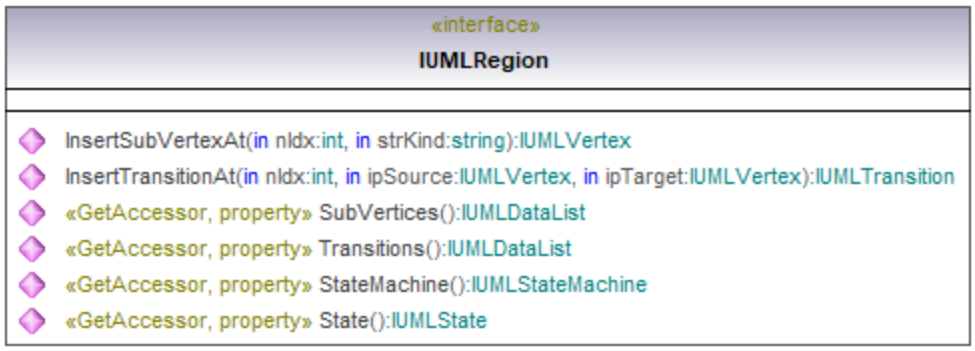
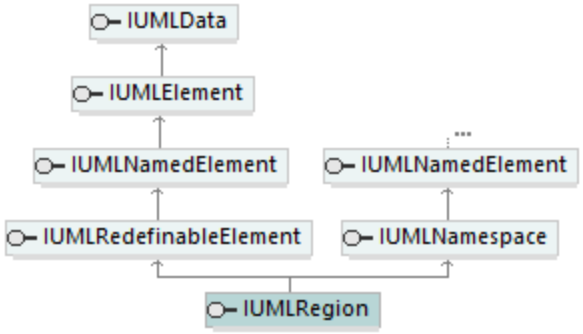
17.5.3.5.148 UModelAPI - IUMLRedefinableTemplateSignature

Interface **IUMLRedefinableTemplateSignature**

diagram	
hierarchy	

17.5.3.5.149 UModelAPI - IUMLRegion

Interface **IUMLRegion**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLState** Interface **IUMLStateMachine** Interface **IUMLVertex** Operation **Container IUMLRegion::InsertRegionAt** Operation **IUMLRegion::InsertRegionAt** Operation **IUMLRegion::InsertRegionAt** Operation **IUMLRegion::Container**

Operation **IUMLRegion::InsertSubVertexAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLVertex**			

Operation **IUMLRegion::InsertTransitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSource	**in**	**IUMLVertex**			
	ipTarget	**in**	**IUMLVertex**			
	return	**return**	**IUMLTransition**			

Operation **IUMLRegion::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

Operation **IUMLRegion::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine**			

Operation **IUMLRegion::SubVertices**

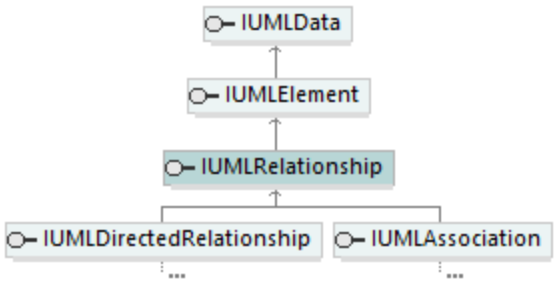
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLVertex.					

Operation **IUMLRegion::Transitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTransition.					

17.5.3.5.150 UModelAPI - IUMLRelationship

Interface **IUMLRelationship**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInformationFlow** Operation **InsertInformationFlowRealizationAt** Operation **InsertInformationFlowRealizationAt**	

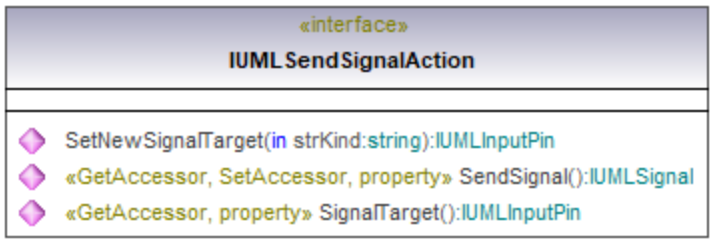
Operation **IUMLRelationship::RelatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

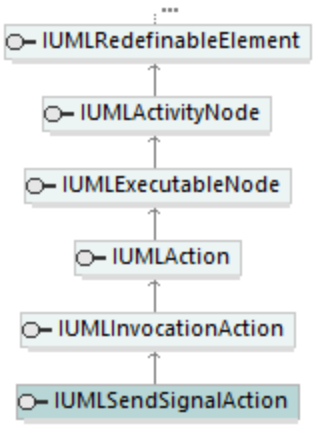
17.5.3.5.151 UModelAPI - IUMLSendSignalAction

Interface **IUMLSendSignalAction**

diagram



hierarchy



Operation **IUMLSendSignalAction::SendSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

Operation **IUMLSendSignalAction::SetNewSignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin**			

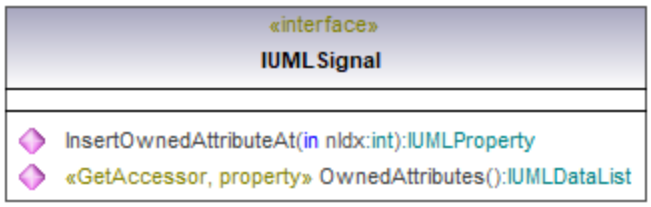
Operation **IUMLSendSignalAction::SignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin**			

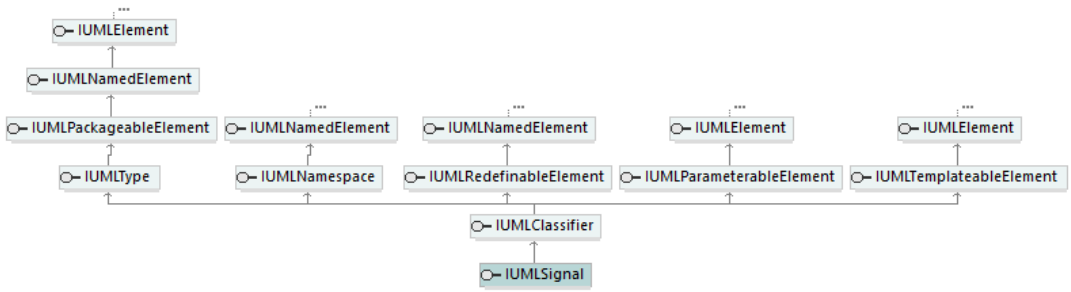
17.5.3.5.152 UModelAPI - IUMLSignal

Interface **IUMLSignal**

diagram



hierarchy



typedElements

Interface **IUMLDataAll**	Operation **OwningSignal** **SendSignal** **Signal**
Interface **IUMLProperty**	Operation **OwningSignal**
Interface **IUMLReception**	Operation **Signal**
Interface **IUMLSendSignalAction**	Operation **SendSignal**
Interface **IUMLSignalEvent**	Operation **Signal**

Operation **IUMLSignal::InsertOwnedAttributeAt**

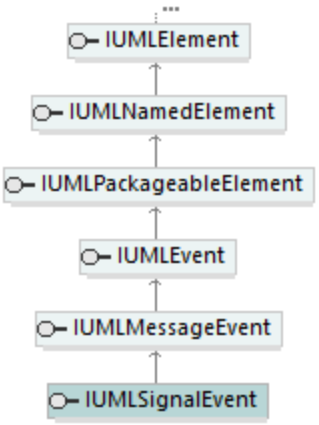
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLSignal::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

17.5.3.5.153 UModelAPI - IUMLSignalEvent

Interface **IUMLSignalEvent**

diagram	
hierarchy	

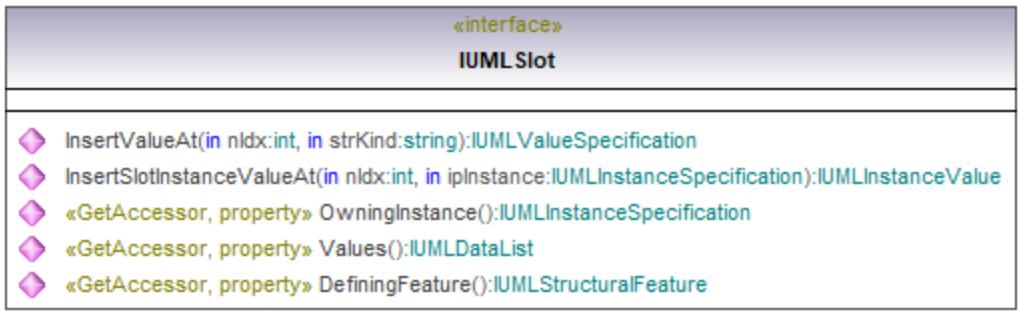
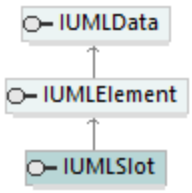
Operation **IUMLSignalEvent::Signal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.154 UModelAPI - IUMLSlot

Interface **IUMLSlot**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInstanceSpecification** Interface **IUMLValueSpecification** Operation **InsertSlotAt** **OwningSlot** Operation **InsertSlotAt** Operation **OwningSlot**

Operation **IUMLSlot::DefiningFeature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStructuralFeature**			

Operation **IUMLSlot::InsertSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLSlot::InsertValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLSlot::OwningInstance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

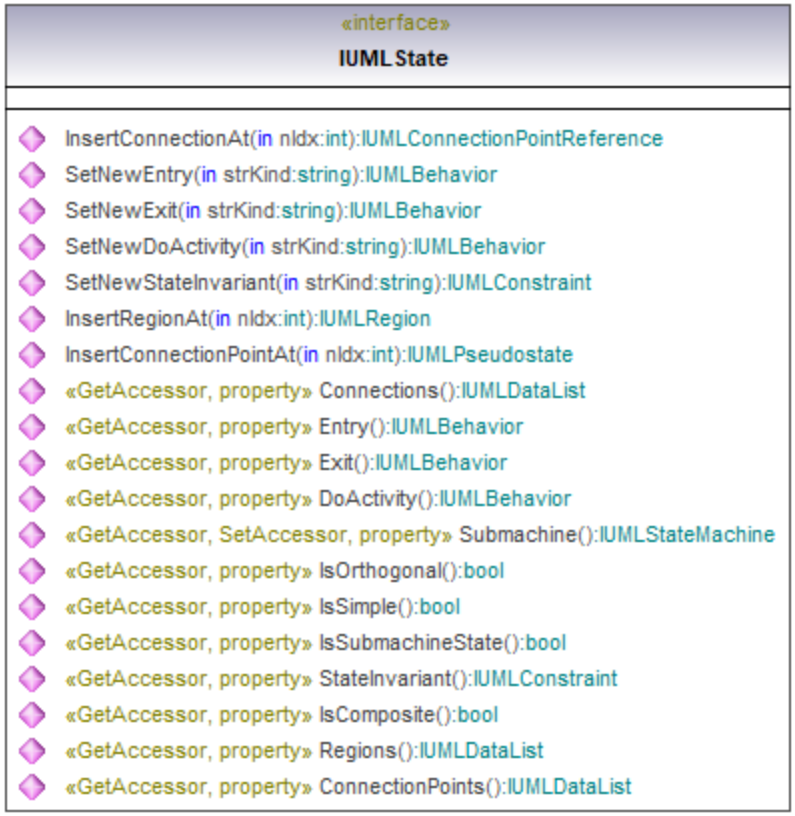
Operation **IUMLSlot::Values**

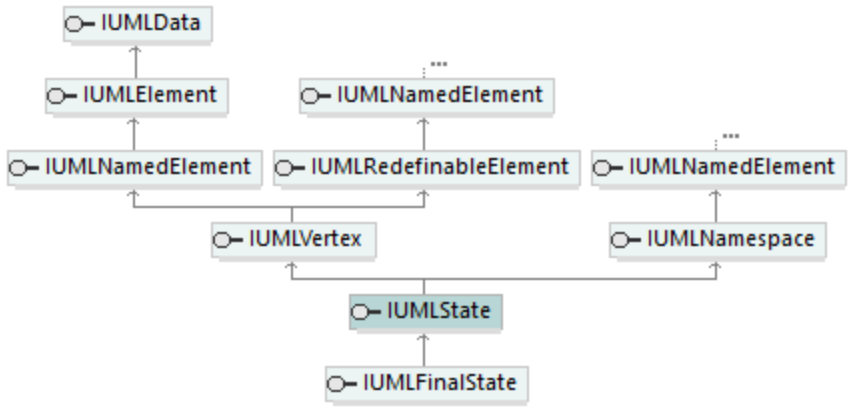
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLValueSpecification.					

17.5.3.5.155 UModelAPI - IUMLState

Interface **IUMLState**

diagram



hierarchy	
typedElements	Interface **IUMLConnectionPointReference** Interface **IUMLConstraint** Interface **IUMLDataAll** Interface **IUMLObjectNode** Interface **IUMLPseudostate** Interface **IUMLRegion** Operation **State** Operation **OwningState** Operation **InsertInStateAt** **OwningState** **State** Operation **InsertInStateAt** Operation **State** Operation **State**

Operation **IUMLState::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPseudostate.					

Operation **IUMLState::Connections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConnectionPointReference.					

Operation **IUMLState::DoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::Exit**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::InsertConnectionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConnectionPointReference**			

Operation **IUMLState::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPseudostate**			

Operation **IUMLState::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLRegion**			

Operation **IUMLState::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLState::IsOrthogonal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLState::IsSimple**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLState::IsSubmachineState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLState::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLRegion.					

Operation **IUMLState::SetNewDoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::SetNewEntry**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::SetNewExit**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::SetNewStateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLState::StateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLState::Submachine**

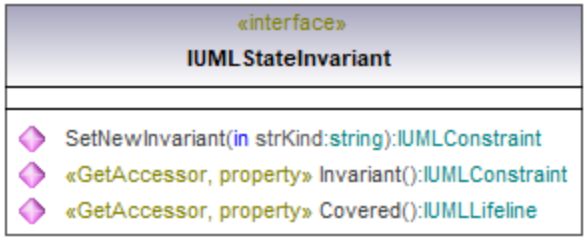
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine**			

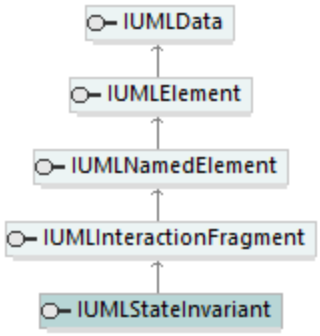
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.156 UModelAPI - IUMLStateInvariant

Interface **IUMLStateInvariant**

diagram	



hierarchy	

Operation **IUMLStateInvariant::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLStateInvariant::Invariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

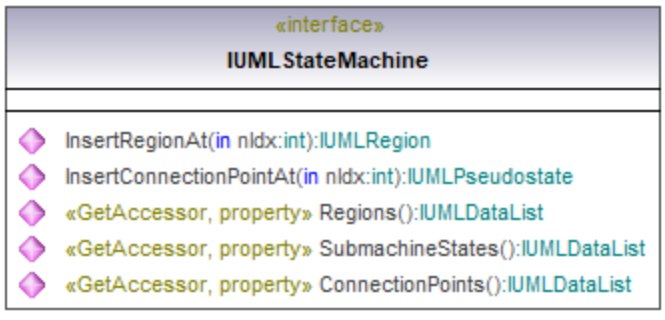
Operation **IUMLStateInvariant::SetNewInvariant**

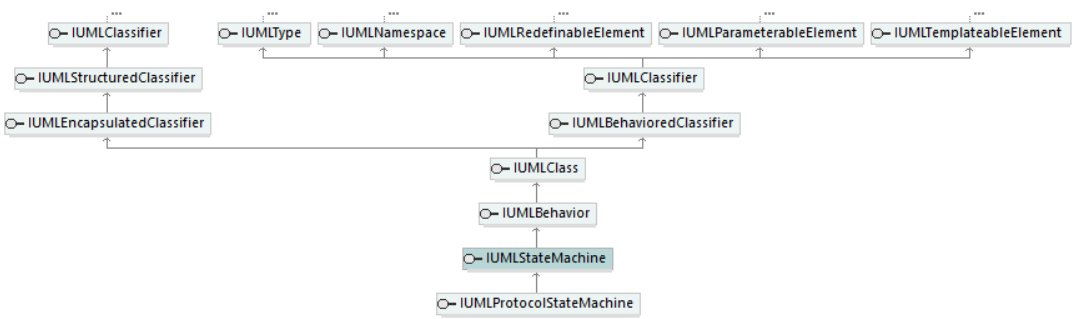
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.157 UModelAPI - IUMLStateMachine

Interface **IUMLStateMachine**

diagram	

hierarchy	
typedElements	Interface **IDocument** Interface **IUMLDataAll** Interface **IUMLPseudostate** Interface **IUMLRegion** Interface **IUMLState** Operation **GenerateStateMachineCode** Operation **StateMachine Submachine** Operation **StateMachine** Operation **StateMachine** Operation **Submachine**

Operation **IUMLStateMachine::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPseudostate.					

Operation **IUMLStateMachine::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPseudostate**			

Operation **IUMLStateMachine::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLRegion**			

Operation **IUMLStateMachine::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLRegion.					

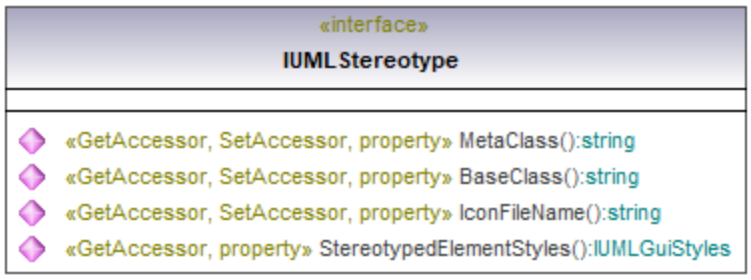
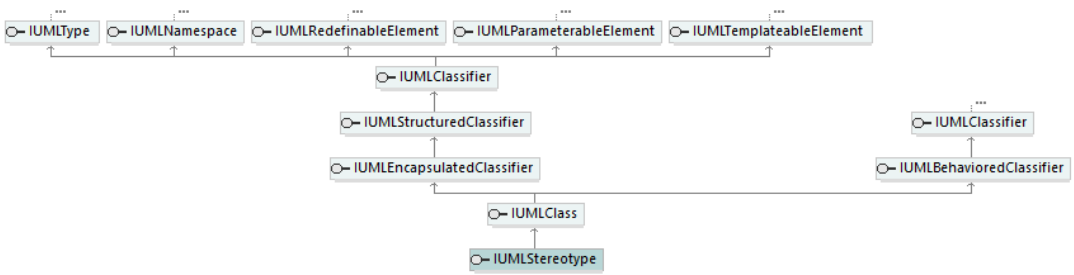
Operation **IUMLStateMachine::SubmachineStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLState.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.158 UModelAPI - IUMLStereotype

Interface **IUMLStereotype**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Operation **ApplyStereotype** **GetStereotypeApplicationForStereotype** **IsStereotypeApplied** **StereotypeUnapplyStereotype** Interface **IUMLElement** Operation **ApplyStereotype** **GetStereotypeApplicationForStereotype** **IsStereotypeApplied** **UnapplyStereotype** Interface **IUMLStereotypeApplication** Operation **Stereotype**

Operation **IUMLStereotype::BaseClass**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLStereotype::IconFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLStereotype::MetaClass**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

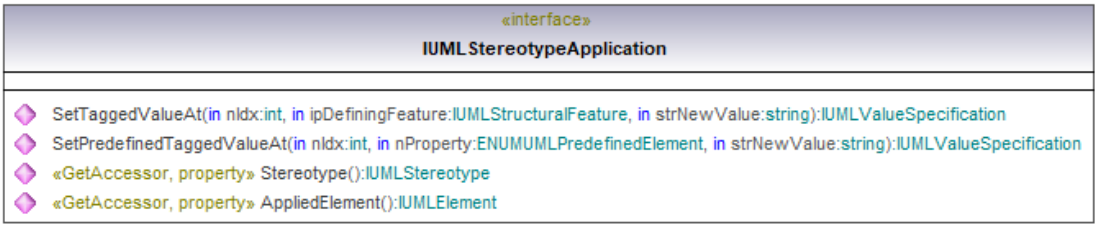
Operation **IUMLStereotype::StereotypedElementStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiStyles			

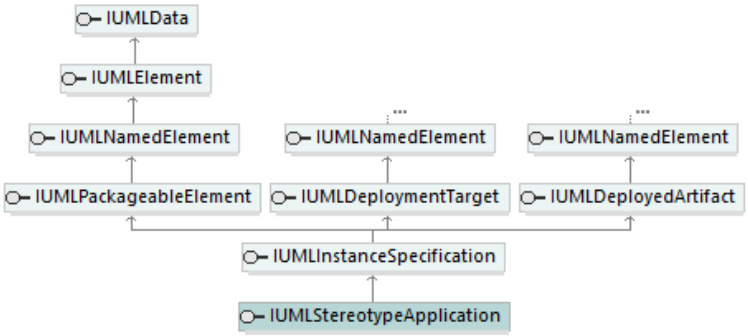
17.5.3.5.159 UModelAPI - IUMLStereotypeApplication

Interface **IUMLStereotypeApplication**

diagram



hierarchy



typedElements

Interface IUMLDataAll	Operation ApplyPredefinedStereotype ApplyStereotype GetStereotypeApplicationForPredefinedStereotype GetStereotypeApplicationForStereotype
Interface IUMLElement	Operation ApplyPredefinedStereotype ApplyStereotype GetStereotypeApplicationForPredefinedStereotype GetStereotypeApplicationForStereotype

Operation **IUMLStereotypeApplication::AppliedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLElement			

Operation **IUMLStereotypeApplication::SetPredefinedTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nProperty	**in**	**ENUMUMLPredefinedElement**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLStereotypeApplication::SetTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	**IUMLStructuralFeature**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLStereotypeApplication::Stereotype**

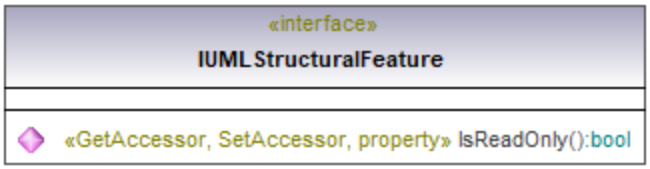
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStereotype**			

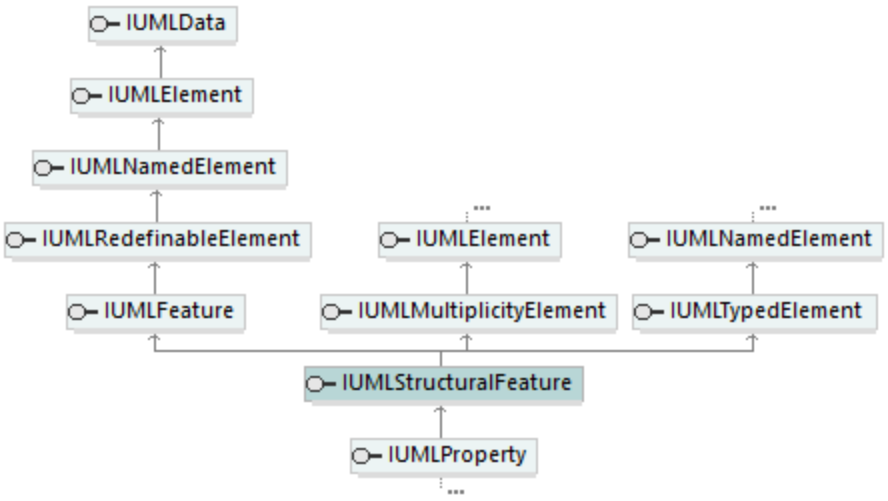
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.160 UModelAPI - IUMLStructuralFeature

Interface **IUMLStructuralFeature**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInstanceSpecification** Interface **IUMLSlot** Interface **IUMLStereotypeApplication** Operation **DefiningFeature** **InsertSlotAt** **SetSlotInstanceValueAt** **SetSlotValueAt** **SetTaggedValueAt** Operation **InsertSlotAt** **SetSlotInstanceValueAt** **SetSlotValueAt** Operation **DefiningFeature** Operation **SetTaggedValueAt**

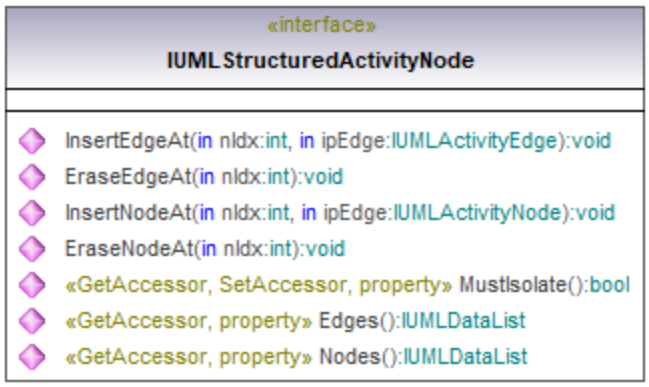
Operation **IUMLStructuralFeature::IsReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

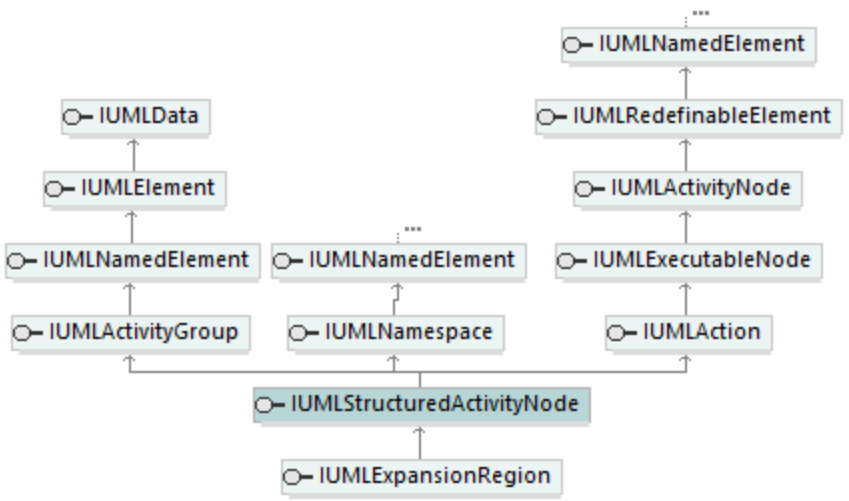
17.5.3.5.161 UModelAPI - IUMLStructuredActivityNode

Interface **IUMLStructuredActivityNode**

diagram



hierarchy



Operation **IUMLStructuredActivityNode::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLStructuredActivityNode::EraseEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge**			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityNode**			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::MustIsolate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLStructuredActivityNode::Nodes**

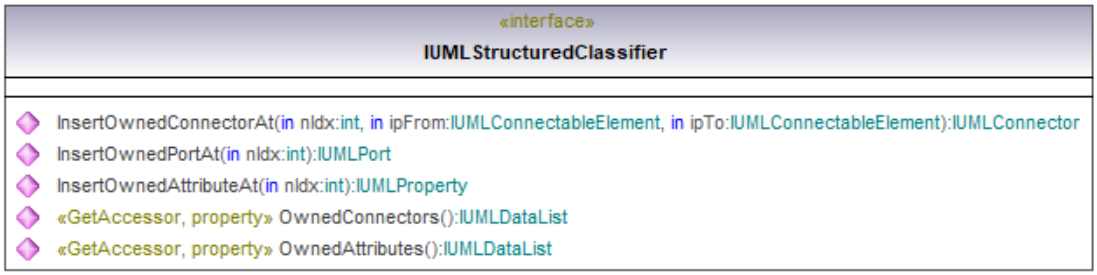
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

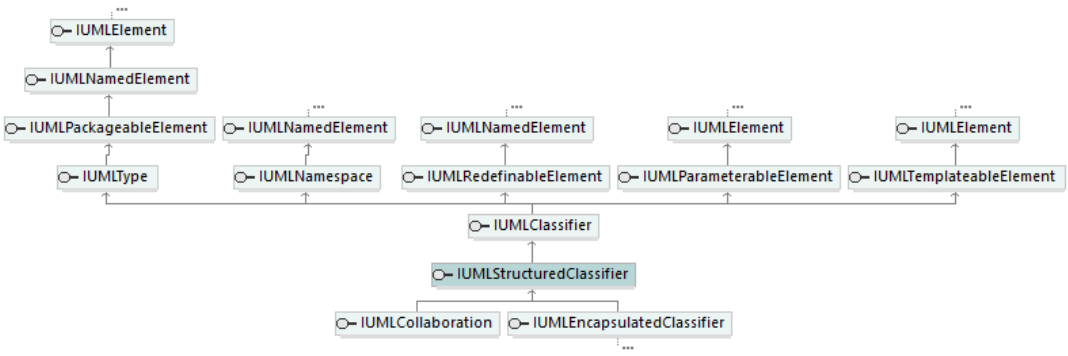
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.162 UModelAPI - IUMLStructuredClassifier

Interface **IUMLStructuredClassifier**

diagram



hierarchy	

Operation **IUMLStructuredClassifier::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLStructuredClassifier::InsertOwnedConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFrom	**in**	**IUMLConnectableElement**			
	ipTo	**in**	**IUMLConnectableElement**			
	return	**return**	**IUMLConnector**			

Operation **IUMLStructuredClassifier::InsertOwnedPortAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPort**			

Operation **IUMLStructuredClassifier::OwnedAttributes**

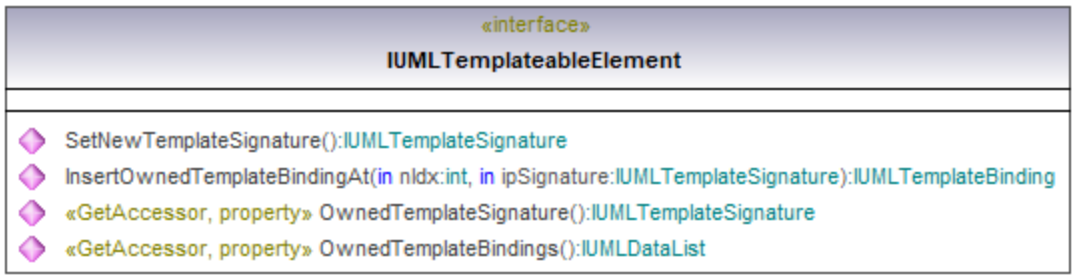
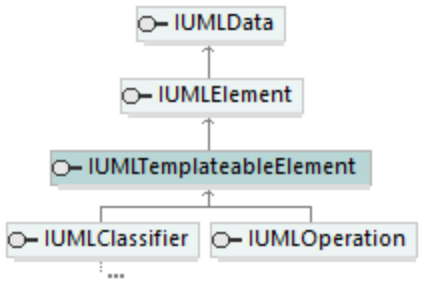
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLStructuredClassifier::OwnedConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConnector.					

17.5.3.5.163 UModelAPI - IUMLTemplateableElement

Interface **IUMLTemplateableElement**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLTemplateBinding** Interface **IUMLTemplateSignature** Operation **BoundElement** **Template** Operation **BoundElement** Operation **Template**

Operation **IUMLTemplateableElement::InsertOwnedTemplateBindingAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSignature	**in**	**IUMLTemplateSignature**			
	return	**return**	**IUMLTemplateBinding**			

Operation **IUMLTemplateableElement::OwnedTemplateBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTemplateBinding.					

Operation **IUMLTemplateableElement::OwnedTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

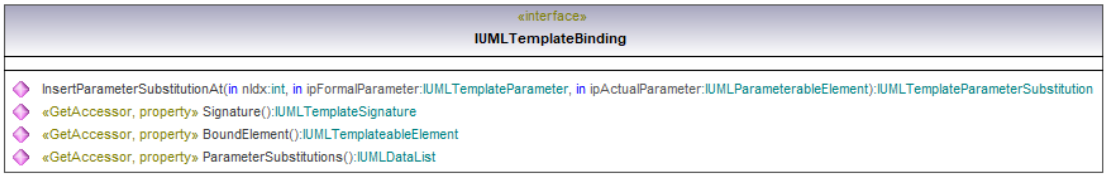
Operation **IUMLTemplateableElement::SetNewTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

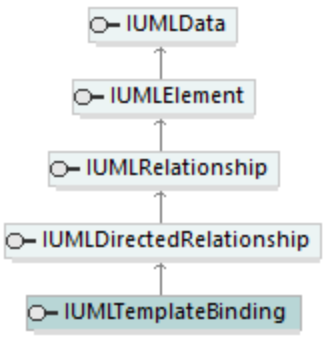
17.5.3.5.164 UModelAPI - IUMLTemplateBinding

Interface **IUMLTemplateBinding**

diagram



hierarchy



typedElements

Interface **IUMLDataAll**

Interface **IUMLTemplateableElement**

Interface **IUMLTemplateParameterSubstitution**

Operation **InsertOwnedTemplateBindingAtTemplateBinding**

Operation **InsertOwnedTemplateBindingAt**

Operation **TemplateBinding**

Operation **IUMLTemplateBinding::BoundElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateableElement**			

Operation **IUMLTemplateBinding::InsertParameterSubstitutionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	ipFormalParameter	in	IUMLTemplateParameter
	ipActualParameter	in	IUMLParameterableElement
	return	return	IUMLTemplateParameterSubstitution

Operation **IUMLTemplateBinding::ParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			
documentation	A list of elements of type IUMLTemplateParameterSubstitution.					

Operation **IUMLTemplateBinding::Signature**

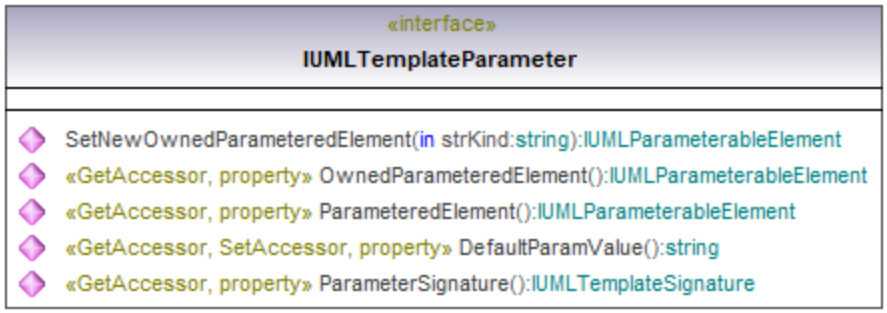
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateSignature			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

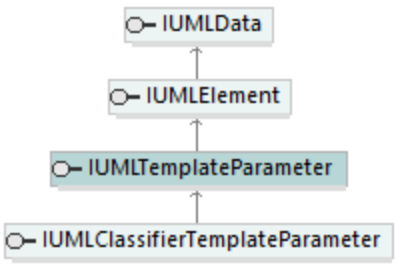
17.5.3.5.165 UModelAPI - IUMLTemplateParameter

Interface **IUMLTemplateParameter**

diagram



hierarchy



typedElements	Interface **IUMLDataAll** Interface **IUMLParameterableElement** Interface **IUMLTemplateBinding** Interface **IUMLTemplateParameterSubstitution**	Operation **Formal InsertParameterSubstitutionAt OwningTemplateParameter TemplateParameter** Operation **OwningTemplateParameter TemplateParameter** Operation **InsertParameterSubstitutionAt** Operation **Formal**

Operation **IUMLTemplateParameter::DefaultParamValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLTemplateParameter::OwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLTemplateParameter::ParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLTemplateParameter::ParameterSignature**

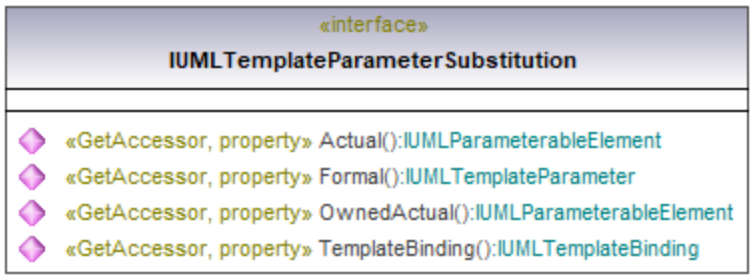
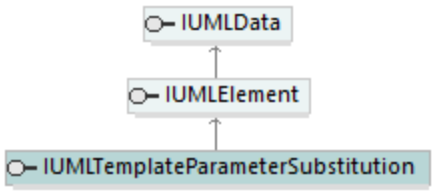
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

Operation **IUMLTemplateParameter::SetNewOwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLParameterableElement**			

17.5.3.5.166 UModelAPI - IUMLTemplateParameterSubstitution

Interface **IUMLTemplateParameterSubstitution**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLTemplateBinding** Operation **InsertParameterSubstitutionAt** Operation **InsertParameterSubstitutionAt**

Operation **IUMLTemplateParameterSubstitution::Actual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLTemplateParameterSubstitution::Formal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

Operation **IUMLTemplateParameterSubstitution::OwnedActual**

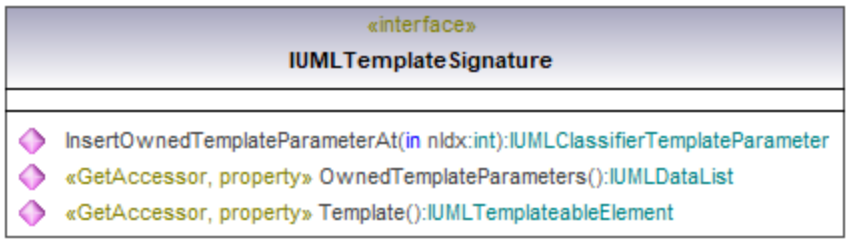
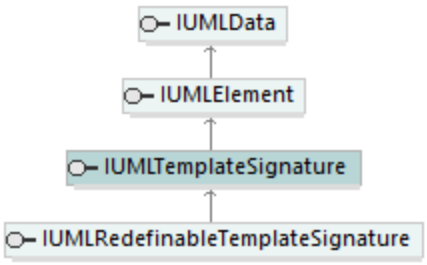
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLTemplateParameterSubstitution::TemplateBinding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateBinding**			

17.5.3.5.167 UModelAPI - IUMLTemplateSignature

Interface **IUMLTemplateSignature**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLTemplateableElement** Interface **IUMLTemplateBinding** Interface **IUMLTemplateParameter** Operation **InsertOwnedTemplateBindingAt** **OwnedTemplateSignature** **ParameterSignature** **SetNewTemplateSignature** **Signature** Operation **InsertOwnedTemplateBindingAt** **OwnedTemplateSignature** **SetNewTemplateSignature** Operation **Signature** Operation **ParameterSignature**

Operation **IUMLTemplateSignature::InsertOwnedTemplateParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLClassifierTemplateParameter**			

Operation **IUMLTemplateSignature::OwnedTemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTemplateParameter.					

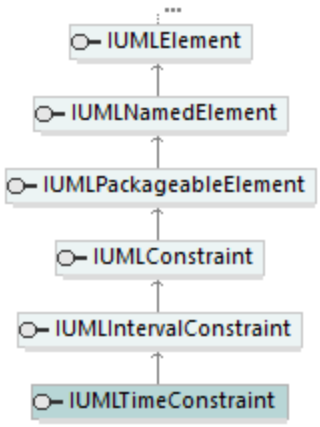
Operation **IUMLTemplateSignature::Template**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLTemplateableElement**

17.5.3.5.168 UModelAPI - IUMLTimeConstraint

Interface **IUMLTimeConstraint**

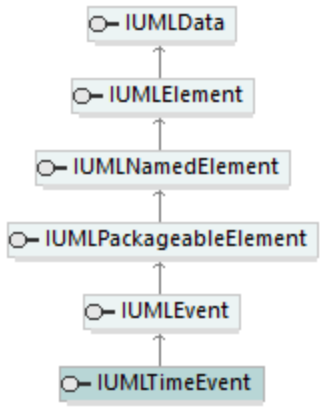
diagram	
hierarchy	

Operation **IUMLTimeConstraint::IsFirstEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.3.5.169 UModelAPI - IUMLTimeEvent

Interface **IUMLTimeEvent**

diagram	
hierarchy	

Operation **IUMLTimeEvent::IsRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLTimeEvent::SetNewWhen**

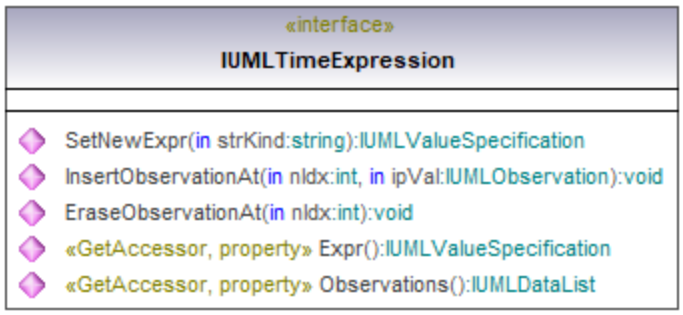
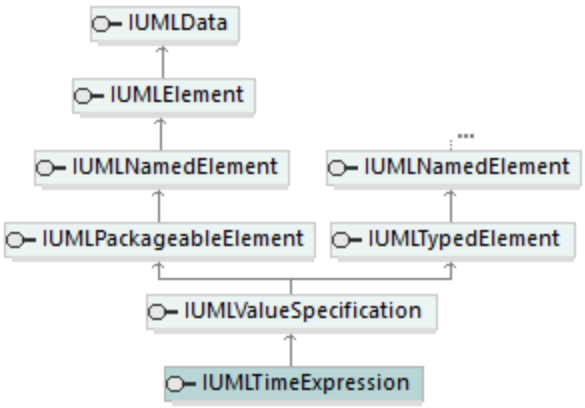
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression**			

Operation **IUMLTimeEvent::When**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression**			

17.5.3.5.170 UModelAPI - IUMLTimeExpression

Interface **IUMLTimeExpression**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLTimeEvent** Operation **SetNewWhen** **When** Operation **SetNewWhen** **When**

Operation **IUMLTimeExpression::EraseObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLTimeExpression::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLTimeExpression::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLObservation**			
	return	**return**	**void**			

Operation **IUMLTimeExpression::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLObservation.					

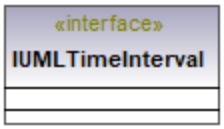
Operation **IUMLTimeExpression::SetNewExpr**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

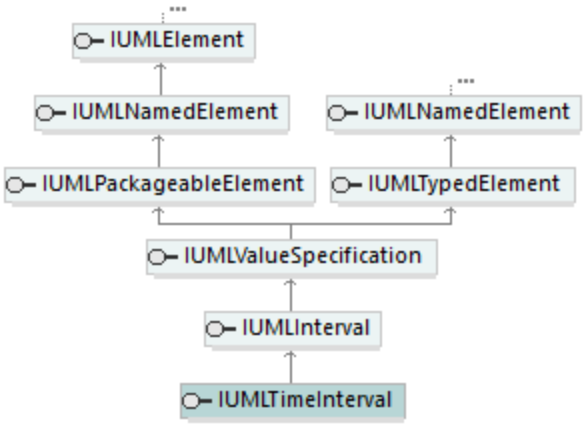
17.5.3.5.171 UModelAPI - IUMLTimeInterval

Interface **IUMLTimeInterval**

diagram

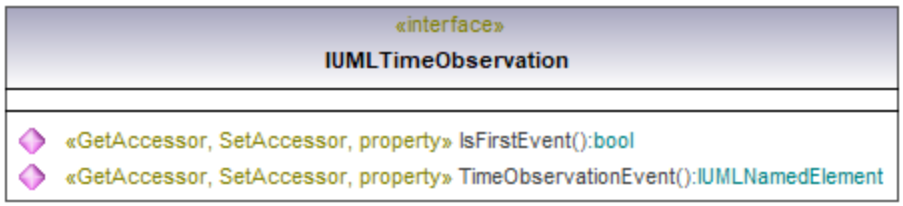
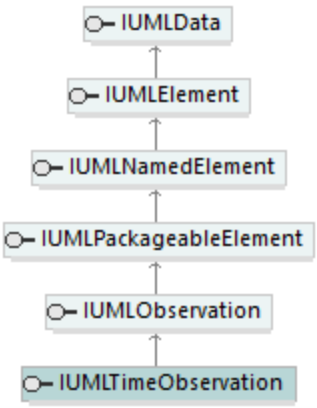


hierarchy



17.5.3.5.172 UModelAPI - IUMLTimeObservation

Interface **IUMLTimeObservation**

diagram	
hierarchy	

Operation **IUMLTimeObservation::IsFirstEvent**

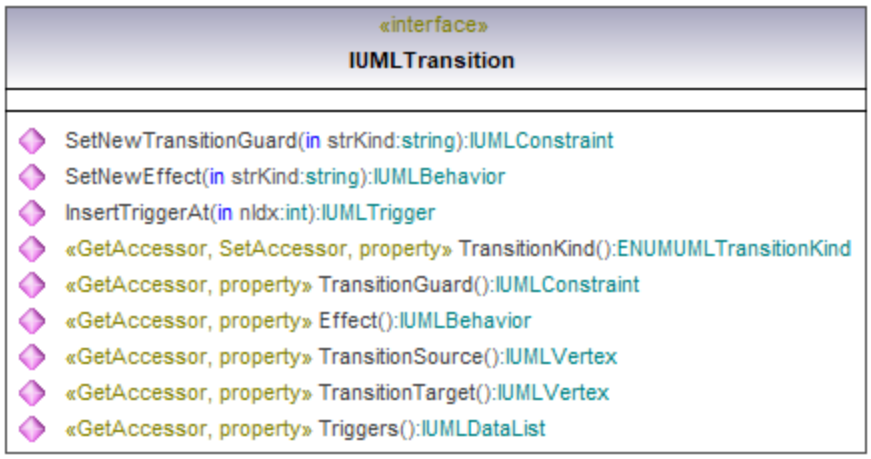
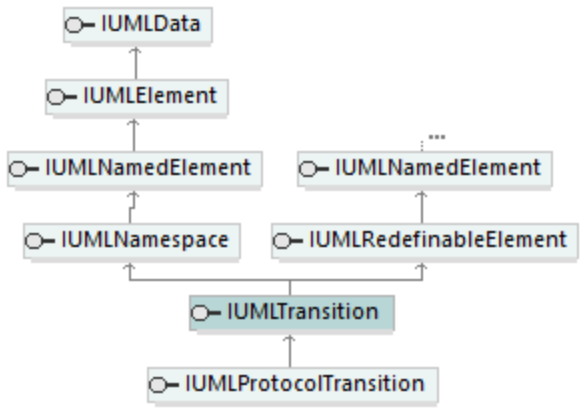
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLTimeObservation::TimeObservationEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamedElement**			

17.5.3.5.173 UModelAPI - IUMLTransition

Interface **IUMLTransition**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLRegion** Operation **InsertTransitionAt** Operation **InsertTransitionAt**

Operation **IUMLTransition::Effect**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLTransition::InsertTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger**			

Operation **IUMLTransition::SetNewEffect**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			

	return	**return**	**IUMLBehavior**			

Operation **IUMLTransition::SetNewTransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLTransition::TransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLTransition::TransitionKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLTransitionKind**			

Operation **IUMLTransition::TransitionSource**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex**			

Operation **IUMLTransition::TransitionTarget**

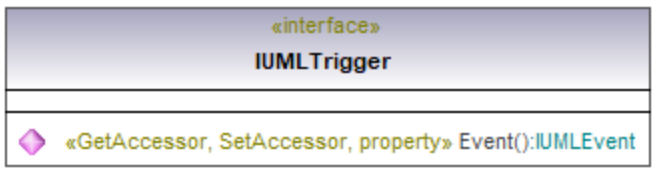
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex**			

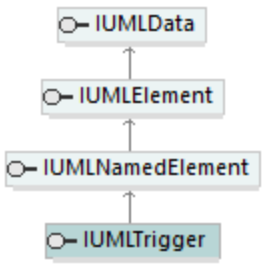
Operation **IUMLTransition::Triggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTrigger.					

17.5.3.5.174 UModelAPI - IUMLTrigger

Interface **IUMLTrigger**

diagram	

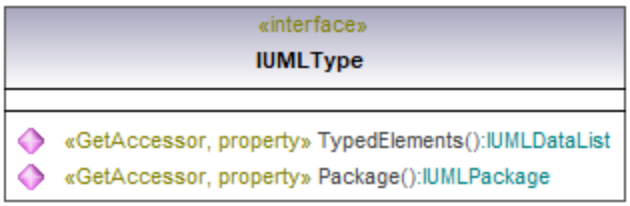
hierarchy	
typedElements	Interface **IUMLAcceptEventAction** Interface **IUMLDataAll** Interface **IUMLTransition** Operation **InsertActionTriggerAt** Operation **InsertActionTriggerAt** Operation **InsertTriggerAt**

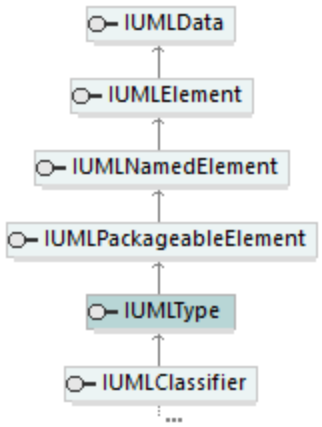
Operation **IUMLTrigger::Event**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEvent**			

17.5.3.5.175 UModelAPI - IUMLType

Interface **IUMLType**

diagram	

hierarchy	
typedElements	Interface **IUMLBehavioralFeature** Interface **IUMLDataAll** Interface **IUMLOperation** Interface **IUMLTypedElement** Operation **InsertRaisedExceptionAt** Operation **InsertRaisedExceptionAt Type** Operation **Type** Operation **Type**

Operation **IUMLType::Package**

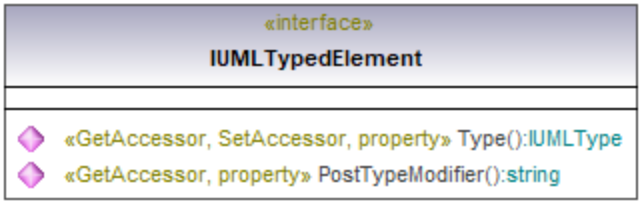
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

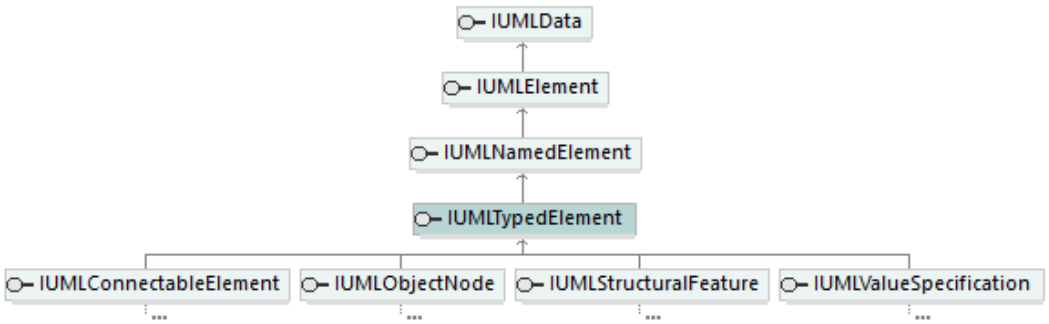
Operation **IUMLType::TypedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTypedElement.					

17.5.3.5.176 UModelAPI - IUMLTypedElement

Interface **IUMLTypedElement**

diagram	

hierarchy	

Operation **IUMLTypedElement::PostTypeModifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

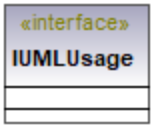
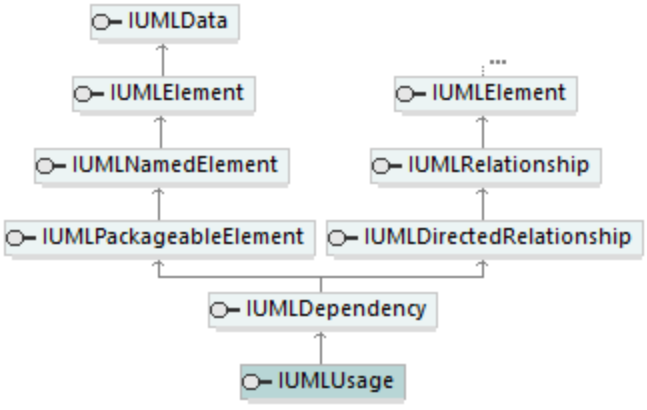
Operation **IUMLTypedElement::Type**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLType**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.177 UModelAPI - IUMLUsage

Interface **IUMLUsage**

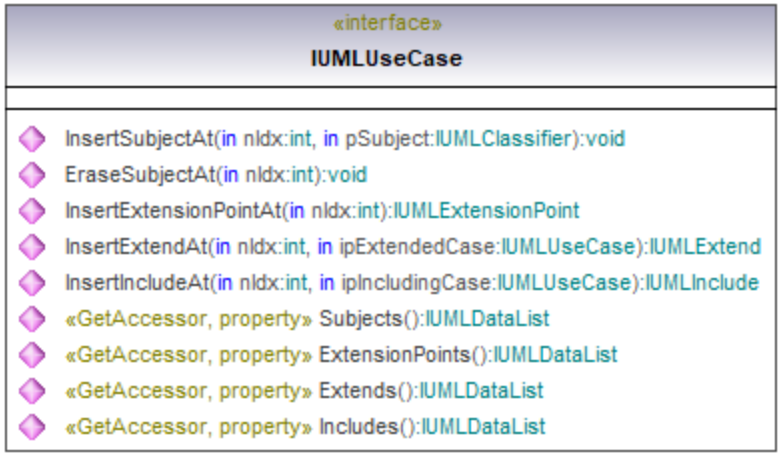
diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

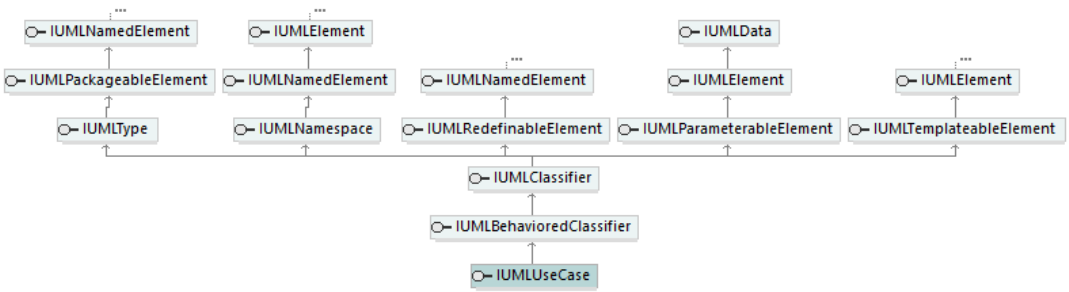
17.5.3.5.178 UModelAPI - IUMLUseCase

Interface **IUMLUseCase**

diagram



hierarchy



typedElements

Interface **IUMLClassifier**	Operation **InsertOwnedUseCaseAt**
Interface **IUMLDataAll**	Operation **Addition ExtendedCase Extension IncludingCase InsertExtendAt InsertIncludeAt InsertOwnedUseCaseAt UseCase**
Interface **IUMLExtend**	Operation **ExtendedCase Extension**
Interface **IUMLExtensionPoint**	Operation **UseCase**
Interface **IUMLInclude**	Operation **Addition IncludingCase**
Interface **IUMLUseCase**	Operation **InsertExtendAt InsertIncludeAt**

Operation **IUMLUseCase::EraseSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLUseCase::Extends**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLExtend.					

Operation **IUMLUseCase::ExtensionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLExtensionPoint.					

Operation **IUMLUseCase::Includes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInclude.					

Operation **IUMLUseCase::InsertExtendAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipExtendedCase	**in**	**IUMLUseCase**			
	return	**return**	**IUMLExtend**			

Operation **IUMLUseCase::InsertExtensionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExtensionPoint**			

Operation **IUMLUseCase::InsertIncludeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipIncludingCase	**in**	**IUMLUseCase**			
	return	**return**	**IUMLInclude**			

Operation **IUMLUseCase::InsertSubjectAt**

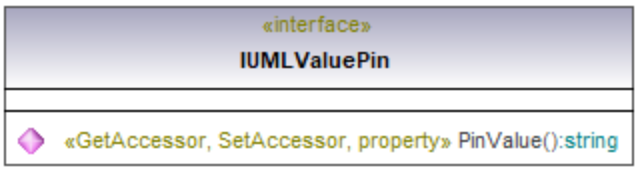
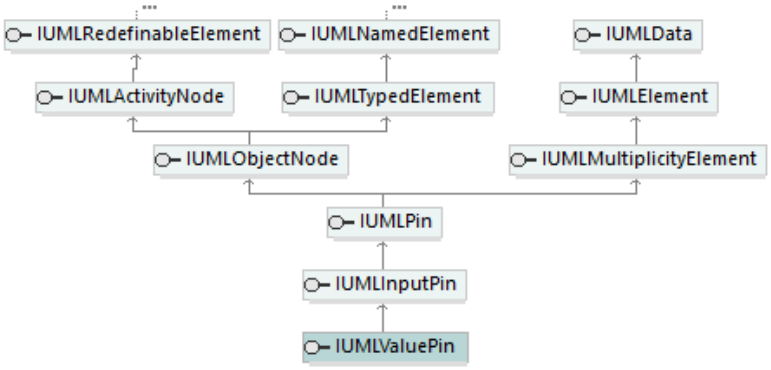
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pSubject	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLUseCase::Subjects**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

17.5.3.5.179 UModelAPI - IUMLValuePin

Interface **IUMLValuePin**

diagram	
hierarchy	

Operation **IUMLValuePin::PinValue**

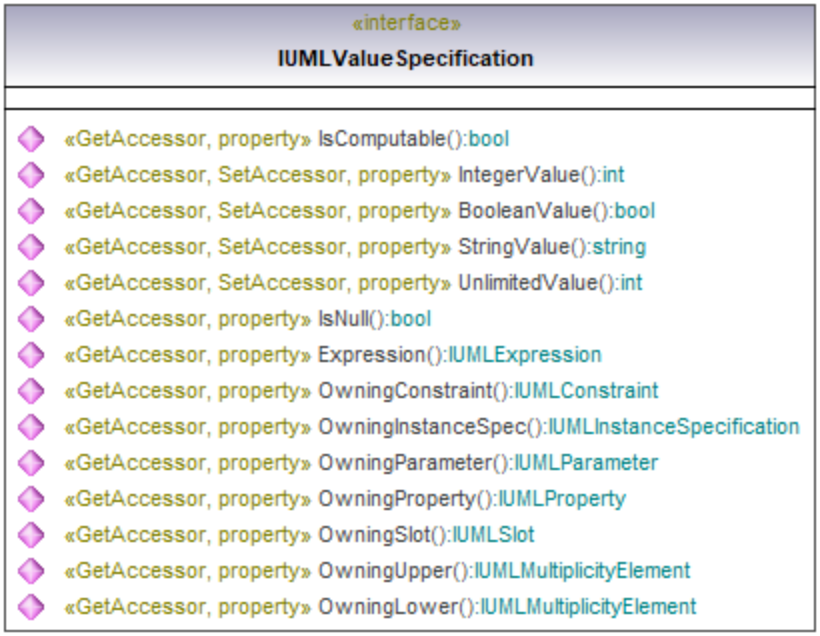
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

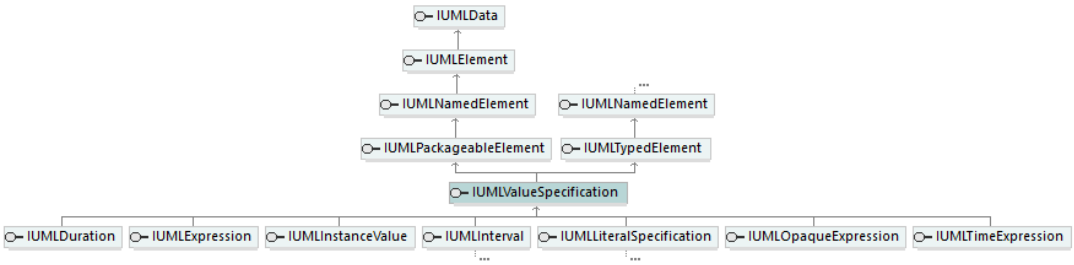
17.5.3.5.180 UModelAPI - IUMLValueSpecification

Interface **IUMLValueSpecification**

diagram



hierarchy



typedElements

Interface **IUMLChangeEvent**

Operation **ChangeExpression** **SetNewChangeExpression**

Interface **IUMLConstraint**

Operation **SetNewSpecification** **Specification**

Interface **IUMLDataAll**

Operation **ActionValue** **ChangeExpression** **DefaultValue** **Expr** **InsertOwnedArgumentAt** **InsertValueAt** **Max** **MaxInt** **Min** **MinInt** **Selector** **SetNewActionValue** **SetNewChangeExpression** **SetNewDefaultValue** **SetNewExpr** **SetNewMax** **SetNewMaxInt** **SetNewMin** **SetNewMinInt** **SetNewSelector** **SetNewSpecification** **SetPredefinedTaggedValueAt** **SetSlotValueAt** **SetTaggedValueAt** **Specification**

Interface **IUMLDuration**

Operation **Expr** **SetNewExpr**

	Interface **IUMLInstanceSpecification** Interface **IUMLInteractionConstraint** Interface **IUMLInterval** Interface **IUMLLifeline** Interface **IUMLMessage** Interface **IUMLParameter** Interface **IUMLProperty** Interface **IUMLSlot** Interface **IUMLStereotypeApplication** Interface **IUMLTimeExpression** Interface **IUMLValueSpecificationAction**	Operation **SetNewSpecification SetSlotValueAt Specification** Operation **MaxInt MinInt SetNewMaxInt SetNewMinInt** Operation **Max Min SetNewMax SetNewMin** Operation **Selector SetNewSelector** Operation **InsertOwnedArgumentAt** Operation **DefaultValue SetNewDefaultValue** Operation **DefaultValue SetNewDefaultValue** Operation **InsertValueAt** Operation **SetPredefinedTaggedValueAt SetTaggedValueAt** Operation **Expr SetNewExpr** Operation **ActionValue SetNewActionValue**

Operation **IUMLValueSpecification::BooleanValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLValueSpecification::Expression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpression**			

Operation **IUMLValueSpecification::IntegerValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLValueSpecification::IsComputable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLValueSpecification::IsNull**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLValueSpecification::OwningConstraint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLValueSpecification::OwningInstanceSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

Operation **IUMLValueSpecification::OwningLower**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMultiplicityElement**			

Operation **IUMLValueSpecification::OwningParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameter**			

Operation **IUMLValueSpecification::OwningProperty**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLValueSpecification::OwningSlot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSlot**			

Operation **IUMLValueSpecification::OwningUpper**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMultiplicityElement**			

Operation **IUMLValueSpecification::StringValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLValueSpecification::UnlimitedValue**

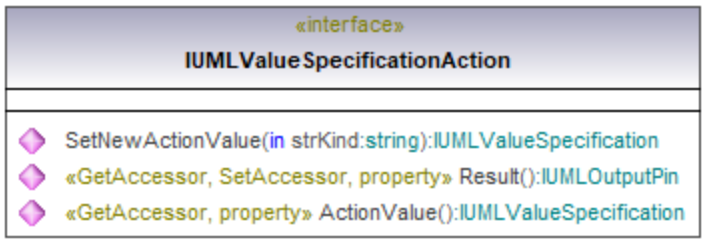
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

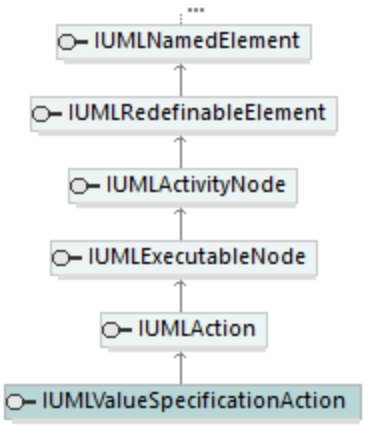
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.181 UModelAPI - IUMLValueSpecificationAction

Interface **IUMLValueSpecificationAction**

diagram



hierarchy	

Operation **IUMLValueSpecificationAction::ActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLValueSpecificationAction::Result**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOutputPin**			

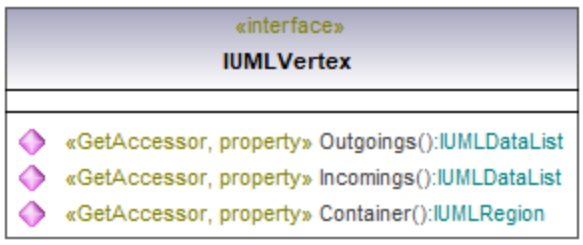
Operation **IUMLValueSpecificationAction::SetNewActionValue**

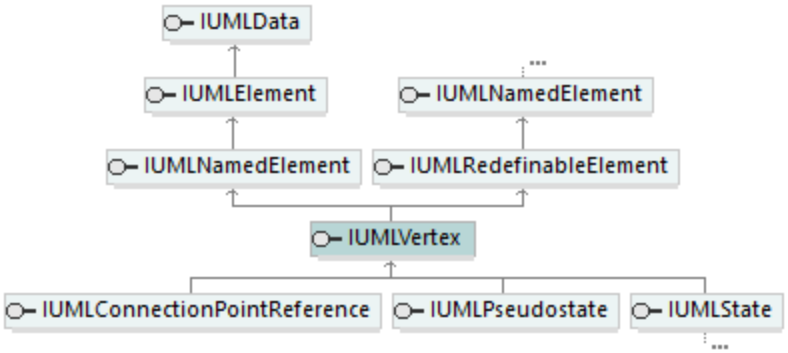
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.182 UModelAPI - IUMLVertex

Interface **IUMLVertex**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** — Operation **InsertSubVertexAt**, **InsertTransitionAt**, **TransitionSource**, **TransitionTarget** Interface **IUMLRegion** — Operation **InsertSubVertexAt**, **InsertTransitionAt** Interface **IUMLTransition** — Operation **TransitionSource**, **TransitionTarget**

Operation **IUMLVertex::Container**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLRegion**			

Operation **IUMLVertex::Incomings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTransition.					

Operation **IUMLVertex::Outgoings**

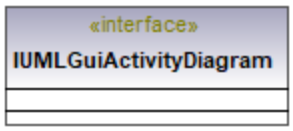
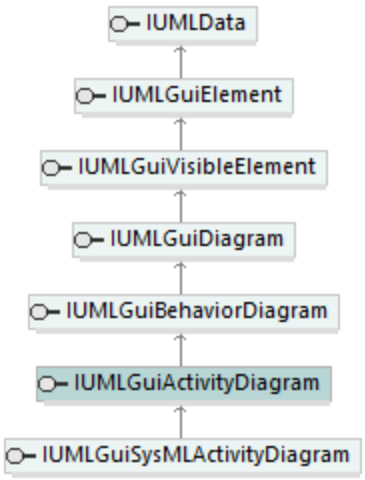
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTransition.					

17.5.3.6 IUMLGuiElement

This is a list of Altova-specific elements for diagrams, and members used to show IUMLElements on diagrams.

17.5.3.6.1 UModelAPI - IUMLGuiActivityDiagram

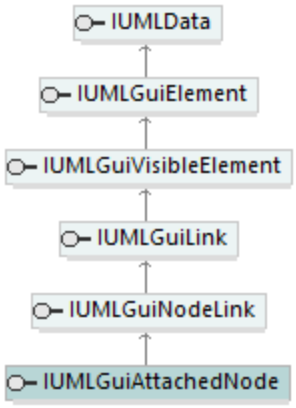
Interface **IUMLGuiActivityDiagram**

diagram	
hierarchy	

17.5.3.6.2 UModelAPI - IUMLGuiAttachedNode

Interface **IUMLGuiAttachedNode**

diagram	

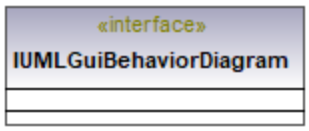
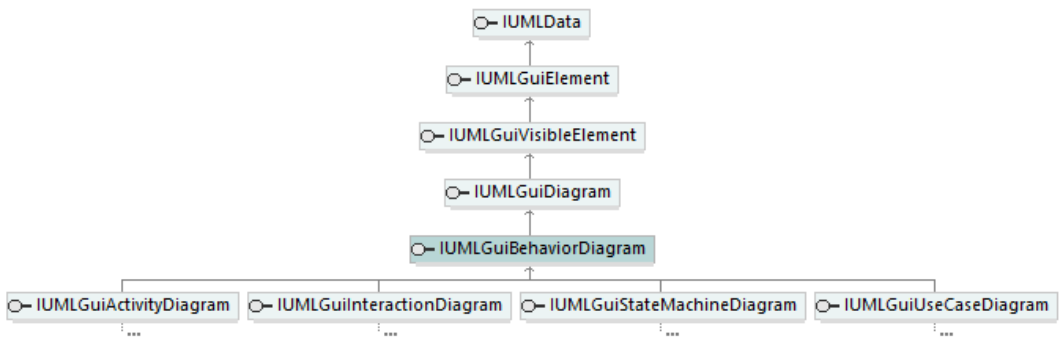
hierarchy	
documentation	This GUI element is a node (possibly without a linked IUMLElement) which is directly attached to a IUMLGuiNodeLink. It disappears and pops up based on data set in the element of the IUMLGuiNodeLink it is attached to. The user usually only has control of this element via styles or the node it is attached to. This node is used as graphical object on diagrams to represent Tagged Values for example.

Operation **IUMLGuiAttachedNode::AttachedTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiLink**			

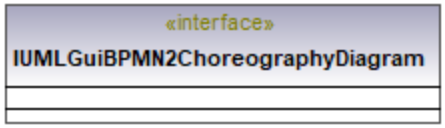
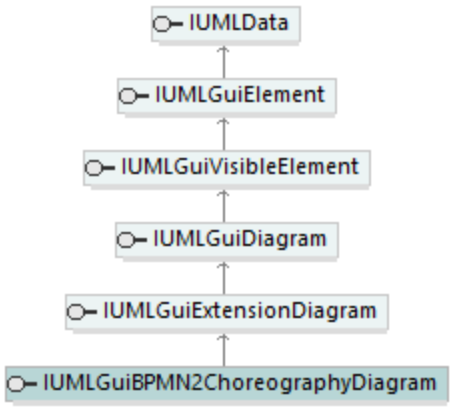
17.5.3.6.3 UModelAPI - IUMLGuiBehaviorDiagram

Interface **IUMLGuiBehaviorDiagram**

diagram	
hierarchy	

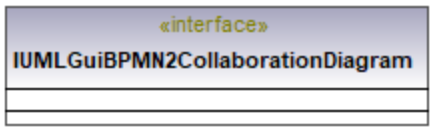
17.5.3.6.4 UModelAPI - IUMLGuiBPMN2ChoreographyDiagram

Interface **IUMLGuiBPMN2ChoreographyDiagram**

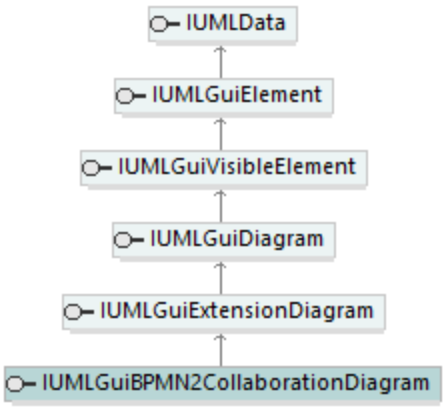
diagram	
hierarchy	

17.5.3.6.5 UModelAPI - IUMLGuiBPMN2CollaborationDiagram

Interface **IUMLGuiBPMN2CollaborationDiagram**

diagram	

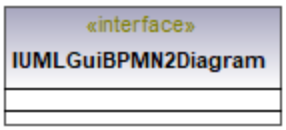
hierarchy



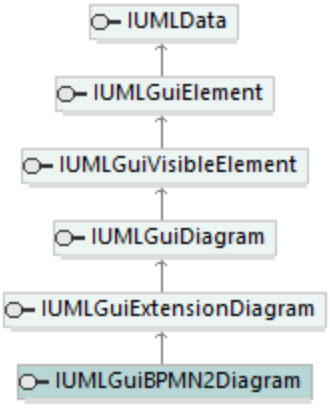
17.5.3.6.6 UModelAPI - IUMLGuiBPMN2Diagram

Interface **IUMLGuiBPMN2Diagram**

diagram

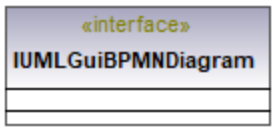
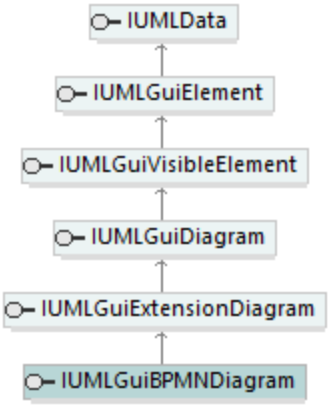


hierarchy



17.5.3.6.7 UModelAPI - IUMLGuiBPMNDiagram

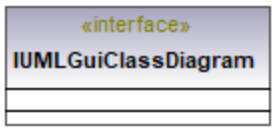
Interface **IUMLGuiBPMNDiagram**

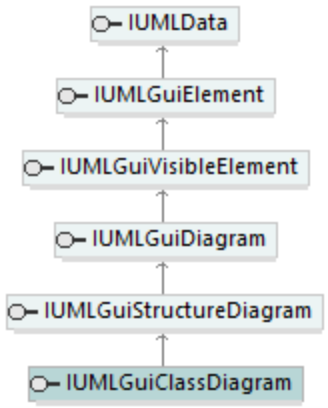
diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.8 UModelAPI - IUMLGuiClassDiagram

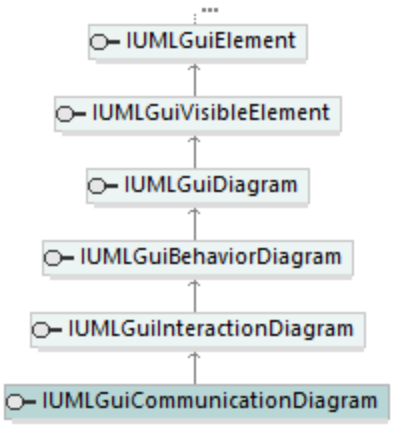
Interface **IUMLGuiClassDiagram**

diagram	

hierarchy	

17.5.3.6.9 UModelAPI - IUMLGuiCommunicationDiagram

Interface **IUMLGuiCommunicationDiagram**

diagram	
hierarchy	

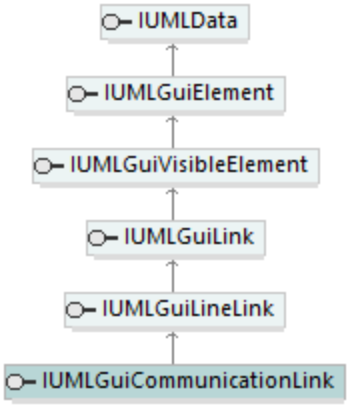
Operation **IUMLGuiCommunicationDiagram::AddUMLGuiCommunicationLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	**in**	**IUMLGuiLink**			

	ipToLink **return**	**in** **return**	**IUMLGuiLink** **IUMLGuiCommunicationLink**

17.5.3.6.10 UModelAPI - IUMLGuiCommunicationLink

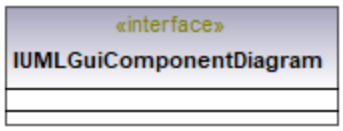
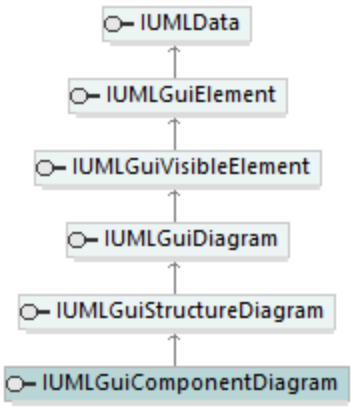
Interface **IUMLGuiCommunicationLink**

diagram	
hierarchy	
typedElements	Interface **IUMLGuiCommunicationDiagram** Operation **AddUMLGuiCommunicationLink**
documentation	This line link is used on communication diagrams to provide a connection link for messages between lifelines.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.11 UModelAPI - IUMLGuiComponentDiagram

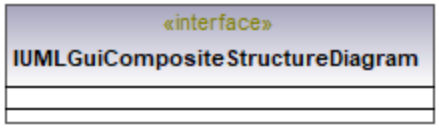
Interface **IUMLGuiComponentDiagram**

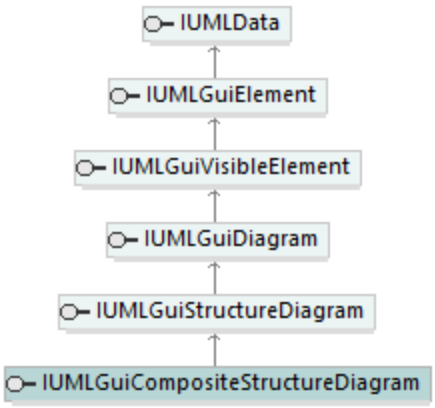
diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.12 UModelAPI - IUMLGuiCompositeStructureDiagram

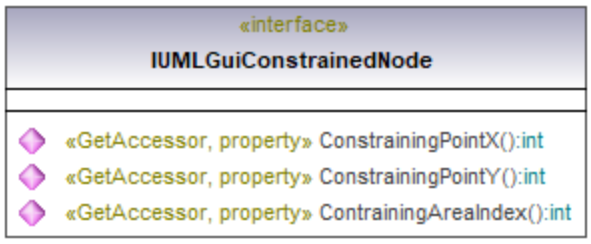
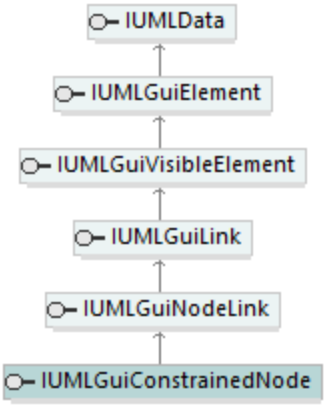
Interface **IUMLGuiCompositeStructureDiagram**

diagram	

hierarchy	

17.5.3.6.13 UModelAPI - IUMLGuiConstrainedNode

Interface **IUMLGuiConstrainedNode**

diagram	
hierarchy	
documentation	This node link is used to represent objects on diagrams which can be directly attached to a parent node link and placed relatively to and in special constraining areas of the parent. Used for example for Pins on Activtiy Diagrams.

Operation **IUMLGuiConstrainedNode::ConstrainingPointX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			
documentation	X coordinate relative to the uppor left position of the contraining area.					

Operation **IUMLGuiConstrainedNode::ConstrainingPointY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			
documentation	Y coordinate relative to the uppor left position of the contraining area.					

Operation **IUMLGuiConstrainedNode::ContrainingAreaIndex**

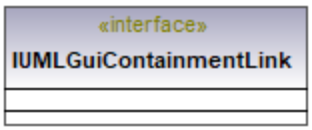
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			
documentation	Defines the index of the area w here this node is currently in and to w hich its relative position has its origin.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

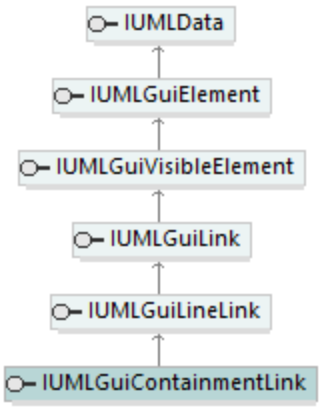
17.5.3.6.14 UModelAPI - IUMLGuiContainmentLink

Interface **IUMLGuiContainmentLink**

diagram



hierarchy



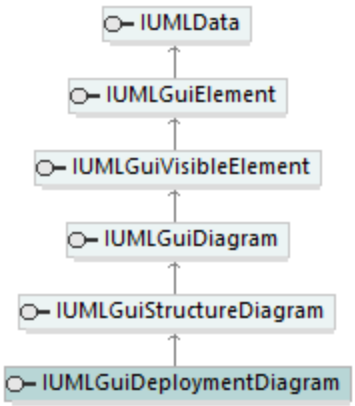
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiDiagram**	Operation **AddUMLGuiContainmentLink** Operation **AddUMLGuiContainmentLink**

17.5.3.6.15 UModelAPI - IUMLGuiDeploymentDiagram

Interface **IUMLGuiDeploymentDiagram**

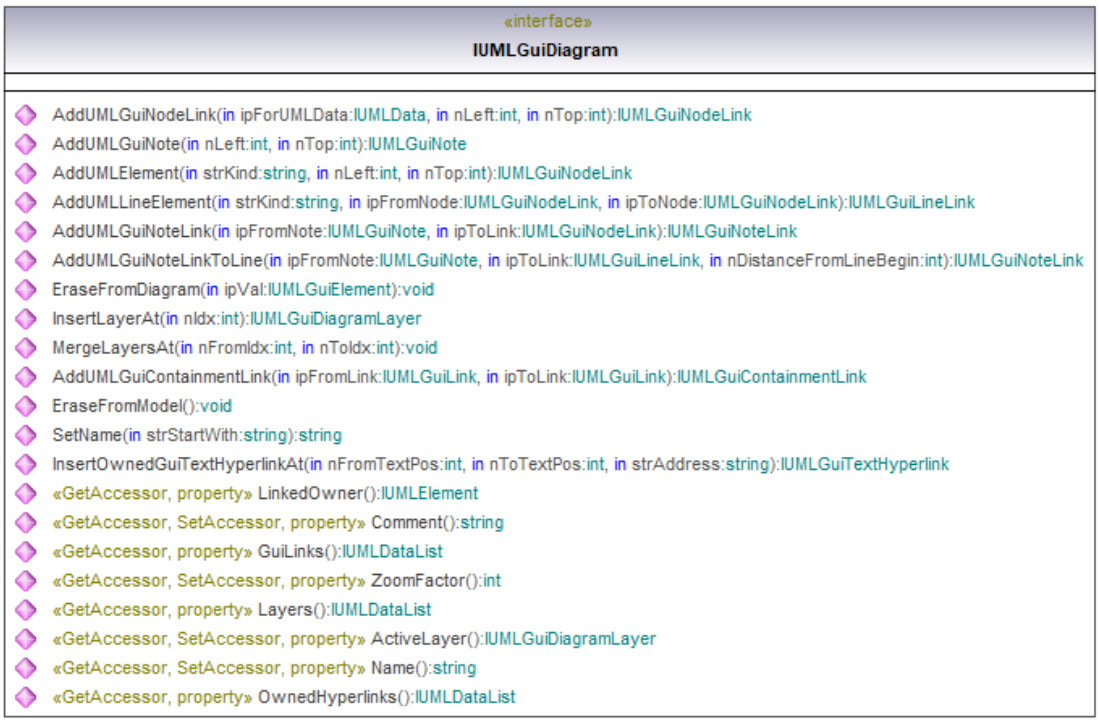
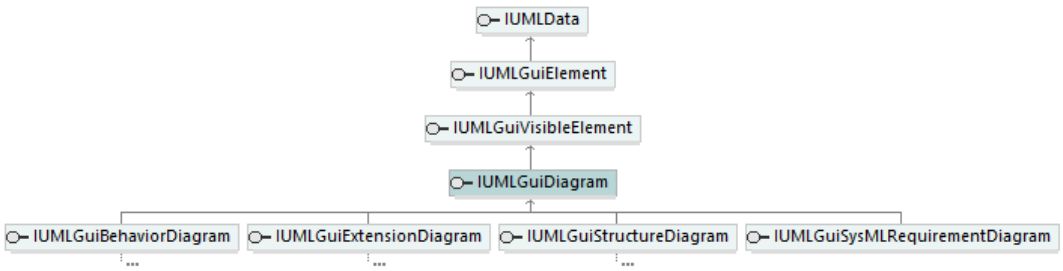
diagram

hierarchy



17.5.3.6.16 UModelAPI - IUMLGuiDiagram

Interface **IUMLGuiDiagram**

diagram	
hierarchy	
typedElements	Interface **IDiagramWindow** Interface **IDocument** Interface **IUMLDataAll** Interface **IUMLGuiRootElement** Interface **IUMLGuiSubDiagramNode** Operation **Diagram** Operation **OpenDiagram** Operation **InsertOwnedDiagramAt** **ReferencedDiagram** Operation **InsertOwnedDiagramAt** Operation **ReferencedDiagram**
documentation	Represents an UML diagram and contains all layers, nodes (represented as IUMLGuiNodeLink) and lines (represented as IUMLGuiLineLink).Use the property GuiLinks to access these.

Operation **IUMLGuiDiagram::ActiveLayer**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLGuiDiagram Layer**

Operation **IUMLGuiDiagram::AddUMLElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNodeLink**			
documentation	Adds a new UML element (e.g. IUMLClass, IUMLPackage,...) to the model and shows it with a new IUMLGuiNodeLink on the diagram.					

Operation **IUMLGuiDiagram::AddUMLGuiContainmentLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	**in**	**IUMLGuiLink**			
	ipToLink	**in**	**IUMLGuiLink**			
	return	**return**	**IUMLGuiContainmentLink**			

Operation **IUMLGuiDiagram::AddUMLGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	**in**	**IUMLData**			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNodeLink**			
documentation	Adds a new IUMLGuiNodeLink for an existing UML element (e.g. IUMLClass, IUMLPackage,...) on the diagram.					

Operation **IUMLGuiDiagram::AddUMLGuiNote**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNote**			

Operation **IUMLGuiDiagram::AddUMLGuiNoteLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote**			
	ipToLink	**in**	**IUMLGuiNodeLink**			
	return	**return**	**IUMLGuiNoteLink**			

Operation **IUMLGuiDiagram::AddUMLGuiNoteLinkToLine**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote**			
	ipToLink	**in**	**IUMLGuiLineLink**			

	nDistanceFromLineBegin	**in**	**int**
	return	**return**	**IUMLGuiNoteLink**

Operation **IUMLGuiDiagram::AddUMLLineElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	ipFromNode	**in**	**IUMLGuiNodeLink**			
	ipToNode	**in**	**IUMLGuiNodeLink**			
	return	**return**	**IUMLGuiLineLink**			
documentation	Adds a new UML line element (e.g. IUMLGeneralization, IUMLAssociation,...) to the model and shows it with a new IUMLGuiLineLink on the diagram.					

Operation **IUMLGuiDiagram::Comment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiDiagram::EraseFromDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	**IUMLGuiElement**			
	return	**return**	**void**			
documentation	Use this function to erase the element from the diagram only. Use IUMLElement::EraseFromModel to erase from the model and all diagrams.					

Operation **IUMLGuiDiagram::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLGuiDiagram::GuiLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLGuiLink which are displayed directly on this diagram. Usually, these are IUMLGuiNodeLinks and IUMLGuiLineLinks.					

Operation **IUMLGuiDiagram::InsertLayerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiDiagramLayer**			

Operation **IUMLGuiDiagram::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			

	strAddress	**in**	**string**
	return	**return**	**IUMLGuiTextHyperlink**

Operation **IUMLGuiDiagram::Layers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of all layers in the diagam. The list contains elements of type IUMLGuiDiagramLayer.					

Operation **IUMLGuiDiagram::LinkedOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLGuiDiagram::MergeLayersAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromIdx	**in**	**int**			
	nToIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiDiagram::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiDiagram::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLGuiDiagram::SetName**

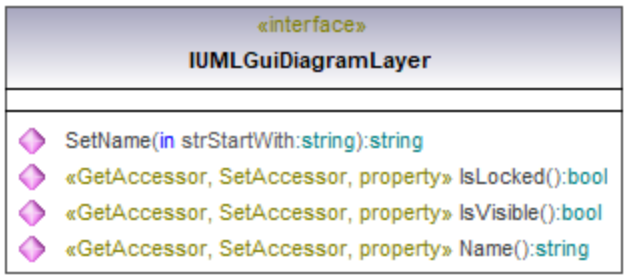
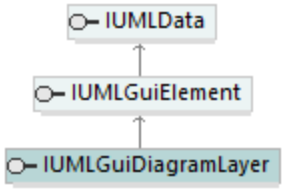
parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			

Operation **IUMLGuiDiagram::ZoomFactor**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

17.5.3.6.17 UModelAPI - IUMLGuiDiagramLayer

Interface **IUMLGuiDiagramLayer**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiDiagram** Interface **IUMLGuiLink** Operation **ActiveLayer** **InsertLayerAt** **Layer** Operation **ActiveLayer** **InsertLayerAt** Operation **Layer**
documentation	Represents a layer on an IUMLGuiDiagram. Makes it possible to group elements on a diagram into categories, to lock/unlock them and to make them visible or invisible.

Operation **IUMLGuiDiagramLayer::IsLocked**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLGuiDiagramLayer::IsVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLGuiDiagramLayer::Name**

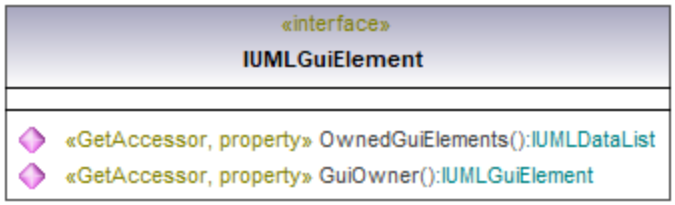
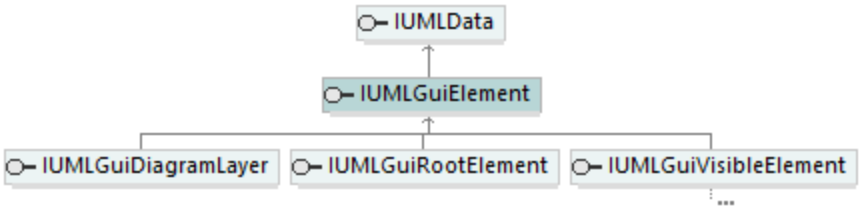
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiDiagramLayer::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			

17.5.3.6.18 UModelAPI - IUMLGuiElement

Interface **IUMLGuiElement**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiDiagram** Interface **IUMLGuiElement** Interface **IUMLGuiLineLink** Operation **EraseFromDiagram** **GuiOwner** **LineBegin** **LineEnd** Operation **EraseFromDiagram** Operation **GuiOwner** Operation **LineBegin** **LineEnd**
documentation	The base class for all graphical objects.

Operation **IUMLGuiElement::GuiOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			

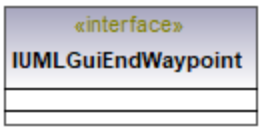
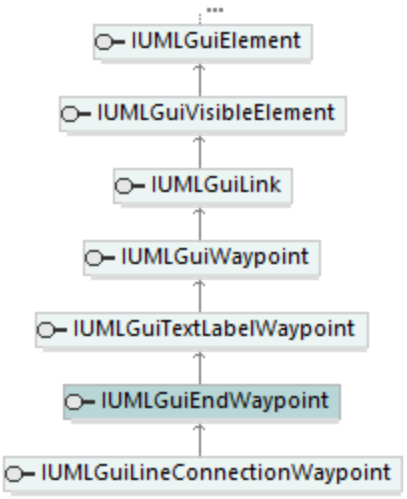
Operation **IUMLGuiElement::OwnedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	Returns a derived list of all owned Gui elements. All elements in this list are a subtype if IUMLGuiElement.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

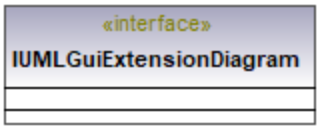
17.5.3.6.19 UModelAPI - IUMLGuiEndWaypoint

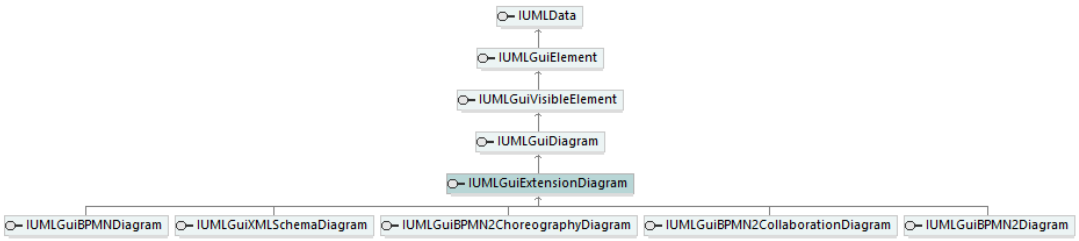
Interface **IUMLGuiEndWaypoint**

diagram	
hierarchy	
documentation	A special waypoint which only occurrs at the end or the begin of a line represented by a IUMLGuiLineLink.

17.5.3.6.20 UModelAPI - IUMLGuiExtensionDiagram

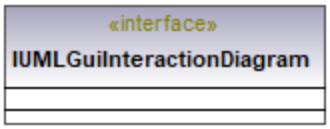
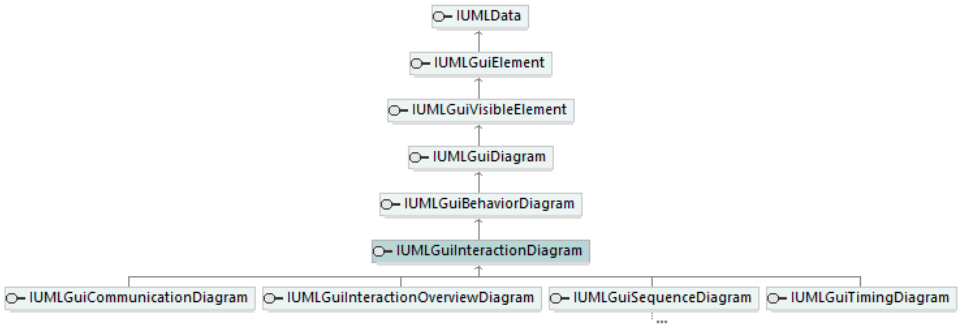
Interface **IUMLGuiExtensionDiagram**

diagram	

hierarchy	
documentation	This diagram type is the base for all UModel specific extension diagrams (for example BPMN diagrams, XML Schema Diagrams) and not a diagram type for itself.

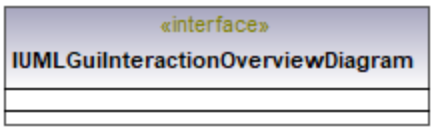
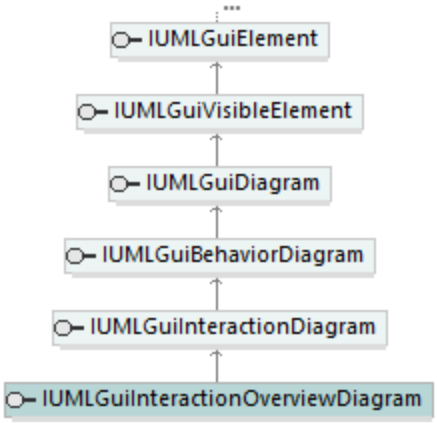
17.5.3.6.21 UModelAPI - IUMLGuiInteractionDiagram

Interface **IUMLGuiInteractionDiagram**

diagram	
hierarchy	

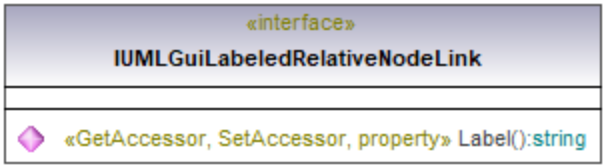
17.5.3.6.22 UModelAPI - IUMLGuiInteractionOverviewDiagram

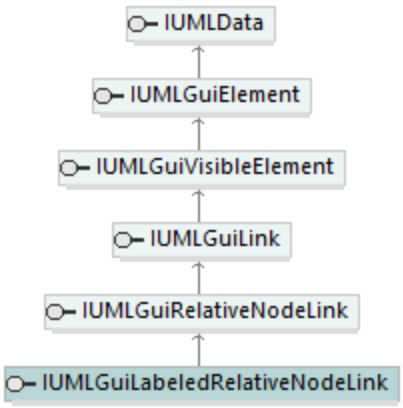
Interface **IUMLGuiInteractionOverviewDiagram**

diagram	
hierarchy	

17.5.3.6.23 UModelAPI - IUMLGuiLabeledRelativeNodeLink

Interface **IUMLGuiLabeledRelativeNodeLink**

diagram	

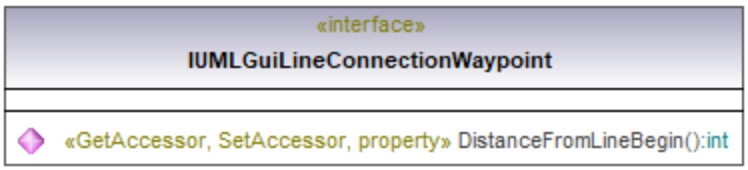
hierarchy	
documentation	This special gui link is used for elements which are relative to another node and have a label, for example for the names of Messages on Communication diagrams.

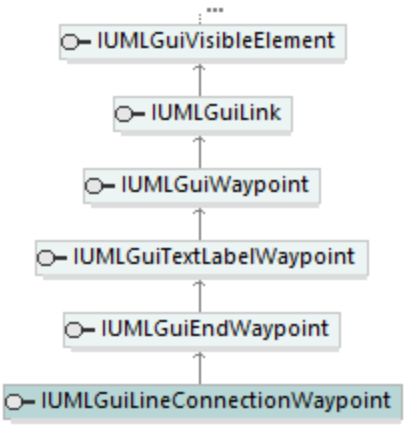
Operation **IUMLGuiLabeledRelativeNodeLink::Label**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.3.6.24 UModelAPI - IUMLGuiLineConnectionWaypoint

Interface **IUMLGuiLineConnectionWaypoint**

diagram	

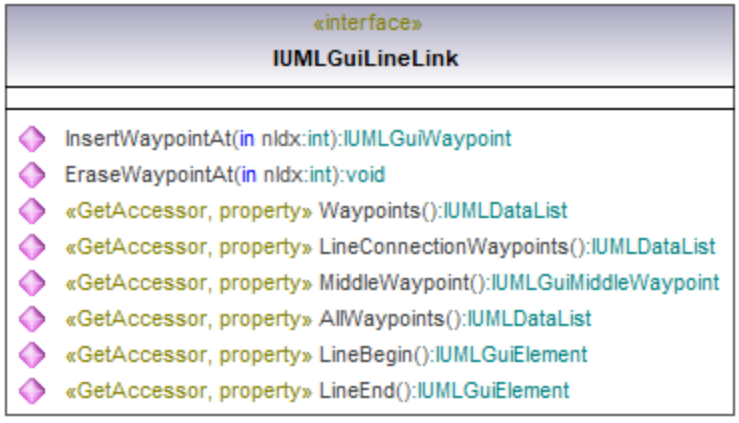
hierarchy	
documentation	This special waypoint marks the part of a line where it is connected to another line. For example, when drawing a noteLink from a note to a line on a diagram in UModel, a waypoint of this type is created where the noteLink connects to the target line. Using the DistanceFromLineBegin property, the waypoint sets a floating fixed position for itself on the line.

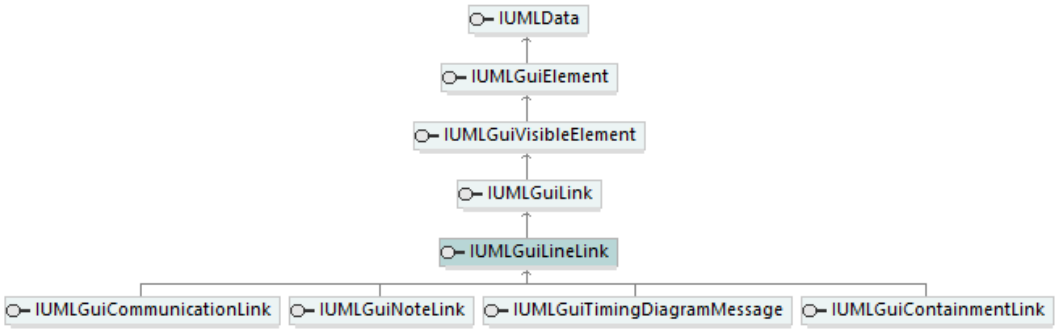
Operation **IUMLGuiLineConnectionWaypoint::DistanceFromLineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

17.5.3.6.25 UModelAPI - IUMLGuiLineLink

Interface **IUMLGuiLineLink**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiDiagram** Operation **AddUMLGuiNoteLinkToLine** **AddUMLLineElement** Operation **AddUMLGuiNoteLinkToLine** **AddUMLLineElement**
documentation	This interface represents a line on a diagram. There are some special lines deriving from this interface available as well. A line is composed of multiple but at least 2 waypoints which are connected to each other, which can be accessed using the AllWaypoints property. Two of these waypoints are usually of type IUMLGuiEndWaypoint. There may be also a Middlewaypoint accessible with the property MiddleWaypoint which can be used for example to access textlabels and a LineConnectionWaypoint when the line is connected to another line, like to a IUMLGuiNoteLink. The LineBegin property refers the first object and LineEnd property refers the second graphical object which the line connects.

Operation **IUMLGuiLineLink::AllWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A derived list of all waypoints which are part of this line. All elements in this list are of type (or subtype of) IUMLGuiWaypoint.					

Operation **IUMLGuiLineLink::EraseWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiLineLink::InsertWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiWaypoint**			

Operation **IUMLGuiLineLink::LineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			
documentation	A reference to the first object, where the line starts.					

Operation **IUMLGuiLineLink::LineConnectionWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of all waypoints which connect the line with other lines. All elements in this list are of type (or subtype of) IUMLGuiWaypoint.					

Operation **IUMLGuiLineLink::LineEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			
documentation	A reference to the second object, where the line ends.					

Operation **IUMLGuiLineLink::MiddleWaypoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiMiddleWaypoint**			

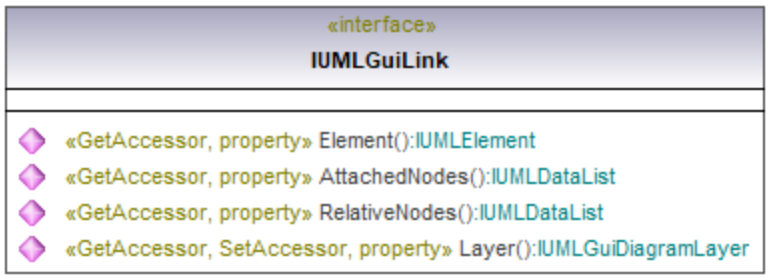
Operation **IUMLGuiLineLink::Waypoints**

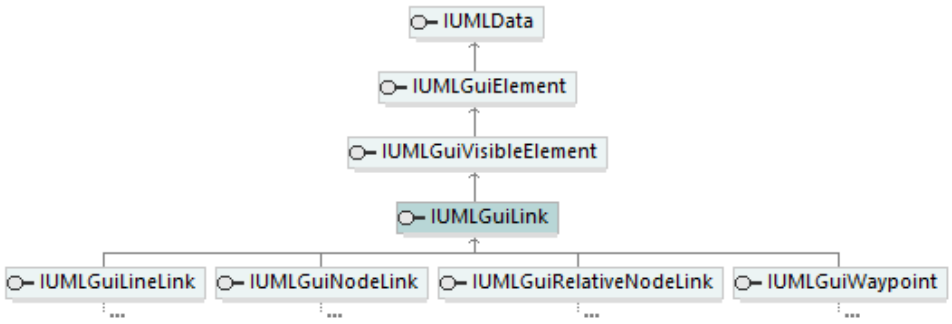
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of all waypoints which form the vertices of this line. All elements in this list are of type (or subtype of) IUMLGuiWaypoint.					

17.5.3.6.26 UModelAPI - IUMLGuiLink

Interface **IUMLGuiLink**

diagram



hierarchy	
typedElements	Interface **IDiagramWindow** Interface **IUMLDataAll** Interface **IUMLGuiAttachedNode** Interface **IUMLGuiCommunicationDiagram** Interface **IUMLGuiDiagram** Operation **FocusedGuiElement** **ScrollToGuiElement** **SelectGuiElement** Operation **AddUMLGuiContainmentLink** **AttachedTo** Operation **AttachedTo** Operation **AddUMLGuiCommunicationLink** Operation **AddUMLGuiContainmentLink**
documentation	A GuiLink represents a graphical object on a diagram which is connected to an element from the UML (like a Class, an Interface or a Lifeline). This connected object can be accessed using the Element property. IUMLGuiLinks further can have attached nodes (IUMLGuiAttachedNode) which appear when necessary, like Tagged Values and are associated with a Layer on the diagram.

Operation **IUMLGuiLink::AttachedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	Returns a derived list of all attached nodes of this element. All elements in this list are of type (or subtype) of IUMLGuiAttachedNode.					

Operation **IUMLGuiLink::Element**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLGuiLink::Layer**

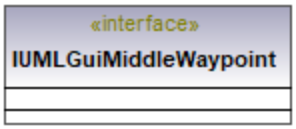
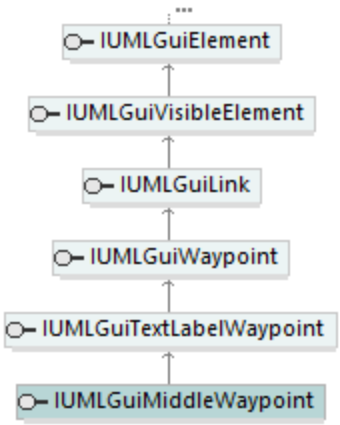
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagramLayer**			

Operation **IUMLGuiLink::RelativeNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	Returns a list of relative nodes to this gui link. The list contains only elements of type (or subtype of) IUMLGuiRelativeNodeLink.					

17.5.3.6.27 UModelAPI - IUMLGuiMiddleWaypoint

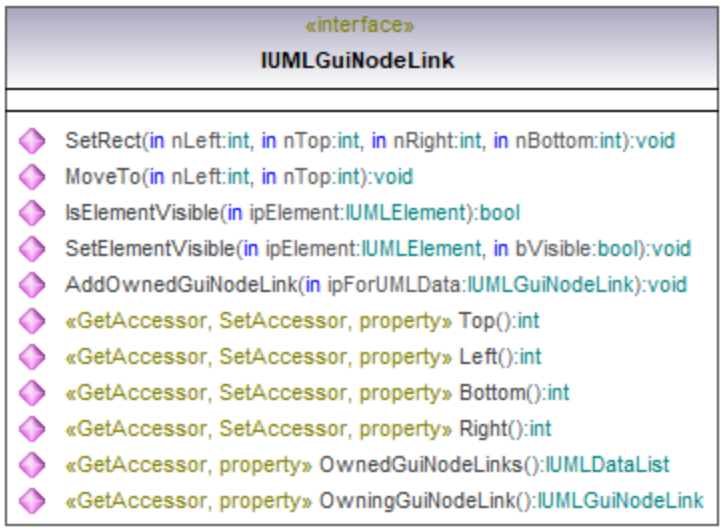
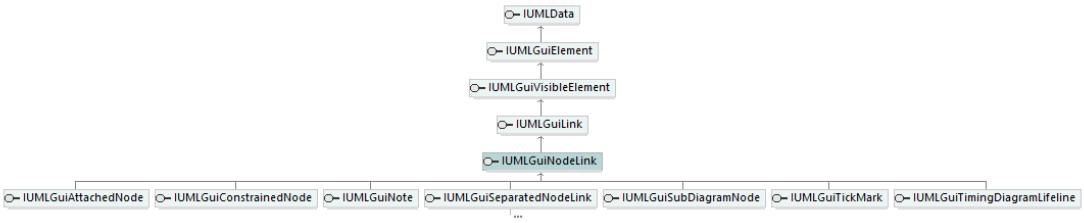
Interface **IUMLGuiMiddleWaypoint**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiLineLink** Operation **MiddleWaypoint** Operation **MiddleWaypoint**
documentation	A middle waypoint is a special waypoint on a line (IUMLGuiLineLink) which appears in the center of the line and can have text labels attached to it.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.28 UModelAPI - IUMLGuiNodeLink

Interface **IUMLGuiNodeLink**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** — Operation **AddOwnedGuiNodeLink**, **AddUMLElement**, **AddUMLGuiNodeLink**, **AddUMLGuiNoteLink**, **AddUMLLineElement**, **OwningGuiNodeLink** Interface **IUMLGuiDiagram** — Operation **AddUMLElement**, **AddUMLGuiNodeLink**, **AddUMLGuiNoteLink**, **AddUMLLineElement** Interface **IUMLGuiNodeLink** — Operation **AddOwnedGuiNodeLink**, **OwningGuiNodeLink**
documentation	A GuiNodeLink represents a graphical object on a diagram which usually represents an element from the UML (for example a Class, an Interface or a Lifeline). It has a position defined by a rectangle which can be positioned freely on the diagram. A GuiNodeLink can itself contain other GuiNodeLinks, for example when displaying a big state in a state machine diagam which contains other, smaller substates. Additionally, if the GuiNodeLink displays cells on it, like for example operations and properties on a class, it can store for each element shown if the element should be visible or not. Use the SetElementVisible and IsElementVisible functions for this.

Operation **IUMLGuiNodeLink::AddOwnedGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default

	ipForUMLData	**in**	**IUMLGuiNodeLink**
	return	**return**	**void**

Operation **IUMLGuiNodeLink::Bottom**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiNodeLink::IsElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipElement	**in**	**IUMLElement**			
	return	**return**	**bool**			

Operation **IUMLGuiNodeLink::Left**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiNodeLink::MoveTo**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiNodeLink::OwnedGuiNodeLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	Returns a list of all owned gui node links, all nodes which are directly contained in this node. All elements in this list are of type (or subtype of) IUMLGuiLink.					

Operation **IUMLGuiNodeLink::OwningGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiNodeLink**			

Operation **IUMLGuiNodeLink::Right**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiNodeLink::SetElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipElement	**in**	**IUMLElement**			
	bVisible	**in**	**bool**			
	return	**return**	**void**			

Operation **IUMLGuiNodeLink::SetRect**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			

	nTop	**in**	**int**
	nRight	**in**	**int**
	nBottom	**in**	**int**
	return	**return**	**void**

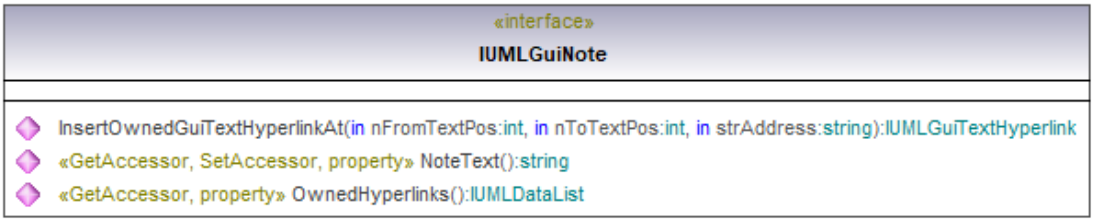
Operation **IUMLGuiNodeLink::Top**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

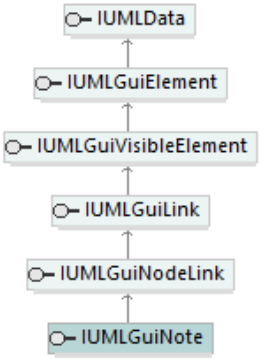
17.5.3.6.29 UModelAPI - IUMLGuiNote

Interface **IUMLGuiNote**

diagram



hierarchy



typedElements

Interface **IUMLDataAll**	Operation **AddUMLGuiNote** **AddUMLGuiNoteLink** **AddUMLGuiNoteLinkToLine**
Interface **IUMLGuiDiagram**	Operation **AddUMLGuiNote** **AddUMLGuiNoteLink** **AddUMLGuiNoteLinkToLine**

documentation

A IUMLGuiNote is the graphical object resembling a note on UModel diagrams displaying a text comment.
It provides access to the note text and a list of hyperlinks in this text. These hyperlinks are nothing more than a list of URLs together with an begin and end number referencing positions in the text. Any text between a such a begin and end position is displayed as hyperlink and triggers UModel to open the URL when clicked.

Operation **IUMLGuiNote::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLGuiTextHyperlink**			

Operation **IUMLGuiNote::NoteText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

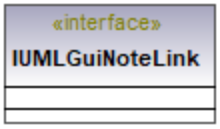
Operation **IUMLGuiNote::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

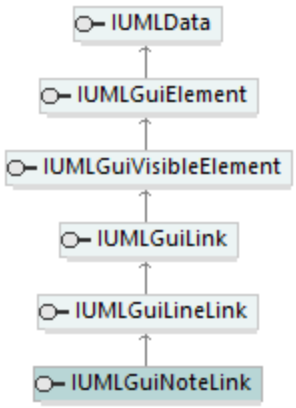
17.5.3.6.30 UModelAPI - IUMLGuiNoteLink

Interface **IUMLGuiNoteLink**

diagram



hierarchy



typedElements

Interface IUMLDataAll

Interface IUMLGuiDiagram

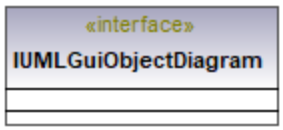
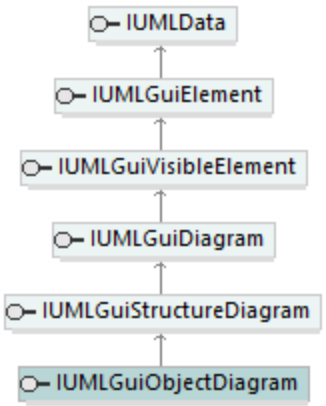
Operation AddUMLGuiNoteLink
AddUMLGuiNoteLinkToLine

Operation AddUMLGuiNoteLink
AddUMLGuiNoteLinkToLine

documentation	A notelink is a special IUMLGuiLineLink which connects a IUMLGuiNote with another IUMLGuiLink element. It is displayed as a dotted line.

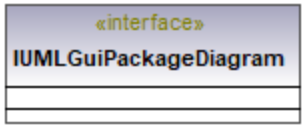
17.5.3.6.31 UModelAPI - IUMLGuiObjectDiagram

Interface **IUMLGuiObjectDiagram**

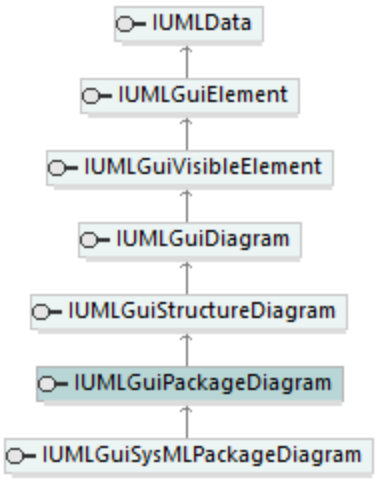
diagram	
hierarchy	

17.5.3.6.32 UModelAPI - IUMLGuiPackageDiagram

Interface **IUMLGuiPackageDiagram**

diagram	

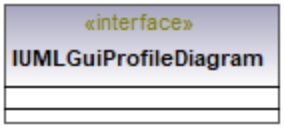
hierarchy



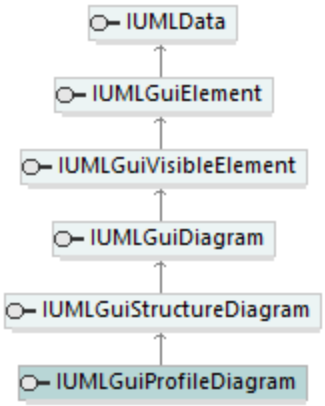
17.5.3.6.33 UModelAPI - IUMLGuiProfileDiagram

Interface **IUMLGuiProfileDiagram**

diagram

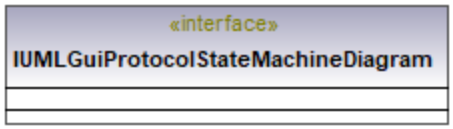
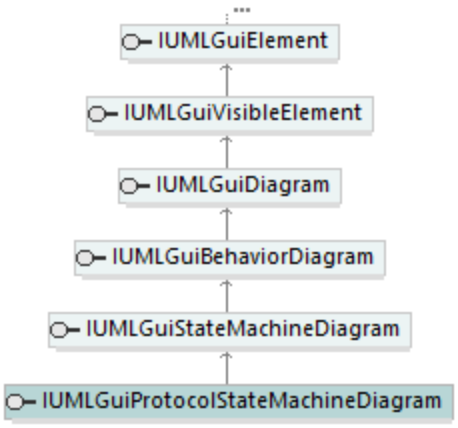


hierarchy



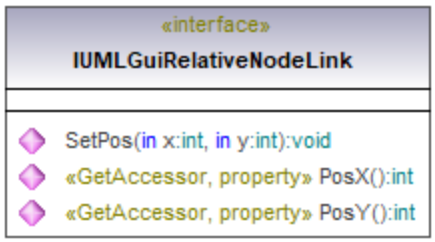
17.5.3.6.34 UModelAPI - IUMLGuiProtocolStateMachineDiagram

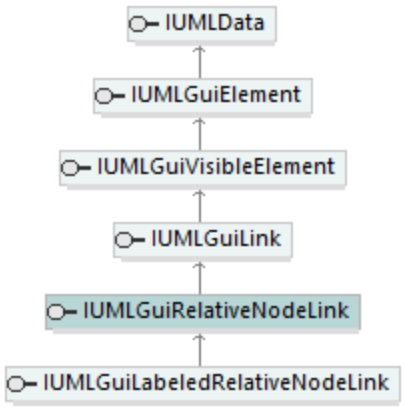
Interface **IUMLGuiProtocolStateMachineDiagram**

diagram	
hierarchy	

17.5.3.6.35 UModelAPI - IUMLGuiRelativeNodeLink

Interface **IUMLGuiRelativeNodeLink**

diagram	

hierarchy	
documentation	This gui link is used for elements which are positioned relative to another node. For example the names of Messages on Communication diagrams use a specialization of this interface.

Operation **IUMLGuiRelativeNodeLink::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiRelativeNodeLink::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

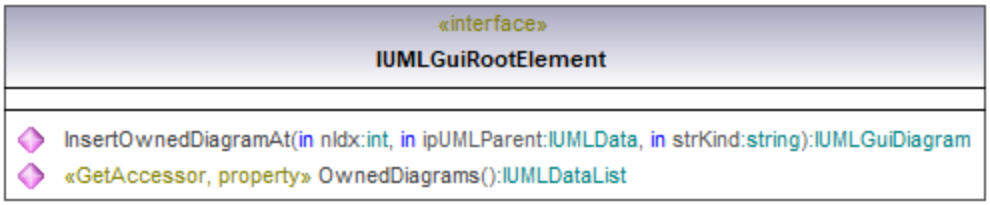
Operation **IUMLGuiRelativeNodeLink::SetPos**

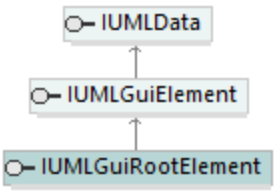
parameter	name	direction	type	type modifier	multiplicity	default
	x	**in**	**int**			
	y	**in**	**int**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.36 UModelAPI - IUMLGuiRootElement

Interface **IUMLGuiRootElement**

diagram	

hierarchy	
typedElements	Interface **IDocument** Operation **GuiRoot**
documentation	This is the root interface for all graphical objects and contains all diagrams which exists in the UModel project.

Operation **IUMLGuiRootElement::InsertOwnedDiagramAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipUMLParent	**in**	**IUMLData**			
	strKind	**in**	**string**			
	return	**return**	**IUMLGuiDiagram**			

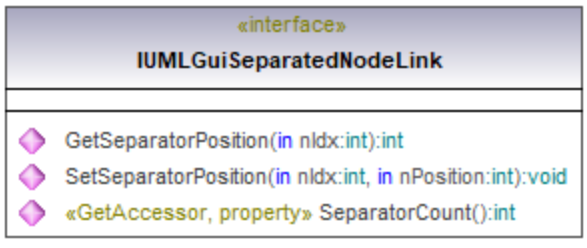
Operation **IUMLGuiRootElement::OwnedDiagrams**

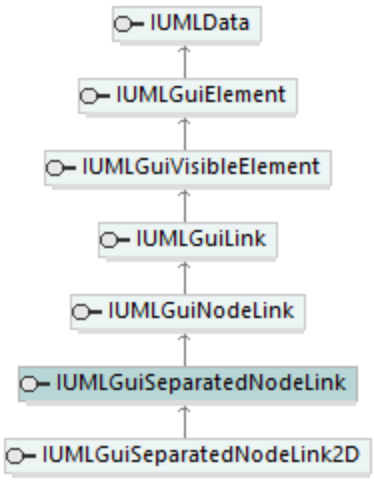
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	Returns a list of all diagrams in this UModel project. All elements in this list are of type (or subtype of) IUMLGuiDiagram.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.37 UModelAPI - IUMLGuiSeparatedNodeLink

Interface **IUMLGuiSeparatedNodeLink**

diagram	

hierarchy	
documentation	This node link represents a graphical object on a UModel diagram which can be separated into two or more parts by one or more either horizontal or vertical lines. For each line, the position of the separator is stored in this node. This node type is used for example by CombinedFragments, ActivityPartitions and States with regions.

Operation **IUMLGuiSeparatedNodeLink::GetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiSeparatedNodeLink::SeparatorCount**

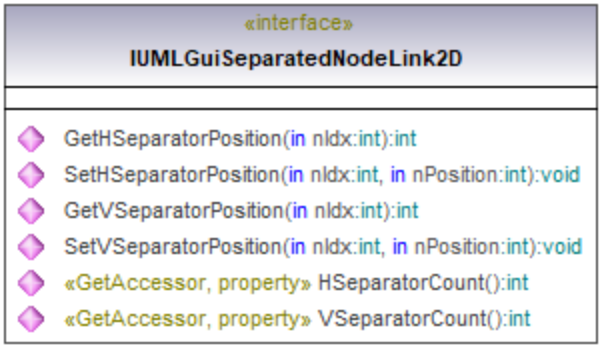
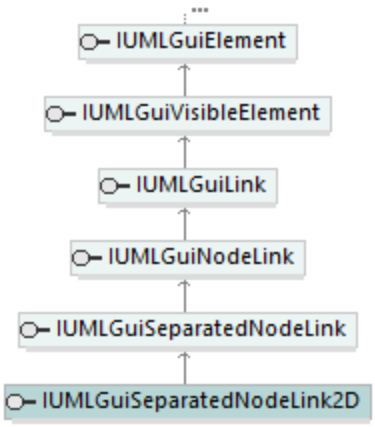
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiSeparatedNodeLink::SetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

17.5.3.6.38 UModelAPI - IUMLGuiSeparatedNodeLink2D

Interface **IUMLGuiSeparatedNodeLink2D**

diagram	
hierarchy	
documentation	This node link represents a graphical object on a UModel diagram which can be separated into parts by one or more horizontal or vertical lines, but in contrast to IUMLGuiSeparatedNodeLink, the node can be subdivided vertically and horizontally at the same time. For each vertical or horizontal separation line, the position of the separator is stored in this node. This node type is used for example by ActivityPartitions.

Operation **IUMLGuiSeparatedNodeLink2D::GetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiSeparatedNodeLink2D::GetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiSeparatedNodeLink2D::HSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**int**

Operation **IUMLGuiSeparatedNodeLink2D::SetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiSeparatedNodeLink2D::SetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiSeparatedNodeLink2D::VSeparatorCount**

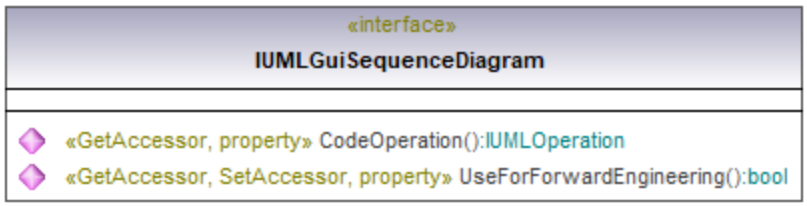
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

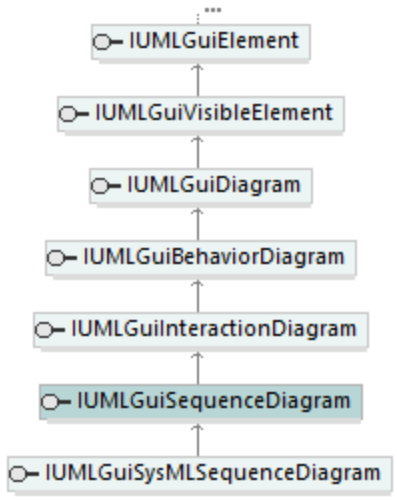
17.5.3.6.39 UModelAPI - IUMLGuiSequenceDiagram

Interface **IUMLGuiSequenceDiagram**

diagram



hierarchy	



Operation **IUMLGuiSequenceDiagram::CodeOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

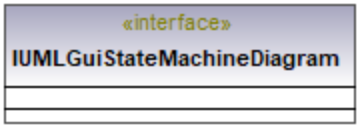
Operation **IUMLGuiSequenceDiagram::UseForForwardEngineering**

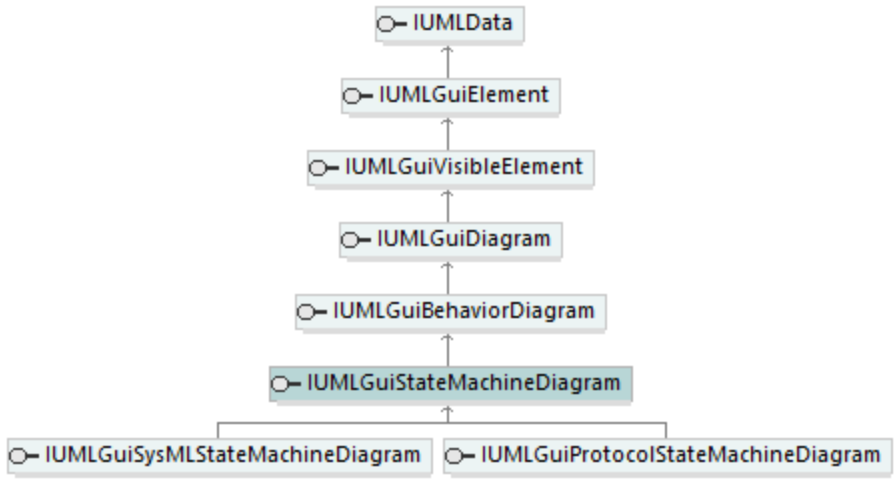
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.3.6.40 UModelAPI - IUMLGuiStateMachineDiagram

Interface **IUMLGuiStateMachineDiagram**

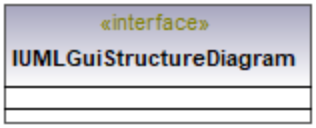
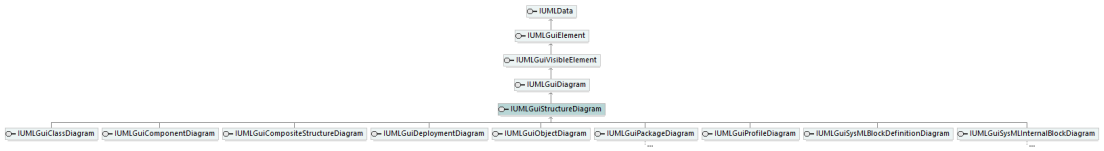
diagram	



hierarchy	

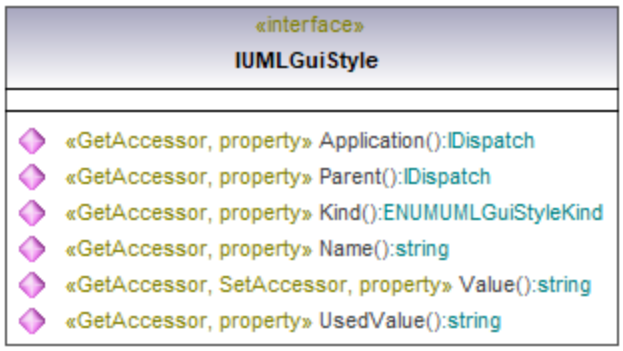
17.5.3.6.41 UModelAPI - IUMLGuiStructureDiagram

Interface **IUMLGuiStructureDiagram**

diagram	
hierarchy	

17.5.3.6.42 UModelAPI - IUMLGuiStyle

Interface **IUMLGuiStyle**

diagram	
typedElements	Interface **IUMLGuiStyles** Operation **GetStyle Item**

Operation **IUMLGuiStyle::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLGuiStyle::Kind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLGuiStyleKind**			

Operation **IUMLGuiStyle::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiStyle::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLGuiStyle::UsedValue**

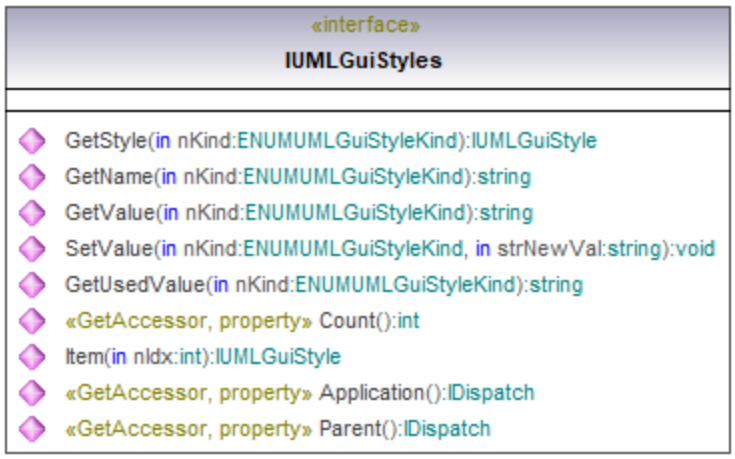
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiStyle::Value**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.3.6.43 UModelAPI - IUMLGuiStyles

Interface **IUMLGuiStyles**

diagram	
typedElements	Interface **IDocument** Interface **IUMLDataAll** Interface **IUMLGuiVisibleElement** Interface **IUMLStereotype** Operation **ElementFamilyStyles** **LineStyles** **NodeStyles** **ProjectStyles** Operation **StereotypedElementStyles** **Styles** Operation **Styles** Operation **StereotypedElementStyles**

Operation **IUMLGuiStyles::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLGuiStyles::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiStyles::GetName**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind**			
	return	**return**	**string**			

Operation **IUMLGuiStyles::GetStyle**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind**			
	return	**return**	**IUMLGuiStyle**			

Operation **IUMLGuiStyles::GetUsedValue**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind**			

	return	**return**	**string**

Operation **IUMLGuiStyles::GetValue**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind**			
	return	**return**	**string**			

Operation **IUMLGuiStyles::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiStyle**			

Operation **IUMLGuiStyles::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

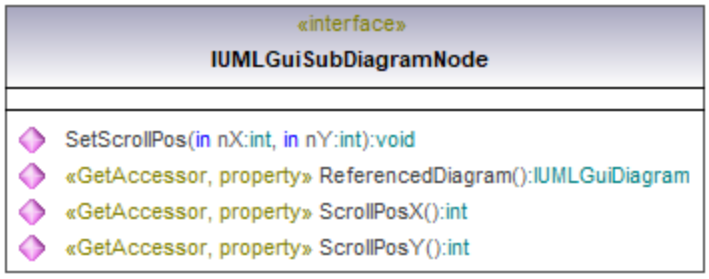
Operation **IUMLGuiStyles::SetValue**

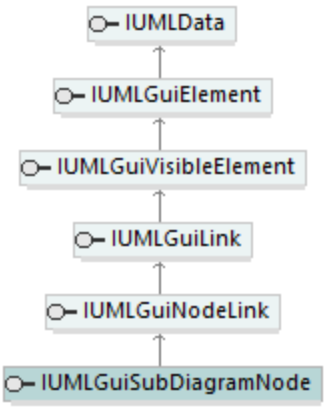
parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

17.5.3.6.44 UModelAPI - IUMLGuiSubDiagramNode

Interface **IUMLGuiSubDiagramNode**

diagram



hierarchy	
documentation	The sub diagram node represents a node link on a diagram which again includes another diagram. This is used for example on interaction overview diagrams to display sequence, communication and timing diagrams inside nodes. The property ReferencedDiagram controls the diagrams which is shown inside the node.

Operation **IUMLGuiSubDiagramNode::ReferencedDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagram**			

Operation **IUMLGuiSubDiagramNode::ScrollPosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiSubDiagramNode::ScrollPosY**

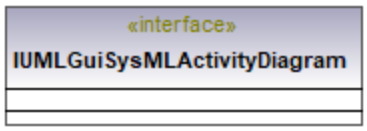
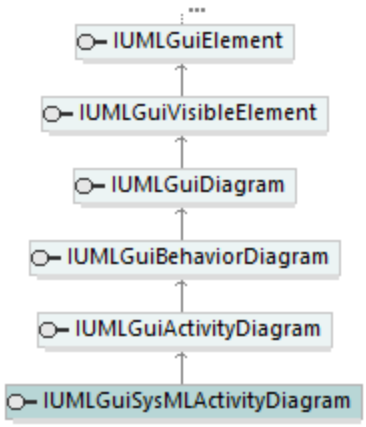
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiSubDiagramNode::SetScrollPos**

parameter	name	direction	type	type modifier	multiplicity	default
	nX	**in**	**int**			
	nY	**in**	**int**			
	return	**return**	**void**			

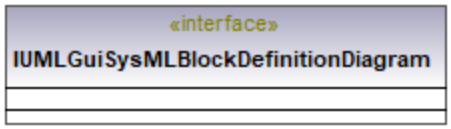
17.5.3.6.45 UModelAPI - IUMLGuiSysMLActivityDiagram

Interface **IUMLGuiSysMLActivityDiagram**

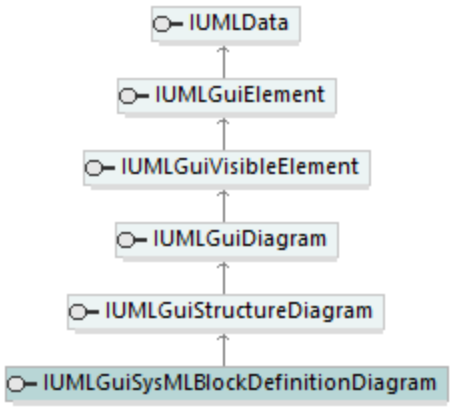
diagram	
hierarchy	

17.5.3.6.46 UModelAPI - IUMLGuiSysMLBlockDefinitionDiagram

Interface **IUMLGuiSysMLBlockDefinitionDiagram**

diagram	

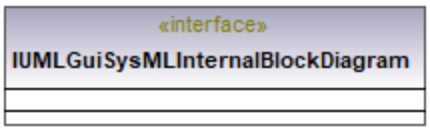
hierarchy



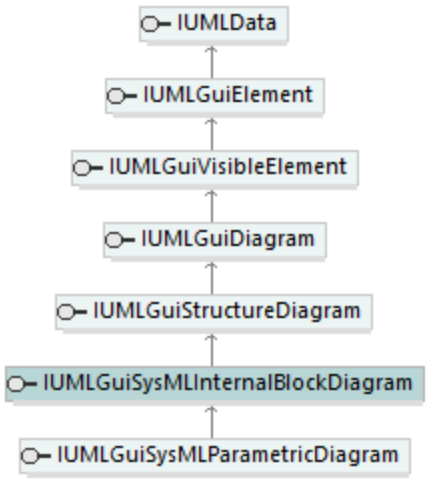
17.5.3.6.47 UModelAPI - IUMLGuiSysMLInternalBlockDiagram

Interface **IUMLGuiSysMLInternalBlockDiagram**

diagram

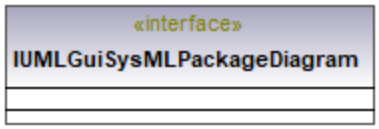
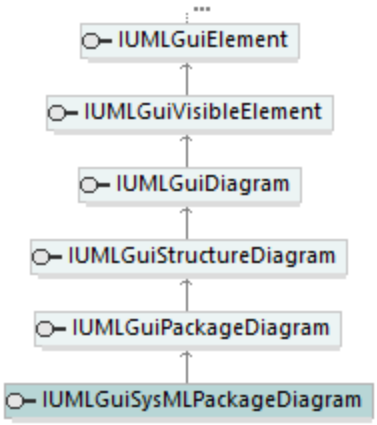


hierarchy



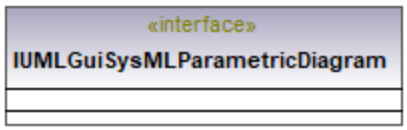
17.5.3.6.48 UModelAPI - IUMLGuiSysMLPackageDiagram

Interface **IUMLGuiSysMLPackageDiagram**

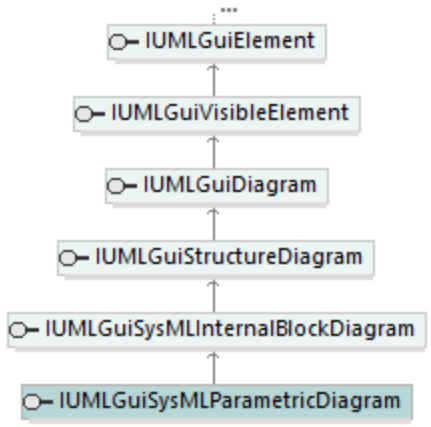
diagram	
hierarchy	

17.5.3.6.49 UModelAPI - IUMLGuiSysMLParametricDiagram

Interface **IUMLGuiSysMLParametricDiagram**

diagram	

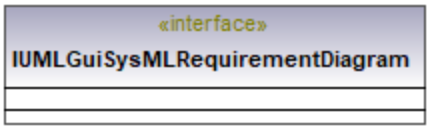
hierarchy



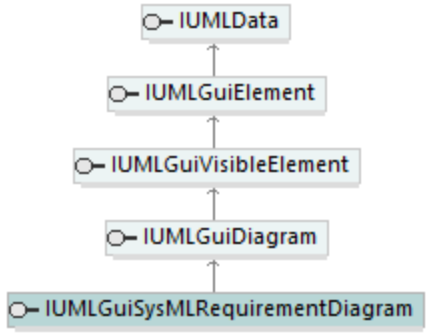
17.5.3.6.50 UModelAPI - IUMLGuiSysMLRequirementDiagram

Interface **IUMLGuiSysMLRequirementDiagram**

diagram

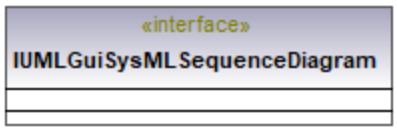
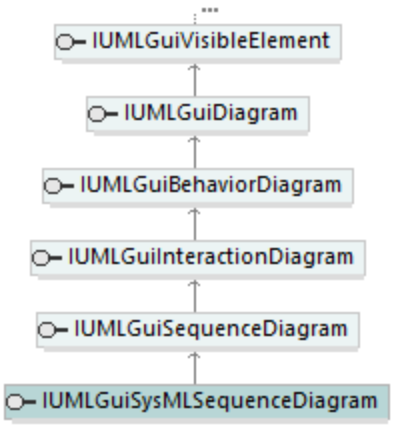


hierarchy



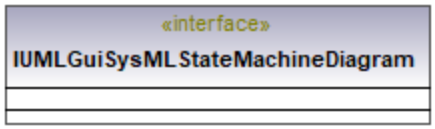
17.5.3.6.51 UModelAPI - IUMLGuiSysMLSequenceDiagram

Interface **IUMLGuiSysMLSequenceDiagram**

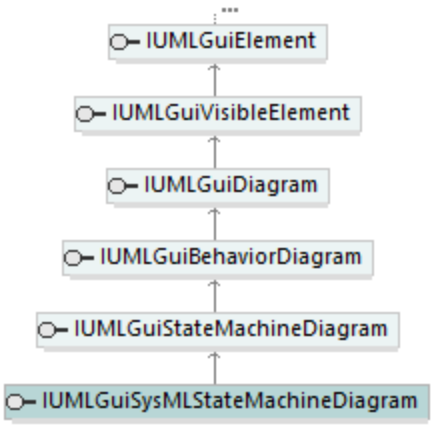
diagram	
hierarchy	

17.5.3.6.52 UModelAPI - IUMLGuiSysMLStateMachineDiagram

Interface **IUMLGuiSysMLStateMachineDiagram**

diagram	

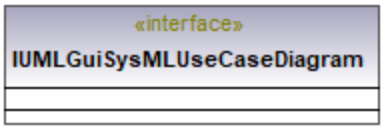
hierarchy



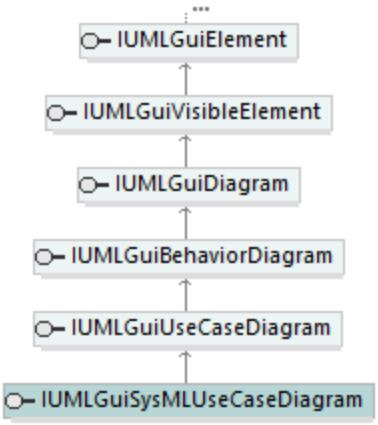
17.5.3.6.53 UModelAPI - IUMLGuiSysMLUseCaseDiagram

Interface **IUMLGuiSysMLUseCaseDiagram**

diagram

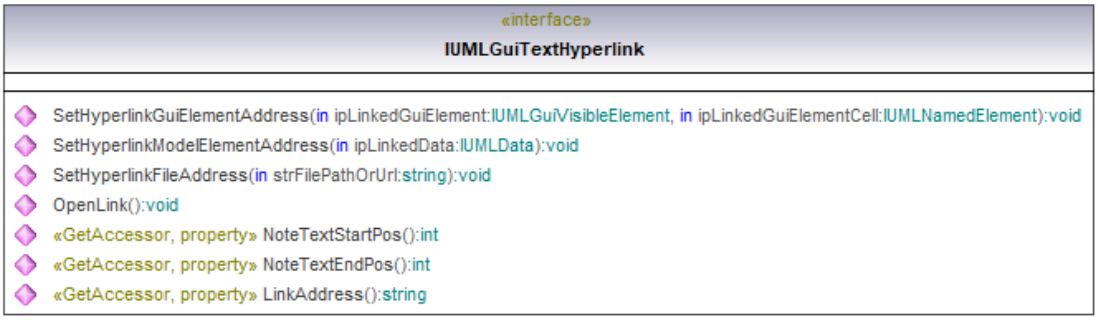
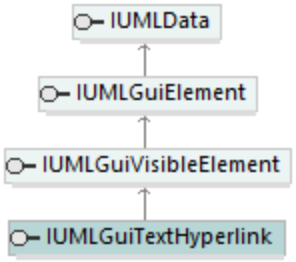


hierarchy



17.5.3.6.54 UModelAPI - IUMLGuiTextHyperlink

Interface **IUMLGuiTextHyperlink**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiDiagram** Interface **IUMLGuiNote** Operation **InsertOwnedGuiTextHyperlinkAt** Operation **InsertOwnedGuiTextHyperlinkAt** Operation **InsertOwnedGuiTextHyperlinkAt**
documentation	Text Hyperlinks store an URL together with a begin and an end number referencing positions in some text. Any text between a such a begin and end position is displayed as hyperlink and triggers UModel to open the URL when clicked. This is used in IUMLGuiNote to create hyperlinks inside of the text comment for example.

Operation **IUMLGuiTextHyperlink::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiTextHyperlink::NoteTextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTextHyperlink::NoteTextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTextHyperlink::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**void**

Operation **IUMLGuiTextHyperlink::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strFilePathOrUrl	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLGuiTextHyperlink::SetHyperlinkGuiElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedGuiElement	**in**	**IUMLGuiVisibleElement**			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLGuiTextHyperlink::SetHyperlinkModelElementAddress**

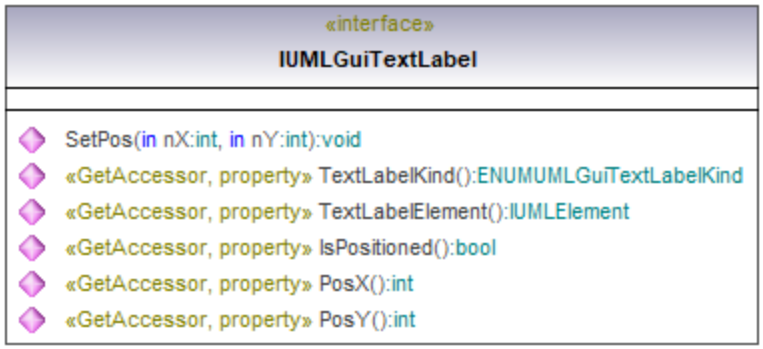
parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	**in**	**IUMLData**			
	return	**return**	**void**			

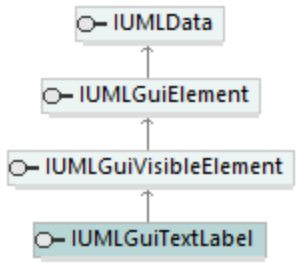
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.55 UModelAPI - IUMLGuiTextLabel

Interface **IUMLGuiTextLabel**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** Operation **GetTextLabelText** **IsTextLabelVisible** **SetTextLabelVisible** Interface **IUMLGuiTextLabelWaypoint** Operation **GetTextLabelText** **IsTextLabelVisible** **SetTextLabelVisible**
documentation	A text label is an graphical object displaying additional data at the begin, end or at the center of a line. The IUMLGuiTextLabel interface provides access to this element. The text label can reference an UML Element using the TextLabelElement property and has a TextLabelKind which affects what text is shown in the text label.

Operation **IUMLGuiTextLabel::IsPositioned**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLGuiTextLabel::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTextLabel::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTextLabel::SetPos**

parameter	name	direction	type	type modifier	multiplicity	default
	nX	**in**	**int**			
	nY	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiTextLabel::TextLabelElement**

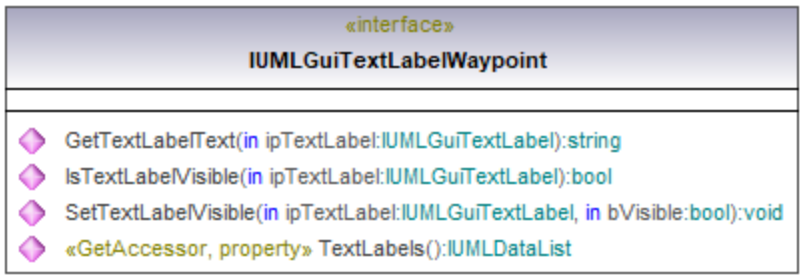
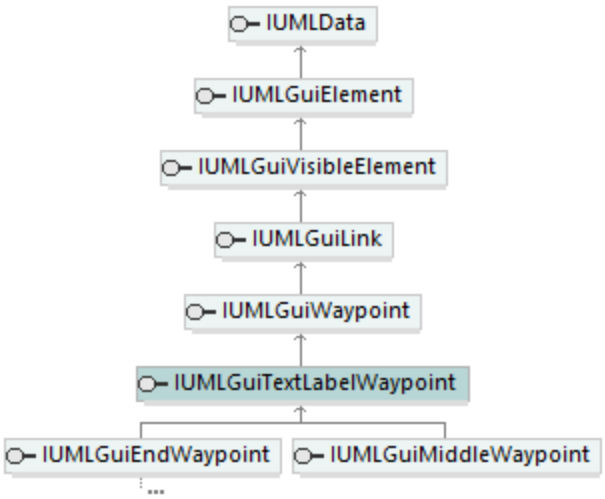
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLGuiTextLabel::TextLabelKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLGuiTextLabelKind**			

17.5.3.6.56 UModelAPI - IUMLGuiTextLabelWaypoint

Interface **IUMLGuiTextLabelWaypoint**

diagram	
hierarchy	
documentation	A text label waypoint is a special waypoint as part of a IUMLGuiLineLink which can have one or more text labels associated with it. Text label waypoints can usually appear at the begin, end or in the middle of a line. The waypoint stores not only the text labels it shows, but also the visibility of each text label.

Operation **IUMLGuiTextLabelWaypoint::GetTextLabelText**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	return	**return**	**string**			

Operation **IUMLGuiTextLabelWaypoint::IsTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	return	**return**	**bool**			

Operation **IUMLGuiTextLabelWaypoint::SetTextLabelVisible**

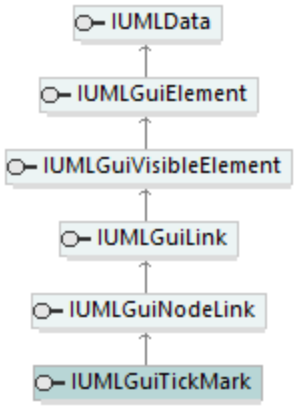
parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	bVisible	**in**	**bool**			
	return	**return**	**void**			

Operation **IUMLGuiTextLabelWaypoint::TextLabels**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	Returns a list of all text labels of this waypoint. Contains only elements of type (or subtype of) IUMLGuiTextLabel.					

17.5.3.6.57 UModelAPI - IUMLGuiTickMark

Interface **IUMLGuiTickMark**

diagram	
hierarchy	
typedElements	Interface **IUMLGuiTimingDiagram** Operation **AddUMLGuiTickMark**
documentation	A tick mark is a special graphical item appearing on the border of lifelines on timing diagrams. It represents a certain point in time and is displayed as short vertical line. It has a Value property which is displayed as text below the vertical line.

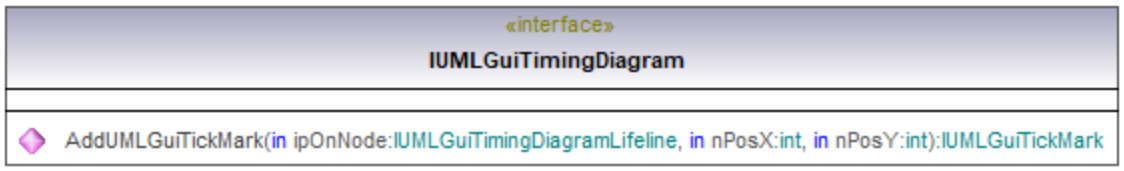
Operation **IUMLGuiTickMark::Value**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

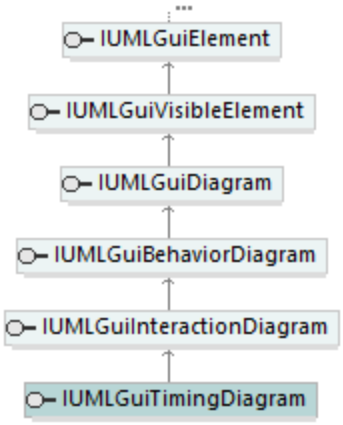
17.5.3.6.58 UModelAPI - IUMLGuiTimingDiagram

Interface **IUMLGuiTimingDiagram**

diagram



hierarchy

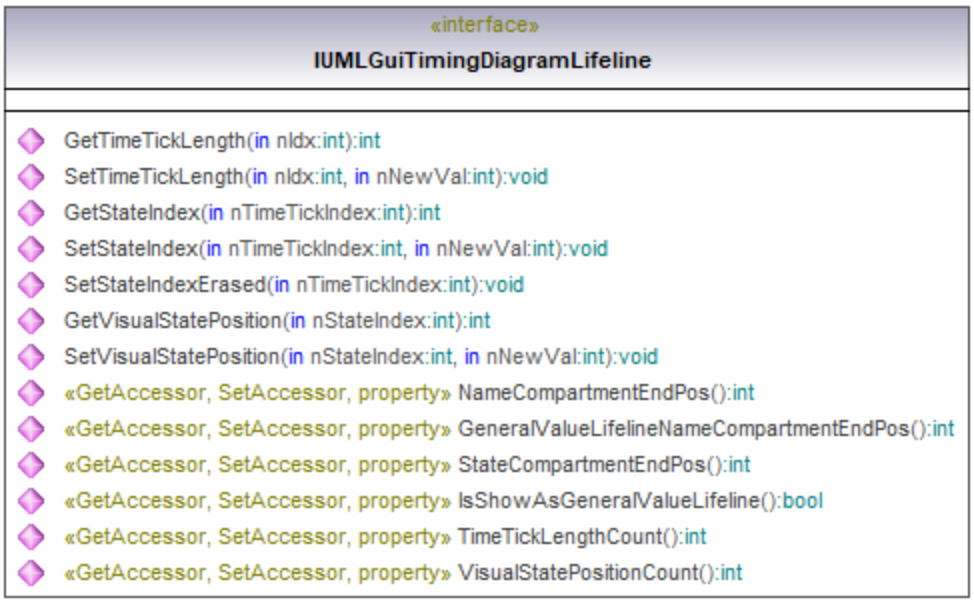
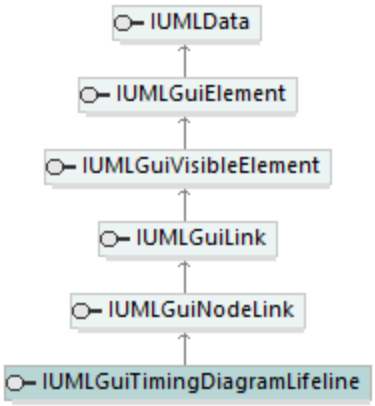


Operation **IUMLGuiTimingDiagram::AddUMLGuiTickMark**

parameter	name	direction	type	type modifier	multiplicity	default
	ipOnNode	**in**	**IUMLGuiTimingDiagramLifeline**			
	nPosX	**in**	**int**			
	nPosY	**in**	**int**			
	return	**return**	**IUMLGuiTickMark**			

17.5.3.6.59 UModelAPI - IUMLGuiTimingDiagramLifeline

Interface **IUMLGuiTimingDiagramLifeline**

diagram	
hierarchy	
typedElements	Interface **IUMLGuiTimingDiagram** Operation **AddUMLGuiTickMark**
documentation	A IUMLGuiTimingDiagramLifeline is the graphical representation of a lifeline on a timing diagram. This type of lifeline has several options to display its data and provides access to these through its numerous properties.

Operation **IUMLGuiTimingDiagramLifeline::GeneralValueLifelineNameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::GetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default

	nTimeTickIndex	**in**	**int**
	return	**return**	**int**

Operation **IUMLGuiTimingDiagramLifeline::GetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::GetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nStateIndex	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::IsShowAsGeneralValueLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLGuiTimingDiagramLifeline::NameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::SetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiTimingDiagramLifeline::SetStateIndexErased**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiTimingDiagramLifeline::SetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiTimingDiagramLifeline::SetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nStateIndex	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiTimingDiagramLifeline::StateCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::TimeTickLengthCount**

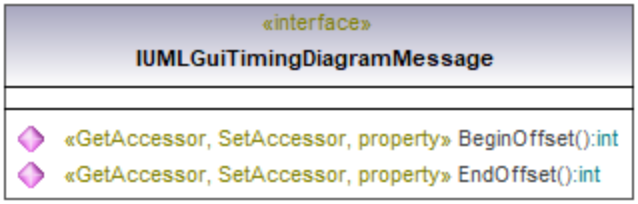
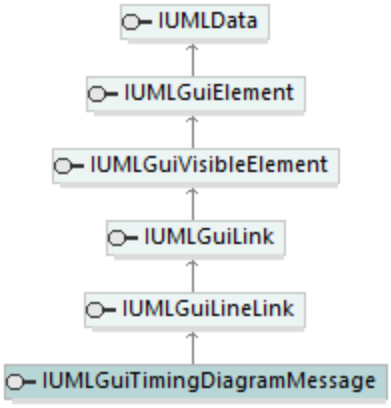
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::VisualStatePositionCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

17.5.3.6.60 UModelAPI - IUMLGuiTimingDiagramMessage

Interface **IUMLGuiTimingDiagramMessage**

diagram	
hierarchy	
documentation	A IUMLGuiTimingDiagramMessage is a line usually connecting two IUMLGuiTimingDiagramLifelines. For each lifeline on one of its ends, it stores its position from the start of the state or general value compartement.

Operation **IUMLGuiTimingDiagramMessage::BeginOffset**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramMessage::EndOffset**

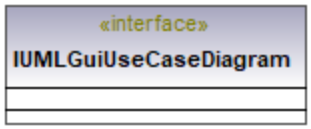
parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**int**

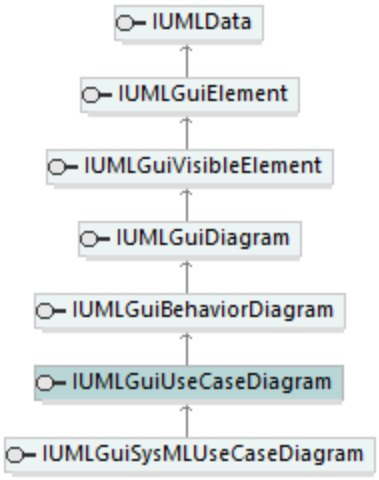
17.5.3.6.61 UModelAPI - IUMLGuiUseCaseDiagram

Interface **IUMLGuiUseCaseDiagram**

diagram



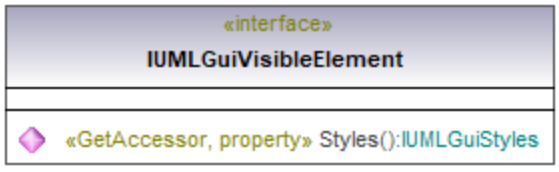
hierarchy

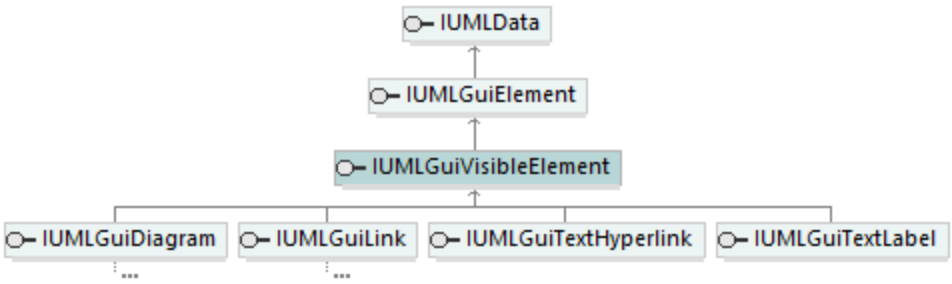


17.5.3.6.62 UModelAPI - IUMLGuiVisibleElement

Interface **IUMLGuiVisibleElement**

diagram



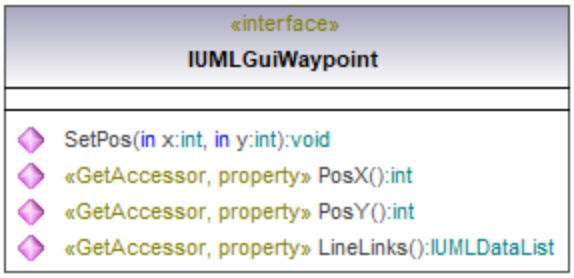
hierarchy	
typedElements	Interface **IUMLCommentTextHyperlink** Interface **IUMLDataAll** Interface **IUMLGuiTextHyperlink** Interface **IUMLHyperlink2GuiElement** Interface **IUMLNamedElement** Operation **SetHyperlinkGuiElementAddress** Operation **InsertOwnedHyperlink2GuiElementAt LinkedGuiElement SetHyperlinkGuiElementAddress** Operation **SetHyperlinkGuiElementAddress** Operation **LinkedGuiElement** Operation **InsertOwnedHyperlink2GuiElementAt**
documentation	The IUMLGuiVisibleElement is the base interface for most visible elements which can be placed on UModel diagrams. Visible elements have a style with which it is possible to infliuence its color and/or shape, depending on the actual type of the visible element.

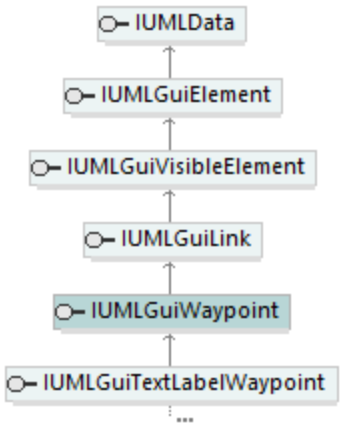
Operation **IUMLGuiVisibleElement::Styles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

17.5.3.6.63 UModelAPI - IUMLGuiWaypoint

Interface **IUMLGuiWaypoint**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiLineLink** Operation **InsertWaypointAt** Operation **InsertWaypointAt**
documentation	A IUMLGuiWaypoint is a part of a IUMLGuiLineLink which defines the position of one point of a line. There are several subtypes of this interface which for example make it possible to attach text labels or lines to a waypoint or line.

Operation **IUMLGuiWaypoint::LineLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLGuiWaypoint::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiWaypoint::PosY**

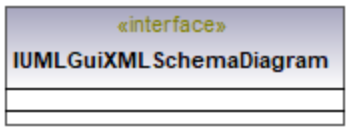
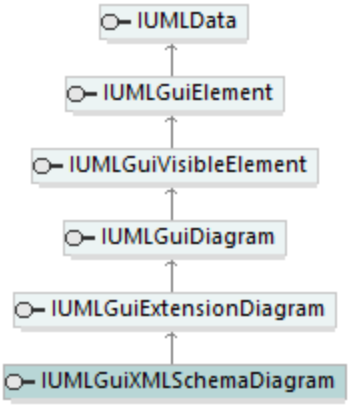
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiWaypoint::SetPos**

parameter	name	direction	type	type modifier	multiplicity	default
	x	**in**	**int**			
	y	**in**	**int**			
	return	**return**	**void**			

17.5.3.6.64 UModelAPI - IUMLGuiXMLSchemaDiagram

Interface **IUMLGuiXMLSchemaDiagram**

diagram	
hierarchy	

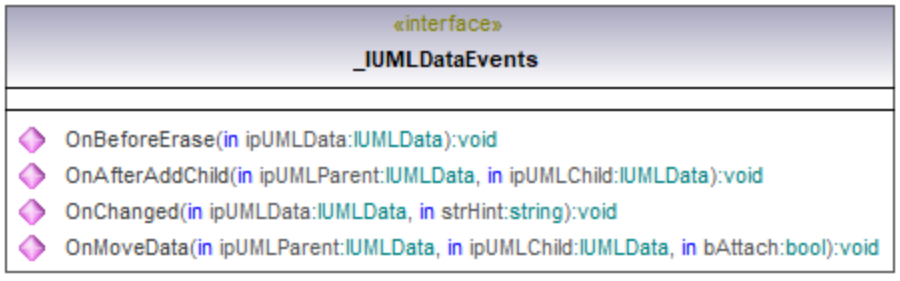
17.5.3.7 Events

This is a list of all events sent by the UModel API on `UMLData` level.

See also How to Use UMLData Events and Event Filters.

17.5.3.7.1 UModelAPI - _IUMLDataEvents

Interface **_IUMLDataEvents**

diagram	

Operation **_IUMLDataEvents::OnAfterAddChild**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLParent	**in**	**IUMLData**			
	ipUMLChild	**in**	**IUMLData**			
	return	**return**	**void**			

Operation **_IUMLDataEvents::OnBeforeErase**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	return	**return**	**void**			

Operation **_IUMLDataEvents::OnChanged**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	strHint	**in**	**string**			
	return	**return**	**void**			
documentation	**strHint** is for future use only!					

Operation **_IUMLDataEvents::OnMoveData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLParent	**in**	**IUMLData**			
	ipUMLChild	**in**	**IUMLData**			
	bAttach	**in**	**bool**			
	return	**return**	**void**			

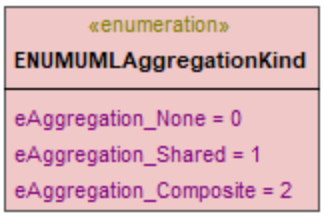
17.5.3.8 Enumerations

This is a list of all enumerations used by the UModel API on `UMLData` level. If your scripting environment does not support enumerations use the number-values instead.

17.5.3.8.1 UModelAPI - ENUMUMLAggregationKind

Enumeration **ENUMUMLAggregationKind**

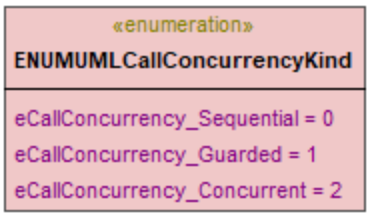
diagram	



typedElements	Interface **IUMLDataAll** Interface **IUMLProperty**	Operation **Aggregation** Operation **Aggregation**

17.5.3.8.2 UModelAPI - ENUMUMLCallConcurrencyKind

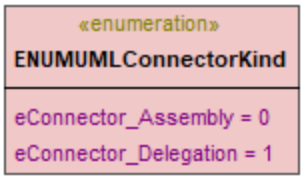
Enumeration **ENUMUMLCallConcurrencyKind**

diagram	

typedElements	Interface **IUMLBehavioralFeature** Interface **IUMLDataAll**	Operation **Concurrency** Operation **Concurrency**

17.5.3.8.3 UModelAPI - ENUMUMLConnectorKind

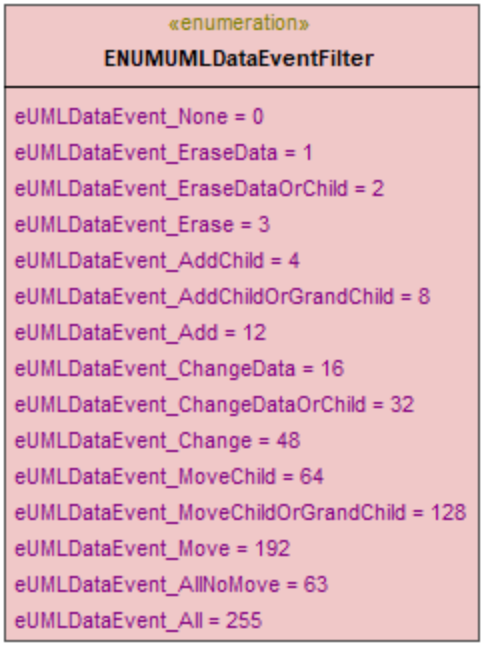
Enumeration **ENUMUMLConnectorKind**

diagram	

typedElements	Interface **IUMLConnector** Interface **IUMLDataAll**	Operation **ConnectorKind** Operation **ConnectorKind**

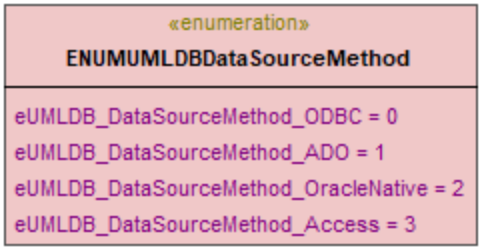
17.5.3.8.4 UModelAPI - ENUMUMLDataEventFilter

Enumeration **ENUMUMLDataEventFilter**

diagram	

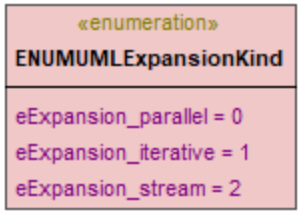
17.5.3.8.5 UModelAPI - ENUMUMLDBDataSourceMethod

Enumeration **ENUMUMLDBDataSourceMethod**

diagram		
typedElements	Interface **IImportDatabaseDlg**	Operation **SelectNewDataSourceByConnectionString**

17.5.3.8.6 UModelAPI - ENUMUMLExpansionKind

Enumeration **ENUMUMLExpansionKind**

diagram	
typedElements	Interface **IUMLDataAll** Operation **Mode** Interface **IUMLExpansionRegion** Operation **Mode**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.8.7 UModelAPI - ENUMUMLGuiStyleKind

Enumeration **ENUMUMLGuiStyleKind**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
typedElements	Interface **IUMLGuiStyle** Operation **Kind** Interface **IUMLGuiStyles** Operation **GetName GetStyle GetUsedValue GetValue SetValue**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.8.8 UModelAPI - ENUMUMLGuiTextLabelKind

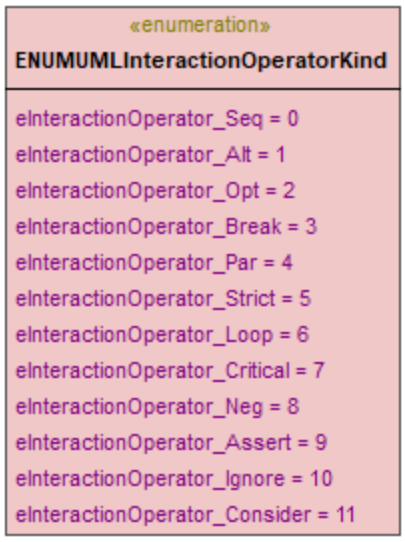
Enumeration **ENUMUMLGuiTextLabelKind**

diagram

typedElements		
	Interface **IUMLDataAll**	Operation **TextLabelKind**
	Interface **IUMLGuiTextLabel**	Operation **TextLabelKind**

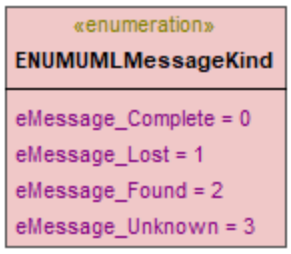
17.5.3.8.9 UModelAPI - ENUMUMLInteractionOperatorKind

Enumeration **ENUMUMLInteractionOperatorKind**

diagram	
typedElements	Interface **IUMLCombinedFragment** Operation **InteractionOperator** Interface **IUMLDataAll** Operation **InteractionOperator**

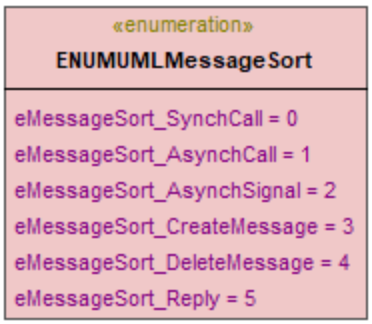
17.5.3.8.10 UModelAPI - ENUMUMLMessageKind

Enumeration **ENUMUMLMessageKind**

diagram	
typedElements	Interface **IUMLDataAll** Operation **MessageKind** Interface **IUMLMessage** Operation **MessageKind**

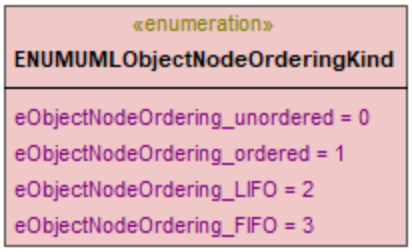
17.5.3.8.11 UModelAPI - ENUMUMLMessageSort

Enumeration **ENUMUMLMessageSort**

diagram		
typedElements	Interface IUMLDataAll Interface IUMLMessage	Operation MessageSort Operation MessageSort

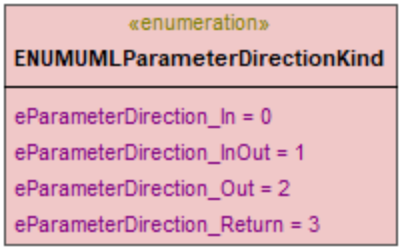
17.5.3.8.12 UModelAPI - ENUMUMLObjectNodeOrderingKind

Enumeration **ENUMUMLObjectNodeOrderingKind**

diagram		
typedElements	Interface IUMLDataAll Interface IUMLObjectNode	Operation Ordering Operation Ordering

17.5.3.8.13 UModelAPI - ENUMUMLParameterDirectionKind

Enumeration **ENUMUMLParameterDirectionKind**

diagram	
typedElements	Interface **IUMLDataAll** Operation **Direction** Interface **IUMLParameter** Operation **Direction**

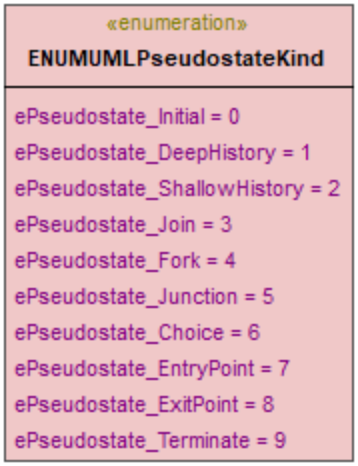
17.5.3.8.14 UModelAPI - ENUMUMLPredefinedElement

Enumeration **ENUMUMLPredefinedElement**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
typedElements	Interface **IUMLDataAll** Operation **ApplyPredefinedStereotype** **FindPredefinedOwnedElement** **GetStereotypeApplicationForPredefinedStereotype** **IsPredefinedStereotypeApplied** **SetPredefinedTaggedValueAt** **UnapplyPredefinedStereotype** Interface **IUMLElement** Operation **ApplyPredefinedStereotype** **FindPredefinedOwnedElement** **GetStereotypeApplicationForPredefinedStereotype** **IsPredefinedStereotypeApplied** **UnapplyPredefinedStereotype** Interface **IUMLStereotypeApplication** Operation **SetPredefinedTaggedValueAt**
documentation	**Deprecated:** ePredefined_Java_finalStereotypeOfProperty ePredefined_Java_finalStereotypeOfOperation ePredefined_Java_finalStereotypeOfClass

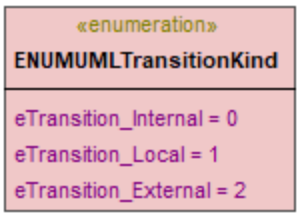
17.5.3.8.15 UModelAPI - ENUMUMLPseudostateKind

Enumeration **ENUMUMLPseudostateKind**

diagram	
typedElements	Interface **IUMLDataAll** Interface **IUMLPseudostate** Operation **PseudostateKind** Operation **PseudostateKind**

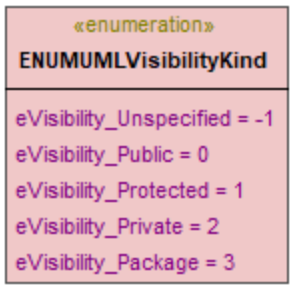
17.5.3.8.16 UModelAPI - ENUMUMLTransitionKind

Enumeration **ENUMUMLTransitionKind**

diagram	
typedElements	Interface **IUMLDataAll** Interface **IUMLTransition** Operation **TransitionKind** Operation **TransitionKind**

17.5.3.8.17 UModelAPI - ENUMUMLVisibilityKind

Enumeration **ENUMUMLVisibilityKind**

diagram	
typedElements	Interface **ILocalOptionsEditing** Interface **IUMLDataAll** Interface **IUMLElementImport** Interface **IUMLNamedElement** Interface **IUMLPackageImport** Operation **OperationsDefaultVisibility** **PropertiesDefaultVisibility** Operation **Visibility** Operation **Visibility** Operation **Visibility** Operation **Visibility**

18 SPL: el lenguaje de programación Spy

Esta sección ofrece una introducción al lenguaje de programación Spy (en adelante, *SPL*), que es el lenguaje de plantillas del generador de código.

En esta sección asumimos que el usuario tiene cierta experiencia en programación y ciertos conocimientos sobre operadores, variables, funciones y clases, así como sobre conceptos básicos de programación orientada a objetos, que se usa en gran medida en SPL.

Las plantillas utilizadas por UModel están en la carpeta **...\UModel**. Puede usar estos archivos para orientarse a la hora de crear sus propias plantillas.

¿Cómo funciona el generador de código?

Las entradas del generador de código son los archivos de plantilla (`.spl`) y el modelo de objetos que viene con XMLSpy. Los archivos de plantilla contienen instrucciones SPL (para crear archivos, leer información del modelo de objetos y realizar cálculos) intercaladas con fragmentos de código literal en el lenguaje de programación de destino.

El archivo de plantilla lo interpreta el generador de código y produce archivos de código fuente **.java** y **.cs** o cualquier otro tipo de proyecto, dependiendo de la plantilla.

18.1 Estructura básica de SPL

Un archivo SPL contiene texto literal, que se debe reproducir, intercalado con instrucciones del generador de código.

Las instrucciones del generador de código van entre corchetes: "**[**" y "**]**".
Entre cada par de corchetes se pueden introducir varias instrucciones. Las instrucciones se separan con una línea nueva o con dos puntos "**:**".

Estos dos ejemplos son igual de válidos:

Ejemplo nº1

```
[$x = 42
$x = $x + 1]
```

Ejemplo nº2

```
[$x = 42: $x = $x + 1]
```

Agregar texto a los archivos

El texto que no aparece entre corchetes **[ ]** se escribe directamente en el archivo de salida actual. Para reproducir corchetes literales, introduzca el carácter de escape \ de la siguiente manera: **\[** texto **\].** Para reproducir el carácter \, use ****.

Comentarios

Los comentarios introducidos en un bloque de instrucciones empiezan con el carácter **'** y terminan en la línea siguiente o con un carácter de cierre de bloque: **]**.

18.2 Variables

Los archivos SPL importantes exigen el uso de variables. Algunas variables vienen predefinidas por el generador de código y también se pueden crear variables nuevas con solo asignarles valores.

El carácter **$** se usa cuando se **declara** o **usa** una variable y un nombre de variable siempre tiene el prefijo **$**. Los nombres de variable distinguen **entre mayúsculas y minúsculas**.

Tipos de variables:

- entero, que también se usa como binario, siendo 0 equivalente a `false` y cualquier otro valor equivale a `true`
- cadena de texto
- objeto, que viene dado por UModel
- iterador, ver instrucción foreach

El tipo de variable se declara en la primera asignación de valor:

```
[$x = 0]
```

ahora `x` es un entero.

```
[$x = "teststring"]
```

ahora `x` es una cadena de texto.

Cadenas

Las constantes de cadena siempre aparecen entre comillas dobles, como en el ejemplo anterior. **\n** y **\t** dentro de comillas dobles equivalen a una línea nueva y a una tabulación, respectivamente. **\"** es una comilla doble literal y \\ es una barra diagonal inversa. Las constantes de cadena también pueden ocupar más de una línea.

Para la concatenación de cadenas se usa el carácter **&**:

```
[$BasePath = $outputpath & "/" & $JavaPackageDir]
```

Objetos

Los objetos representan la información que contiene el proyecto de UModel. Los objetos tienen **propiedades**, a las que puede acceder con el operador **.** . En SPL no puede crear objetos nuevos (vienen predefinidos por el generador de código y se derivan del código de entrada, pero se pueden asignar objetos a variables.

Ejemplo:

```
class [=$class.Name]
```

Este ejemplo reproduce la palabra "class" seguida de un espacio y el valor de la propiedad **Name** del objeto **$class**.

La tabla que aparece a continuación muestra la correspondencia entre elementos UML y propiedades SPL.

Variables predefinidas

Elemento UML	Propiedad SPL	Multi-plicidad	UML Attribute / Association	UModel Attribute / Association	Descripción
CaracterísticaDeComportamiento	isAbstract		isAbstract:Boolean		
CaracterísticaDeComportamiento	raisedException	*	raisedException:Type		
CaracterísticaDeComportamiento	ownedParameter	*	ownedParameter:Parameter		
ClasificadorConComportamiento	interfaceRealization	*	interfaceRealization:InterfaceRealization		
	isAbstract		isAbstract:Boolean		
	raisedException	*	raisedException:Type		
	ownedParameter	*	ownedParameter:Parameter		
	interfaceRealization	*	interfaceRealization:InterfaceRealization		
Clase	ownedOperation	*	ownedOperation:Operation		
Clase	nestedClassifier	*	nestedClassifier:Classifier		
Clasificador	namespace	*		namespace:Package	packages with code language <<namespace>> set
Clasificador	rootNamespace	*		project root namespace:String	VB only - root namespace
Clasificador	generalization	*	generalization:Generalization		
Clasificador	isAbstract		isAbstract:Boolean		
ParámetroDePlantillaDeClasificador	constrainingClassifier	*	constrainingClassifier		
Comentario	body		body:String		
Tipo de datos	ownedAttribute	*	ownedAttribute:Property		
Tipo de datos	ownedOperation	*	ownedOperation:Operation		
Elemento	kind			kind:String	

Elemento	owner	0..1	owner:Element		
Elemento	appliedStereotype	*		appliedStereotype:StereotypeApplication	applied stereotypes
Elemento	ownedComment	*	ownedComment:Comment		
ImportaciónDeElemento	importedElement	1	importedElement:PackageableElement		
Enumeración	ownedLiteral	*	ownedLiteral:EnumerationLiteral		
Enumeración	nestedClassifier	*		nestedClassifier::Classifier	
Enumeración	interfaceRealization	*		interfaceRealization:Interface	
LiteralDeEnumeración	ownedAttribute	*		ownedAttribute:Property	
LiteralDeEnumeración	ownedOperation	*		ownedOperation:Operation	
LiteralDeEnumeración	nestedClassifier	*		nestedClassifier:Classifier	
Característica	isStatic		isStatic:Boolean		
Generalización	general	1	general:Classifier		
Interfaz	ownedAttribute	*	ownedAttribute:Property		
Interfaz	ownedOperation	*	ownedOperation:Operation		
Interfaz	nestedClassifier	*	nestedClassifier:Classifier		
RealizaciónDeInterfaz	contract	1	contract:Interface		
ElementoMultiplicidad	lowerValue	0..1	lowerValue:ValueSpecification		
ElementoMultiplicidad	upperValue	0..1	upperValue:ValueSpecification		
ElementoConNombre	name		name:String		
ElementoConNombre	visibility		visibility:VisibilityKind		
ElementoConNombre	isPublic			isPublic:Boolean	visibility <public>
ElementoConNombre	isProtected			isProtected:Boolean	visibility <protected>
ElementoConNombre	isPrivate			isPrivate:Boolean	visibility <private>
ElementoConNombre	isPackage			isPackage:Boolean	visibility <package>
ElementoConNombre	namespacePrefix			namespacePrefix:String	XSD only - namespace prefix when exists
ElementoConNombre	parseableName			parseableName:String	CSharp, VB only - name with escaped keywords (@)

EspacioDeNombres	elementImport	*	elementImport:ElementImport		
Operación	ownedReturnParameter	0..1		ownedReturnParameter:Parameter	parameter with direction return set
Operación	type	0..1		type	type of parameter with direction return set
Operación	ownedOperationParameter	*		ownedOperationParameter:Parameter	all parameters excluding parameter with direction return set
Operación	implementedInterface	1		implementedInterface:Interface	CSharp only - the implemented interface
Operación	ownedOperationImplementations	*		implementedOperation:OperationImplementation	VB only - the implemented interfaces/operations
ImplementaciónDeOperación	implementedOperationOwner	1		implementedOperationOwner:Interface	interface implemented by the operation
ImplementaciónDeOperación	implementedOperationName			name:String	name of the implemented operation
ImplementaciónDeOperación	implementedOperationParseableName			parseableName:String	name of the implemented operation with escaped keywords
Paquete	namespace	*		namespace:Package	packages with code language <<namespace>> set
ElementoEmpaquetable	owningPackage	0..1		owningPackage	set if owner is a package
ElementoEmpaquetable	owningNamespacePackage	0..1		owningNamespacePackage:Package	owning package with code language <<namespace>> set
Parámetro	direction		direction:ParameterDirectionKind		
Parámetro	isIn			isIn:Boolean	direction <in>
Parámetro	isInOut			isInOut:Boolean	direction <inout>
Parámetro	isOut			isOut:Boolean	direction <out>
Parámetro	isReturn			isReturn:Boolean	direction <return>
Parámetro	isVarArgList			isVarArgList:Boolean	true if parameter is a variable argument list

Parámetro	defaultValue	0..1	defaultValue:ValueSpecification		
Propiedad	defaultValue	0..1	defaultValue:ValueSpecification		
ElementoRedefinible	isLeaf		isLeaf:Boolean		
Slot	name			name:String	name of the defining feature
Slot	values	*	value:ValueSpecification		
Slot	value			value:String	value of the first value specification
AplicaciónEstereotipo	name			name:String	name of applied stereotype
AplicaciónEstereotipo	taggedValue	*		taggedValue:Slot	first slot of the instance specification
CaracterísticaEstructural	isReadOnly		isReadOnly		
Clasificador estructurado	ow nedAttribute	*	ow nedAttribute:Property		
EnlaceDePlantilla	signature	1	signature:TemplateSignature		
EnlaceDePlantilla	parameterSubstitution	*	parameterSubstitution:TemplateParameterSubstitution		
ParámetroDePlantilla	paramDefault			paramDefault:String	template parameter default value
ParámetroDePlantilla	ow nedParameteredElement	1	ow nedParameteredElement:ParameterableElement		
SustituciónDeParámetroDePlantilla	parameterSubstitution			parameterSubstitution:String	Java only - code w ildcard handling
SustituciónDeParámetroDePlantilla	parameterDimensionCount			parameterDimensionCount:Integer	code dimension count of the actual parameter
SustituciónDeParámetroDePlantilla	actual	1	Ow nedActual:ParameterableElement		
SustituciónDeParámetroDePlantilla	formal	1	formal:TemplateParameter		
FirmaDePlantilla	template	1	template:TemplateableElement		
FirmaDePlantilla	ow nedParameter	*	ow nedParameter:TemplateParameter		
ElementoQueSePuedeConvertirEnPlantillas	isTemplate			isTemplate:Boolean	true if template signature set
ElementoQueSePuedeConvertirEnPlantillas	ow nedTemplateSignature	0..1	ow nedTemplateSignature:TemplateSignature		
ElementoQueSePuedeConvertirEnPlantillas	templateBinding	*	templateBinding:TemplateBinding		
Tipo	typeName	*		typeName:PackageableElement	qualified code type names

ElementoConTipo	type	0..1	type:Type		
ElementoConTipo	postTypeModifier			postTypeModifier :String	postfix code modifiers
EspecificaciónDeValor	value			value:String	string value of the value specification

Agregar un prefijo a los atributos de una clase durante la generación de código

Quizás sea necesario añadir el prefijo "m_" a todos los atributos nuevos del proyecto.

Todos los elementos de código nuevos se escriben usando las plantillas SPL. En el archivo `Attribute.spl` de la carpeta `UModelSPL\C#[Java]\Default` puede cambiar cómo se escriben los nombres con una simple operación de búsqueda y reemplazo.

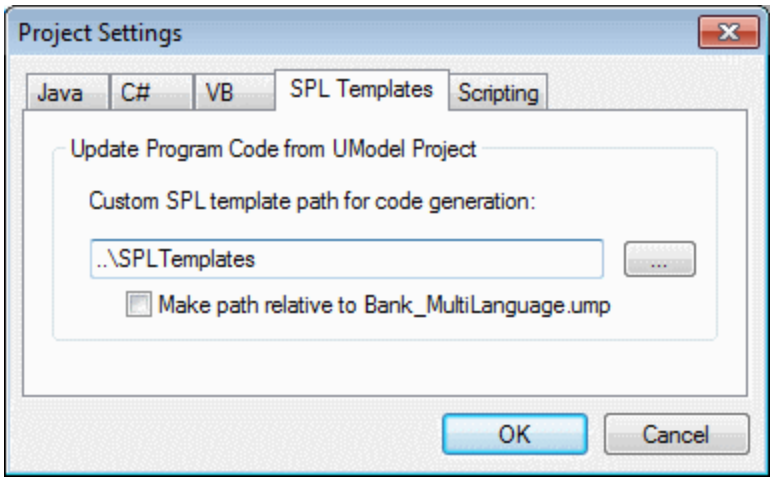
Por ejemplo, busque `$Property.name` y reemplácela con `"m_" & $Property.name`

Además, recomendamos actualizar inmediatamente el modelo con el código tras la generación de código. Esto garantiza que el código y el modelo estén sincronizados.

Nota: antes de modificarlas, recuerde copiar las plantillas SPL en un directorio de nivel superior (es decir, justo encima del directorio predeterminado **UModelSPL\C#**). Esto evita que las plantillas se sobrescriban cuando se instale una versión nueva de UModel. Compruebe que está marcada la casilla *Las definidas por el usuario reemplazan las predeterminadas* de la pestaña **Sincronizar el código con el modelo** del cuadro de diálogo "Configurar sincronización".

Plantillas SPL

Por cada proyecto de UModel se pueden especificar plantillas SPL distintas, haciendo clic en la opción de menú **Proyecto | Configuración del proyecto** (tal y como muestra el cuadro de diálogo). También puede usar rutas de acceso relativas. Las plantillas que no se encuentren en el directorio indicado se buscan en el directorio predeterminado local.



Objetos globales

`$Options` un objeto que almacena opciones globales:

`generateComments:bool` genera comentarios en la documentación (true/false)

`$Indent`	una cadena utilizada para aplicar sangría al código generado y que representa el nivel de anidamiento actual
`$IndentStep`	una cadena utilizada para aplicar sangría al código generado y que representa un nivel de anidamiento
`$NamespacePrefix`	solo en XSD, el prefijo de espacio de nombres de destino, si existe

Rutinas de manipulación de cadenas

```
integer Compare(s)
```

El valor devuelto indica la relación lexicográfica entre la cadena y `s` (distingue entre mayús/minús):

<0:	la cadena es menor que `s`
0:	la cadena es idéntica a `s`
>0:	la cadena es mayor que `s`

```
integer CompareNoCase(s)
```

El valor devuelto indica la relación lexicográfica entre la cadena y `s` (no distingue entre mayús/minús):

<0:	la cadena es menor que `s`
0:	la cadena es idéntica a `s`
>0:	la cadena es mayor que `s`

```
integer Find(s)
```

Busca la primera aparición de la subcadena `s`.
Devuelve el índice basado en cero del primer carácter de `s` o -1 si `s` no se encuentra.

```
string Left(n)
```

Devuelve los primeros `n` caracteres de la cadena.

```
integer Length()
```

Devuelve la longitud de la cadena.

```
string MakeUpper()
```

Devuelve la cadena en mayúsculas.

```
string MakeUpper(n)
```

Devuelve la cadena con los primeros `n` caracteres en mayúsculas.

```
string MakeLower()
```

Devuelve la cadena en minúsculas.

```
string MakeLower(n)
```

Devuelve la cadena con los primeros `n` caracteres en minúsculas.

```
string Mid(n)
```

Devuelve la cadena empezando por la posición de índice basado en cero `n`

```
string Mid(n,m)
```

Devuelve la cadena empezando por la posición de índice basado en cero `n` y la longitud `m`

```
string RemoveLeft(s)
```

Devuelve la cadena sin la subcadena `s` si Left(s.Length()) es igual a la subcadena `s`.

```
string RemoveLeftNoCase(s)
```

Devuelve la cadena sin la subcadena `s` si Left(s.Length()) es igual a la subcadena `s` (no distingue entre mayús/minús).

```
string RemoveRight(s)
```

Devuelve la cadena sin la subcadena `s` si Right(s.Length()) es igual a la subcadena `s`.

```
string RemoveRightNoCase(s)
```

Devuelve la cadena sin la subcadena `s` si Right(s.Length()) es igual a la subcadena `s` (no distingue entre mayús/minús).

```
string Repeat(s,n)
```

Devuelve la cadena que contiene la subcadena `s` repetida `n` veces.

```
string Right(n)
```

Devuelve los últimos `n` caracteres de la cadena.

18.3 Operadores

Los operadores de SPL funcionan como en casi todos los lenguajes de programación.

A continuación ofrecemos la lista de operadores de SPL ordenados por orden de prioridad (de mayor a menor):

.	Acceso a la propiedad del objeto
( )	Agrupación de expresiones
true	Constante binaria "true"
false	Constante binaria "false"
&	Concatenación de cadenas de texto
-	Signo para un número negativo
not	Negación lógica
*	Multiplicación
/	División
%	Resto
+	Suma
-	Resta
<=	Menor o igual que
<	Menor que
>=	Mayor o igual que
>	Mayor que
=	Igual
<>	No igual
and	Conjunción lógica (con evaluación en cortocircuito)
or	Disyunción lógica (con evaluación en cortocircuito)
=	Asignación

18.4 Condiciones

En SPL puede usar instrucciones `if` estándar. Esta es la sintaxis para estas instrucciones:

```
if condición
	instrucciones
else
	instrucciones
endif
```

o sin la parte **else**:

```
if condición
	instrucciones
endif
```

Nota: observe que la condición no aparece entre paréntesis.
Al igual que en otros lenguajes de programación, las condiciones se construyen con operadores lógicos y de comparación.

Ejemplo:

```
[if $namespace.ContainsPublicClasses and $namespace.Prefix <> ""]
	lo que usted quiera ['inserta lo que usted quiera en el archivo de resultados]
[endif]
```

Instrucción *switch*

SPL también incluye una instrucción de control para múltiples selecciones.

Sintaxis:

```
switch $variable
	case X:
		instrucciones
	case Y:
	case Z:
		instrucciones
	default:
		instrucciones
endswitch
```

Las etiquetas "case" son constantes o variables.

Al igual que C, SPL no admite el paso implícito de una etiqueta case a otra y, por tanto, no es necesario usar la instrucción `break`.

18.5 Colecciones y foreach

Colecciones e iteradores

Una colección contiene varios objetos, como una matriz normal y corriente. Los iteradores sirven para almacenar e incrementar índices de matriz al acceder a objetos.

Sintaxis:

```
foreach iterador in colección
       instrucciones
next
```

Ejemplo nº1:

```
[foreach $class in $classes
       if not $class.IsInternal
              ]      class [=$class.Name];
[      endif
next]
```

Ejemplo 2:

```
[foreach $i in 1 To 3
      Write "// Step " & $i & "\n"
      ' Do some work
next]
```

La instrucción **foreach** recorre todos los elementos de **$classes** y en cada uno de ellos ejecuta el código que sigue a la instrucción hasta llegar a la instrucción **next**.

En cada iteración se asigna **$class** al siguiente objeto de clase. Simplemente se trabaja con el objeto de clase en lugar de usar `classes[i]->Name()` como en C++.

Los iteradores de colección tienen estas propiedades adicionales:

`Index`	El índice actual, empezando por 0
`IsFirst`	true si el objeto actual es el primer objeto de la colección (el índice es 0)
`IsLast`	true si el objeto actual es el último objeto de la colección

Ejemplo:

```
[foreach $enum in $facet.Enumeration
       if not $enum.IsFirst
              ], [
       endif
       ]"[=$enum.Value]"[
next]
```

Rutinas para la manipulación de colecciones:

collection **SortByName**(bAscending)

devuelve una colección cuyos elementos están ordenados por nombre (con distinción de mayúsculas y minúsculas) en orden ascendente o descendente.

collection **SortByNameNoCase**(bAscending)
devuelve una colección cuyos elementos están ordenados por nombre (sin distinción de mayúsculas y minúsculas) en orden ascendente o descendente.

Ejemplo:
```
$SortedNestedClassifier = $Class.nestedClassifier.SortByNameNoCase( true )
```

collection **SortByKind**(bAscending)
devuelve una colección cuyos elementos están ordenados por tipo (p. ej. clase, interfaz...) en orden ascendente o descendente.

collection **SortByKindAndName**(bAscendingKind, bAscendingName)
devuelve una colección cuyos elementos están ordenados por tipo (p. ej. clase, interfaz...) en orden ascendente o descendente y si el tipo es idéntico al nombre (con distinción de mayúsculas y minúsculas en orden ascendente o descendente).

collection **SortByKindAndNameNoCase**(bAscending)
devuelve una colección cuyos elementos están ordenados por tipo (p. ej. clase, interfaz...) en orden ascendente o descendente y si el tipo es idéntico al nombre (sin distinción de mayúsculas y minúsculas en orden ascendente o descendente).

18.6 Subrutinas

El generador de código admite subrutinas en forma de procedimientos o funciones.

Características:
- Se pasan valores por valor y por referencia
- Parámetros locales/globales (locales dentro de las subrutinas)
- Variables locales
- Invocación
- Invocación recurrente (las subrutinas se pueden llamar a sí mismas)

18.6.1 Declaración de subrutinas

Subrutinas

Ejemplo de sintaxis:

```
Sub SimpleSub()
... lines of code
EndSub
```

- **Sub** es la palabra clave que denota el procedimiento.
- **SimpleSub** es el nombre asignado a la subrutina.
- Los **paréntesis** pueden contener una lista de parámetros.
- El bloque de código de una subrutina empieza inmediatamente después del paréntesis de cierre.
- **EndSub** denota el final del bloque de código.

Nota: no se permiten **declaraciones** de subrutinas recursivas o en cascada (es decir, una subrutina no puede contener otra subrutina).

Parámetros

Los parámetros también se pueden pasar con procedimientos usando esta sintaxis:

- Todos los parámetros deben ser variables.
- Las variables deben tener el prefijo **$**.
- Las variables locales se definen en una subrutina.
- Las variables globales se declaran de forma explícita, fuera de las subrutinas.
- Si hay varios parámetros, se separan con comas dentro de los paréntesis.
- Los parámetros pueden pasar valores.

Parámetros: pasar valores

Los parámetros se pueden pasar de dos maneras: por valor y por referencia, usando las palabras clave **ByVal** y **ByRef** respectivamente.

Sintaxis:

```
' definir sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
] ...
```

- **ByVal** especifica que el parámetro se pasa por valor. Recuerde que la mayoría del os objetos solamente se pueden pasar por referencia.
- **ByRef** especifica que el parámetro se pasa por referencia. Se trata del método predeterminado si no se especifica ByVal ni ByRef.

Valores devueltos de funciones

Para devolver un valor desde una subrutina use la instrucción **return**. Este tipo de función se puede llamar desde dentro de una expresión.

Ejemplo:

```
' definir una función
[Sub MakeQualifiedName( ByVal $namespacePrefix, ByVal $localName )
if $namespacePrefix = ""
  return $localName
else
  return $namespacePrefix & ":" & $localName
endif
EndSub
]
```

18.6.2 Invocación de subrutinas

Use **call** para invocar una subrutina, seguido del nombre y los parámetros del procedimiento (si procede).

```
Call SimpleSub()
```

o

```
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
```

Invocación de funciones

Para invocar una función (cualquier subrutina que contenga una instrucción **return**) use su nombre dentro de una expresión. No use la instrucción **call** para llamar a funciones.

Ejemplo:

```
$QName = MakeQualifiedName($namespace, "entry")
```

19 Información sobre licencias

En esta sección encontrará información sobre:

- la distribución de este producto de software
- la activación del software y medición de licencias
- el contrato de licencia para el usuario final que rige el uso de este producto de software

Los términos del contrato de licencia que aceptó al instalar el producto de software son vinculantes, por lo que rogamos lea atentamente toda esta información.

Para leer los términos y condiciones de cualquiera de las licencias de Altova, consulte la página de información legal de Altova en el sitio web de Altova.

19.1 Distribución electrónica de software

Este producto está disponible por distribución electrónica de software, un método de distribución que ofrece ventajas únicas:

- Puede evaluar el software de forma totalmente gratuita durante 30 días antes de decidir si compra el producto (*Nota: la licencia para Altova Mobile Together Designer es gratuita*).
- Si decide comprarlo, puede hacer un pedido en línea en el sitio web de Altova y conseguir en pocos minutos el software con licencia.
- Si realiza el pedido en línea, siempre recibirá la versión más reciente de nuestro software.
- El paquete de instalación del producto incluye un sistema de ayuda en pantalla al que se puede acceder desde la interfaz de la aplicación. La versión más reciente del manual del usuario está disponible en www.altova.com (i) en formato HTML y (ii) en formato PDF para descargar e imprimir si lo desea.

Período de evaluación de 30 días

Después de descargar el producto de software, puede probarlo de forma totalmente gratuita durante un plazo de 30 días. Pasados unos 20 días, el software empieza a recordarle que no tiene una licencia. El mensaje de aviso aparece una sola vez cada vez que se inicie la aplicación. Para seguir utilizando el programa una vez pasado el plazo de 30 días, deberá comprar una licencia permanente, que se entrega en forma de código clave. Para desbloquear el producto debe introducir ese código clave en el cuadro de diálogo "Activación del software".

Las licencias de los productos pueden comprarse directamente en la tienda en línea del sitio web de Altova.

Distribuir la versión de evaluación a otros usuarios de su organización

Si desea distribuir la versión de evaluación en la red de su compañía o si desea usarlo en un PC que no está conectado a Internet, solamente puede distribuir los programas de instalación (siempre y cuando no se modifiquen de forma alguna). Todo usuario que acceda al instalador debe solicitar su propio código clave de evaluación (de 30 días). Una vez pasado este plazo de 30 días, todos los usuarios deben comprar también una licencia para poder seguir usando el producto.

19.2 Activación del software y medición de licencias

Durante el proceso de activación del software de Altova, puede que la aplicación utilice su red interna y su conexión a Internet para transmitir datos relacionados con la licencia durante la instalación, registro, uso o actualización del software a un servidor de licencias operado por Altova y para validar la autenticidad de los datos relacionados con la licencia y proteger a Altova de un uso ilegítimo del software y mejorar el servicio a los clientes. La activación es posible gracias al intercambio de datos de la licencia (como el sistema operativo, la dirección IP, la fecha y hora, la versión del software, el nombre del equipo, etc.) entre su equipo y el servidor de licencias de Altova.

Su producto incluye un módulo integrado de medición de licencias que le ayudará a evitar infracciones del contrato de licencia para el usuario final. Puede comprar una licencia de un solo usuario o de varios usuarios para el producto de software y el módulo de medición de licencias se asegura de que no se utiliza un número de licencias mayor al permitido.

Esta tecnología de medición de licencias usa su red de área local (LAN) para comunicarse con las instancias de la aplicación que se ejecutan en equipos diferentes.

Licencia de un solo usuario

Cuando se inicia la aplicación, se inicia el proceso de medición de licencias y el software envía un breve datagrama de multidifusión para averiguar si hay otras instancias del producto activas en otros equipos del mismo segmento de red al mismo tiempo. Si no recibe ninguna respuesta, la aplicación abre un puerto para escuchar a otras instancias de la aplicación.

Licencia de varios usuarios

Si se usa más de una instancia de la aplicación dentro de la misma red LAN, estas instancias se comunicarán entre ellas al iniciarse. Estas instancias intercambian códigos claves para que ayudarle a no sobrepasar por error el número máximo de licencias concurrentes. Se trata de la misma tecnología de medición de licencias que suele utilizarse en Unix y en otras herramientas de desarrollo de bases de datos. Gracias a ella puede comprar licencias de varios usuarios de uso concurrente a un precio razonable.

Las aplicaciones se diseñaron de tal modo que envían pocos paquetes pequeños de red y no cargan demasiado su red. Los puertos TCP/IP (2799) utilizados por su producto de Altova están registrados oficialmente en la IANA (*para más información consulte el sitio web de la IANA www.iana.org*) y nuestro módulo de medición de licencias es una tecnología probada y eficaz.

Si usa un servidor de seguridad, puede notar las comunicaciones del puerto 2799 entre los equipos que ejecutan los productos de Altova. Si quiere, puede bloquear ese tráfico, siempre y cuando esto no resulte en una infracción del contrato de licencia.

Nota sobre los certificados

Su aplicación de Altova contacta con el servidor de licencias de Altova (`link.altova.com`) vía HTTPS. Para esta comunicación, Altova usa un certificado SSL registrado. Si se reemplaza este certificado (por ejemplo, si

lo reemplaza su departamento de informática o un organismo externo), entonces su aplicación de Altova le advertirá de que la conexión puede no ser segura. Si usa el certificado sustitutivo para iniciar la aplicación, lo hace por su cuenta y riesgo. Si ve un mensaje de advertencia de que la conexión puede no ser segura, compruebe el origen del certificado y consulte con su equipo técnico (que decidirán si se debe continuar con el reemplazo del certificado de Altova).

Si su organización necesita usar su propio certificado (por ejemplo, para monitorizar la comunicación hacia y desde equipos cliente), entonces recomendamos que instale en su red Altova LicenseServer, el software gratuito de gestión de licencias de Altova. Así, sus equipos cliente pueden seguir usando los certificados de su organización y AltovaLicenseServer puede usar el certificado de Altova cuando necesite comunicarse con Altova.

19.3 Contrato de licencia para el usuario final

- Encontrará el Contrato de licencia de Altova para el usuario final en: https://www.altova.com/legal/eula
- Encontrará la Política de privacidad de Altova en: https://www.altova.com/privacy

Índice

.

.NET Framework,
incluir archivo, 168

A

Abrir,
URL, 735
Abrir proyecto,
control de código fuente, 686
Absolutos,
vínculos absolutos y relativos, 327
Abstracta,
clase, 32
Acceso directo,
asignar/eliminar, 758
mostrar en información rápida, 762
tecla, 758
teclado, 758
Acelerar,
rendimiento, 172
Activar/desactivar,
modo compacto, 475
Actividad, 358
agregar a estado, 358
agregar operación, 358
BPMN, 493
iconos, 714
Actividades,
agregar diagrama de actividades a transición, 358
diagrama de, 339
diagramas SysML, 533
Actor,
definido por el usuario, 21
personalizar, 21
Actualizar,
archivo de proyectos, 231
Actualizar estado,
control de código fuente, 706
ADO,
como interfaz de conexión de datos, 560
configurar una conexión, 566
ADO.NET,
configurar una conexión, 571
Advertencias,
durante ingeniería de código, 95
Agregar, 21, 697
al control de código fuente, 697
diagrama a un paquete, 21
paquete a un proyecto, 21
proyecto al control de código fuente, 697
proyecto nuevo, 157
Ajuste,
líneas de ajuste mientras se arrastran objetos, 762
Alinear,
elementos mientras se arrastran, 21
Anotación,
esquema XML, 475
texto (BPMN), 506
Anotación de texto,
BPMN, 506
API COM,
en el editor de scripting, 798
Aplicación,
externa (argumentos), 755
Application object, 835
Archivo, 735
abrir desde URL, 735
combinar archivos de proyecto, 290
ejemplo del tutorial, 17
nuevo / cargar / guardar con archivo de procesamiento por lotes, 106
ump, 157
Archivos binarios,
importar C# y Java, 217
Archivos de proyecto,
Borland - MS Visual Studio .Net, 739
Archivos locales,
vínculos absolutos o relativos, 327
Argumentos,
herramientas externas, 755
Artefacto,
agregar al nodo, 59
BPMN, 506
manifestación, 59
Asignar,
acceso directo a un comando, 758

Asociación, 32
agregado/compuesto, 32
asociaciones reflexivas, 143
BPMN, 503
cambiar las propiedades de, 143
caso de uso, 21
como relación, 139
crear, 139, 143
entre clases, 32
ver, 143
ver propiedad con tipo, 299
vínculos entre objetos, 46
Asociación de colecciones,
crear, 146
requisitos previos, 146
resolver a plantillas de colecciones, 146
Asociaciones,
ver, 90
Atributo,
mostrar / ocultar, 433
ventana de finalización automática, 762
Attribute,
coloring, 438
Automática,
respuesta automática a mensajes, 404
Automáticamente,
agregar operación, 358
Ayuda, 775

B

Ball and socket,
notación de forma esférica, 433
Barra de herramientas, 753
activar/desactivar, 753
agregar comando, 752
crear nueva, 753
mostrar iconos de tamaño grande, 762
restaurar comandos, 753
Barras de herramientas,
restaurar, 745
Base,
clase, 41
Base de datos,
actualizar desde el modelo, 554
configurar para ingeniería de ida y vuelta, 553
importar en UModel, 538, 541
modelar con UModel, 540
Bases de datos,
compatibles, 16
BD, 540, 541, 553, 554
Binarios,
compatibilidad con binarios confusos, 217
Borland,
archivo de proyecto BSDJ, 739
BPMN, 492
artefactos, 506
asociación, 503
convertir 1.0 en 2.0, 492
BPMN 2.0,
eventos, 493
objetos de flujo, 493
tareas, 493
BSDJ,
proyecto Borland, 739
Buscar, 737
diagramas, 114
elementos, 114
elementos de modelado, 737
pestañas de búsqueda, 80
texto, 114
y reemplazar, 737
Business Process Modeling Notation, 492
iconos, 731
Búsqueda,
diagramas, 114
elementos, 114
texto, 114

C

C#,
error handling, 851
generar código, 173, 180
importar archivo binario, 217
importar código fuente, 202
opciones de generación de código, 179
opciones de importación de código, 205
propiedades autoimplementadas, 180
C++,
añadir a un diagrama, 195
error handling, 851

C++,
generar código, 195
importar código fuente, 204
ingeniería inversa, 204
Calificador de asociación,
crear, 143
CallBehavior,
insertar, 340
CallOperation,
insertar, 340
Cambiar de nombre,
clasificador, 232
Cambiar de proveedor,
control de código fuente, 707
Cambiar nombres de clases,
efecto en el nombre del archivo de código, 234
Cambio de estado,
definir en una línea de tiempo, 425
Capas,
añadir a diagramas, 134
bloquear, 134
eliminar, 134
mostrar, 134
ocultar, 134
Carpeta,
carpeta de ejemplos, 17
Carpetas,
obtener carpetas en control de código fuente, 692
Casilla de activación,
especificación de ejecución, 398
Caso de uso, 21
agregar, 21
asociación, 21
compartimento, 21
multilínea, 21
Casos de uso,
diagrama de, 385
diagramas SysML, 536
iconos, 729
Catálogo,
archivo de catálogo, 762
Ciclo de vida,
diagrama de, 424
iconos, 728
Clase, 433
en diagramas de componentes, 53
expandir y contraer compartimentos, 433
iconos, 716
interfaz con ball and socket (notación de forma esférica), 433
operación de clase (invalidar), 433
varias instancias en el diagrama, 433
Clase base,
invalidar, 433
Clases,
abstractas y concretas, 32
agregar, 32
agregar operaciones, 32
agregar propiedades, 32
asociaciones, 32
base, 41
cambios de nombre (sincronización), 234
derivadas, 41
diagrama de, 433
diagramas, 32
habilitar ventana de finalización automática, 762
sincronización, 231
Clasificador,
cambiar de nombre, 232
nuevo, 232
restringir, 296
Class,
syntax coloring, 438
Code,
generate sequence diagram from, 841
generate sequence diagram manually, 842
Código, 234
agregar código a diagrama de secuencia, 421
código Java y nombres de archivos de clases, 234
generar código a partir de diagramas de secuencia, 417
generar diagrama de secuencia a partir de código, 410
generar varios diagramas de secuencia a partir de código, 416
predeterminado, 762
refactorizar, 234
sincronización, 231
SPL, 1348
Código de refactorización,
nombres de clases (sincronizar), 234
Colaboración,
diagrama de estructura de un compuesto, 452
Color,
syntax coloring - enable/disable, 438
Comando,
agregar a barra de herramientas / menú, 752
eliminar de menú, 760
menú contextual, 760
procesamiento de la línea de comandos, 101

Comando,
restaurar menú, 760
Comandos,
línea de comandos: nuevo / cargar / guardar, 106
procesamiento de la línea de comandos, 106
CombinaciónDePaquete, 457
ver, 90
Combinar,
código con el modelo, 739
modelo con el código, 739
omitir directorio, 762
proyectos, 290
Compacto,
modo (activar/desactivar), 475
Comparar archivos de código fuente, 704
Compartimento,
contenedor, 505
expandir uno / varios, 433
Compartir,
desde el control de código fuente, 701
paquete y diagrama, 169
Compatibilidad,
actualizar proyectos, 231
Complemento,
configuración de la interfaz del usuario, 826
registrar, 823
registro, 824
XMLSPY, 815
Complemento de UModel para Visual Studio,
instalar, 647
Componentes, 53
diagrama, 53
diagrama de, 454
iconos, 719
insertar clase, 53
realización, 53
Comportamiento,
diagramas de, 339
Composición,
crear asociación, 32
Compuertas,
BPMN 2.0, 493
Compuerta compleja (bifurcación/convergencia), 493
Compuerta exclusiva basada en datos (XOR), 493
Compuerta exclusiva basada en eventos (XOR), 493
Compuerta inclusiva (OR), 493
Compuerta paralela (AND), 493
Comunicación,
diagrama de, 385
iconos, 717
Concreta,
clase, 32
Conexión de base de datos,
configurar, 560
ejemplo de configuración, 588
iniciar el asistente, 561
Configuración,
control de código fuente, 762
Configurar,
interfaz gráfica XMLSPY, 826
sincronización, 231
Confusos,
compatibilidad con binarios confusos, 217
Contención,
dibujar en un diagrama, 149
Contenedor,
compartimento, 505
Contraer,
compartimentos de clase, 433
Contraído,
subproceso, 501
Contrato de licencia para el usuario final, 1364, 1368
Control de código fuente, 682
abrir proyecto, 686
actualizar estado, 706
agregar al control de código fuente, 697
anular desprotección, 696
cambiar de proveedor, 707
comandos, 686
desproteger, 693
ejecutar interfaz nativa, 706
habilitar / deshabilitar, 689
mostrar diferencias, 704
mostrar historial, 702
obtener archivo, 691
obtener la versión más reciente, 690
opciones / configuración, 762
propiedades, 706
proteger, 695
quitar del, 700
Control de versiones,
comandos, 686
Controladores de BD,
resumen, 563
Controladores ODBC,
comprobar disponibilidad, 578

Convergencia,
crear en diagrama de actividades, 342
Convertir,
BPMN 1.0 en 2.0, 492
C# en Java (de un modelo a otro), 312
Correo electrónico,
enviar proyecto, 735
CPU,
carga - acelerar actualización de estado en segundo plano, 682
CR/LF,
en el archivo ump al guardarlo, 157
Crear,
métodos getter / setter, 433
CSPROJ - CSDPROJ,
MS Visual Studio .Net, 739
Cuadrícula,
líneas de ajuste, 762
líneas de ajuste mientras se arrastran objetos, 21
CVS, 682

D

Definidas por el usuario,
plantillas SPL, 231
Definido por el usuario,
actor, 21
Dependencia,
incluir, 21
uso, 53
Dependencias,
ver, 90
Derivada,
clase, 41
Descargar proyecto de control de código fuente, 686
Deshabilitar el control de código fuente, 689
Deshacer desprotección, 696
Desproteger, 693
Diagram - sequence,
generate from code, 841
generate manually from code, 842
generate sequence diagram from code, 842
Diagrama, 433, 762
agregar a Favoritos, 87
agregar actividad a transición, 358
agregar código a diagrama de secuencia, 421
BPMN, 492
compartir paquete y diagrama, 169
de actividades, 339
de base de datos (importación), 538
de bloque interno SysML, 524
de casos de uso, 385
de casos de uso SysML, 536
de ciclo de vida, 424
de clases, 433
de componentes, 454
de comunicación, 385
de esquema XML, 474
de esquema XML (importación), 475
de estructura de un compuesto, 451
de implementación, 454
de máquina de estados, 356
de máquina de estados SysML, 535
de objetos, 455
de paquetes, 456
de secuencia, 394
de secuencia SysML, 534
desplazamiento rápido, 92
encontrar elementos no usados, 116
estilos, 89
generar código a partir de diagramas de secuencia, 417
generar diagrama de dependencias entre paquetes, 456
global de interacción, 389
guardar como PNG, 735
guardar elementos como mapa de bits, 737
guardar los diagramas abiertos con el proyecto, 762
iconos, 713
insertar diagrama SysML, 520
insertar elementos en, 110
omitir elementos de archivos incluidos, 762
relación de iconos, 82
varias instancias de la clase, 433
ver un resumen de, 92
Diagrama de actividades,
crear rama / convergencia, 342
elementos, 345
insertar elementos, 340
Diagrama de bloque interno,
SysML, 524
Diagrama de ciclo de vida,
cambiar de un tipo a otro, 425
ciclo de vida, 425
evento/estímulo, 429
insertar elementos, 425

Diagrama de ciclo de vida,
línea de vida, 425
marca de graduación, 428
mensaje, 431
restricciónDeDuración, 430
restricciónDeTiempo, 431
valor general de la línea de vida, 425
Diagrama de comunicación,
generar a partir de un diagrama de secuencia, 386
Diagrama de definición de bloques,
SysML, 521
Diagrama de paquetes,
insertar elementos, 457
SysML, 530
Diagrama de requisitos,
SysML, 532
Diagrama de secuencia, 410
agregar código a, 421
fragmento combinado, 399
generar a partir de código, 410
generar a partir de un diagrama de comunicación, 386
generar código a partir de, 417
generar diagrama de secuencia a partir de métodos getter/setter, 416
generar varios diagramas de secuencia a partir de código, 416
insertar elementos, 396
invariante de estado, 404
línea de vida, 398
mensajes, 404
nombres de operación que se deben omitir, 410
puerta, 403
uso de interacción, 403
Diagrama global de interacción,
insertar elementos, 390
Diagramas, 338
abrir, 129
acercar/alejar, 138
ajustar a la ventana, 138
añadir capas a, 134
borrar del proyecto, 130
cambiar el tamaño de, 131
cambiar la apariencia de, 131
crear, 97, 125
de comportamiento, 339
de estructura, 433
generar, 126
generar desde Panel Jerarquía, 90
ver dentro del proyecto, 86
Directorio,
cambiar ubicación del proyecto, 157
carpeta de ejemplos, 17
omitir durante la combinación, 762
Directorio de trabajo,
control de código fuente, 686
Diseño, 742
Disparador,
definir disparador de transición, 358
Distribución,
de productos de software de Altova, 1364, 1365
División,
evitar división entre las páginas, 735
Documentación, 327
añadir a elementos, 122
generar código fuente con, 122
generar proyecto UML, 327
oimportar desde código fuente, 122
vínculos relativos, 327
DSN de archivo,
configurar, 578
DSN de sistema,
configurar, 578
DSN de usuario,
configurar, 578

E

Edición, 737
Editor de scripting,
vista general, 790
Ejecutar interfaz nativa, 706
Ejemplos,
carpeta del tutorial, 17
Elemento,
agregar a Favoritos, 87
cambiar propiedades de, 88
estilos, 89
generar documentación, 327
guardar elemento seleccionado como mapa de bits, 737
Elementos,
agregar a diagrama, 110
agregar al modelo, 109
alinear con un diagra,a, 132
añadir a diagrama, 110
añadir al modelo, 82, 109

Elementos,
aplicar imágenes personalizadas a, 123
borrar del diagrama, 113
borrar del proyecto, 113
buscar, 114
cambiar el tamaño, 132
cambiar la apariencia de, 123
copiar, 112
diseño automático, 132
documentar, 93, 122
encontrar en un diagrama, 116
hipervínculos, 119
insertar en diagrama de máquina de estados, 357
mover, 112
moverse entre capas, 134
omitir elementos de archivos incluidos, 762
reemplazar, 114
renombrar, 112
restringir, 117
Elimina,
acceso directo, 758
Eliminar, 752
barra de herramientas, 753
comando de menú contextual, 760
comando de una barra de herramientas, 752
icono de una barra de herramientas, 752
Enlace,
plantilla, 298
Entorno de scripting, 788
Enviar por correo electrónico,
proyecto, 735
Error handling,
general description, 851
Errores,
durante ingeniería de código, 95
Especializar,
generalizar, 41
Especificación de ejecución,
línea de vida, 398
Esquema,
XML, 474
XML (importación), 475
Esquema XML,
anotación, 475
diagrama de, 474
iconos, 730
Estado, 358
agregar actividad, 358
definir transición entre, 358
insertar estado simple, 358
ortogonal, 365
submáquina, 365
Estado compuesto, 365
agregar región, 365
Estado de submáquina,
agregar punto de entrada/salida, 365
Estereotipos,
agregar estilos personalizados a, 471
agregar iconos personalizados a, 471
añadir al panel Propiedades, 88
aplicar a elementos, 152, 466
crear, 463, 466
definición, 150
ejemplo, 466
ejemplos, 150, 461
Estilos,
aplicar a diagramas, 131
aplicar a elementos, 123
aplicar a líneas, 141
elemento precedente, 123, 131
en cascada, 123, 131, 141
precedencia, 141
Estructura,
diagramas de, 433
Estructura de un compuesto,
diagrama de, 451
iconos, 718
insertar elementos, 452
Etiquetas,
identificadores ID y UUID, 643
Evento,
BPMN, 493
Evento/estímulo,
diagrama de ciclo de vida, 429
Excepción,
agregar excepción generada, 433
Excepciones generadas,
agregar, 433
Expandido,
subproceso, 500
Expandir,
todos los compartimentos de clase, 433
Exportar,
como XMI, 643
Extensión,
XMI, 643

F

Favoritos,
agregar a, 87
eliminar de, 87
panel, 87
Finalización automática,
función, 32
ventana para edición de clases, 762
Finalizaciópn automática de tipos de datos,
desencadenar, 136
deshabilitar, 136
Firebird,
conectarse por JDBC, 588
conectarse por ODBC, 590
Firma,
plantilla, 296, 297
Flujo,
condicional, 503
mensaje, 503
predeterminado, 503
secuencia, 503
Flujo condicional, 503
Flujo continuo,
modelar / transmitir por secuencias de actividades SysML, 533
Flujo de secuencia, 503
Flujo de trabajo,
proyecto, 157
Flujo del elemento,
crear, 524
Flujo del mensaje, 503
Flujo predeterminado, 503
Flujos,
crear flujo del elemento, 524
Fragmento combinado, 399
Fusión,
a 3 bandas, 291
a 3 bandas manual, 292

G

Generalización,
como relación, 110, 139
crear, 139
Generalizaciones,
ver, 90
Generalizar,
especializar, 41
Generar,
diagrama de secuencia a partir de código, 410
diagrama de secuencia a partir de diagrama de comunicación, 386
documentación del proyecto UML, 327
RealizacionesDeComponente automáticamente, 232
respuesta a mensajes automáticamente, 404
varios diagramas de secuencia a partir de código, 416
Generate,
sequence diagram from code, 841
Generate manually,
sequence diagram from code, 842
Get,
métodos getter / setter, 433
Getter / Setter,
generar diagrama de secuencia a partir de métodos getter/setter, 416
Grupo,
BPMN, 506
Guardar,
diagrama como imagen, 735
elementos como mapas de bits, 737

H

Habilitar,
líneas de ajuste mientras se arrastran objetos, 762
Habilitar el control de código fuente, 689
Herramientas, 745
agregar al menú Herramientas, 755
opciones, 762
Hipervínculos,
en el texto de la documentación, 122
Historial,
mostrar, 702
HRESULT,
and error handling, 851

IBM DB2,
conectarse por JDBC, 592
conectarse por ODBC, 594
IBM DB2 for i,
conectarse por JDBC, 600
conectarse por ODBC, 601
IBM Informix,
conectarse por JDBC, 604
Icono,
actividad, 714
agregar a barra de herramientas / menú, 752
Business Process Modeling Notation, 731
caso de uso, 729
ciclo de vida, 728
clase, 716
componentes, 719
comunicación, 717
esquema XML, 730
estructura de un compuesto, 718
implementación, 720
interacción global, 721
máquina de estados, 727
mostrar iconos de tamaño grande, 762
objeto, 722
paquete, 723
secuencia, 726
Iconos,
visibilidad, 433
Iconos de los diagramas de UModel, 713
ID,
identificadores ID y UUID, 643
Imágenes,
usar como fondo de elemento, 123
Implementación,
diagrama, 59
diagrama de, 454
iconos, 720
ImportaciónDeElemento,
ver, 90
ImportaciónDePaquete, 457
ver, 90
Importar, 643
archivo XMI, 643
archivos binarios, 217
base de datos SQL, 538
esquema XML, 475
XMI generado con UModel, 643
Imprimir,
vista previa, 735
Incluir, 168, 169
.NET Framework, 168
cambiar estado, 169
compartir paquete y diagrama, 169
dependencia, 21
proyecto de UModel, 168
Información legal, 1364
Información rápida, 762
mostrar accesos directos en, 762
ver, 762
Información sobre derechos de autor, 1364
Ingeniería de código,
advertencias, 95
del código al modelo, 73
del modelo al código, 64
errores, 95
generar RealizacionesDeComponente, 232
mensajes de informaciçon, 95
mover archivo de proyecto a una ubicación nueva, 157
resolver asociaciones, 146
Ingeniería directa, 64
Ingeniería inversa, 73
C++, 204
Iniciar,
con el proyecto anterior, 762
UModel, 18
Insertar, 340
acción (CallBehavior), 340
acción (CallOperation), 340
elementos en diagrama de paquetes, 457
elementos en diagrama global de interacción, 390
elementos en diagramas de ciclo de vida, 425
elementos en estructura de un compuesto, 452
estado simple, 358
Instalación,
carpeta de ejemplos, 17
Instalador,
multiusuario, 17
Instancia,
diagrama, 46
objeto, 46
varias clases, 433

Integración,
paquete de integración para el complemento para VS .NET, 645
Inteligente,
finalización automática, 32
Interacción,
diagrama global de, 389
Interacción global,
iconos, 721
Interfaz, 433
ball and socket (notación de forma esférica), 433
implementar, 433
Interfaz del usuario, 80
configurar con ayuda del complemento, 826
Invalidar,
clase base, 433
operaciones de clase, 433
plantillas SPL predeterminadas, 231
Invariante de estado, 404
Ir a,
línea de vida, 398

J

Java,
código y nombres de archivos de clases, 234
generar código, 173, 186
importar archivo binario, 217
importar archivos .class en un modelo, 225
importar archivos .jar en un modelo, 222
importar código fuente, 202
opciones de generación de código, 179
opciones de importación de código, 205
JavaScript,
error handling, 851
JDBC,
como interfaz de conexión de datos, 560
conectarse a Teradata, 636
configurar una conexión (Windows), 581
Jerarquía del sistema,
diagrama de paquetes SysML, 530
Jerarquías (diagrama),
niveles en la documentación, 327

L

Licencia, 1368
información sobre, 1364
Licencia del producto de software, 1368
Línea,
ortogonal, 53
Línea de tiempo,
definir cambios de estado, 425
Línea de vida,
atributos, 398
ir a, 398
propiedad de tipo, 398
valor general, 425
Línea nueva,
en línea de vida, 386
operandoDeInteracción, 399
Líneas,
cambiar el estilo de, 141
de ajuste, 762
directa, 141
formato, 46
ortogonal, 141
personalizada, 141
Líneas de ajuste, 21
Llamada,
mensaje, 404
Llamada del mensaje,
operación ir a, 404

M

Macros,
desarrollar, 794
ejecutar, 813
habilitar, 801, 812
Manifestación,
artefacto, 59
Mapa de bits,
guardar elementos como, 737
Máquina de estados,
diagrama de, 356
diagrama SysML, 535

Máquina de estados,
elementos, 378
estados compuestos, regiones, 365
estados, actividades, transiciones, 358
iconos, 727
insertar elementos, 357
Marca de graduación,
diagrama de ciclo de vida, 428
MariaDB,
conectarse por ODBC, 606
Medición de licencias,
en los productos de Altova, 1366
Mejorar,
rendimiento, 172
Mensaje, 404
crear objeto, 404
diagrama de ciclo de vida, 431
flechas, 404
insertar, 404
llamada, 404
mover, 404
numeración, 404
operación ir a, 404
Menú, 760
agregar / eliminar comando, 752
agregar menú a, 755
archivo, 735
ayuda, 775
diseño, 742
edición, 737
eliminar comandos de, 760
herramientas, 745
personalizar, 760
predeterminado/XMLSPY, 760
proyecto, 739
ventanas, 773
vista, 744
Menú contextual,
comandos, 760
Metadatos,
salida XMI, 643
Método,
agregar excepción agregada, 433
generar diagrama de secuencia a partir de métodos, 410
generar diagrama de secuencia a partir de métodos getter/setter, 416
generar varios diagramas de secuencia a partir de métodos, 416
Métodos,
getter / setter, 433
Microsoft Access,
conectarse por ADO, 566, 608
Microsoft SQL Server,
conectarse por ADO, 610
conectarse por ODBC, 613
MisDocumentos,
archivos de ejemplo, 17
Modelado,
mejorar el rendimiento, 172
Modelo,
agregar elementos a, 109
añadir elementos a, 82, 109
cambiar el nombre de las clases (efecto en Java), 234
Modelo Java,
convertir en C++, 305
Modelos,
transformar un modelo en otro, 312
Mostrar,
ocultar slot, 433
valores etiquetados, 475
Mostrar / ocultar,
atributos, operaciones, 433
Mostrar diferencias, 704
Mostrar historial, 702
Mover,
proyecto, 157
Mover las flechas de los mensajes, 404
MS Visual Source Safe, 682
MS Visual Studio .Net,
archivo de proyecto CSPROJ - CSDPROJ, 739
MS VS .NET,
complemento de UModel, 645
Multilínea, 21
caso de uso, 21
operandoDeInteracción, 399
texto del actor, 21
Multiusuario,
carpeta de ejemplos, 17
MySQL,
conectarse por ODBC, 619

N

Nodo, 59

Nodo, 59
agregar, 59
agregar artefacto, 59
estilos, 89
propiedades de los bloques como, 521
Nombre,
de región (ver/ocultar), 365
Nuevo,
clasificador, 232
Numeración,
mensajes, 404

O

Object model,
overview, 835
Objeto,
crear mensaje, 404
iconos, 722
vínculos (asociaciones), 46
Objeto de datos,
BPMN, 506
Objetos,
diagrama, 46
diagrama de, 455
Objetos de conexión, 503
Objetos de flujo, 493
Obtener archivo,
control de código fuente, 691
Obtener carpetas,
control de código fuente, 692
Obtener la versión más reciente, 690
Ocultar,
mostrar slot, 433
ODBC,
como interfaz de conexión de datos, 560
conectarse a MariaDB, 606
conectarse a Teradata, 638
configurar una conexión, 578
OLE DB,
como interfaz de conexión de datos, 560
Omitir, 762
directorios, 762
elementos en la lista, 762
nombres de operación, 410
Opciones, 762
control de código fuente, 762
herramientas, 762
OpenJDK,
importar binarios, 218
Operación, 433
agregar automáticamente en actividad, 358
invalidar, 433
ir a (en la llamada del mensaje), 404
mostrar / ocultar, 433
omitir en la generación de diagramas de secuencia, 410
plantilla, 299
ventana de finalización automática, 762
volver a utilizar, 41
Operaciones,
agregar, 32
Operador,
interacción, 399
Operador de interacción,
definir, 399
Operando,
interacción, 399
Operando de interacción,
multilínea, 399
Operation,
coloring, 438
Oracle,
conectarse por JDBC, 621
conectarse por ODBC, 623
Ortogonal,
estado, 365
línea, 53
Ortografía,
corrector, 93

P

Página,
evitar división entre las páginas, 735
Panel Árbol de diagramas, 86
Panel Capas, 94
Panel Documentación, 93
Panel Estilos, 89
Panel Estructura del modelo,
expandir o contraer elementos, 82
explorar el proyecto desde, 82
mostrar u ocultar elementos, 82

Panel Estructura del modelo,
ordenar elementos, 82
relación de iconos, 82
Panel Jerarquía, 90
Panel Mensajes,
referencia, 95
Panel Propiedades,
añadir propiedades personalizadas, 88
Panel Vista general,
desplazarse, 92
Paquete,
compartir, 169
iconos, 723
paquetes predeterminados, 82
relación de iconos, 82
Paquetes,
diagrama de, 456
generar diagrama de dependencias entre, 456
Paramétrico, 529
diagrama SysML, 529
restricciones (definir), 529
Parámetro,
de procesamiento por lotes, 101
plantilla, 299
Parcial,
generar documentación parcial, 327
Perfiles,
aplicar a un paquete, 164, 462
built-in, 462
crear, 462
definición, 461
Período de evaluación,
de los productos de software de Altova, 1364, 1365
Personalizar, 751
actor, 21
comandos de las barras de herramientas / menús, 752
menú, 760
menú contextual, 760
Pestañas de búsqueda, 80
Plantilla,
enlace, 298
firma, 296, 297
operación/parámetro, 299
Plantillas,
plantillas SPL, 1350
SPL definidas por el usuario, 231
Plantillas SPL,
ruta de acceso, 1350
PNG,
guardar diagrama, 735
Por lotes, 106
modo de procesamiento por lotes, 106
nuevo / cargar / guardar, 106
procesamiento, 101, 106
PostgreSQL,
conectarse directamente (de forma nativa), 586
conectarse por ODBC, 629
Predeterminadas,
plantillas SPL, 231
Predeterminado,
código de proyecto, 762
menú, 760
Pretty print,
preparar proyecto para pretty print al guardarlo, 157
salida XMI, 643
Procesamiento por lotes,
modo de procesamiento por lotes, 106
Proceso,
subproceso contraído, 501
subproceso expandido, 500
Progress OpenEdge (base de datos),
conectarse por JDBC, 630
conectarse por ODBC, 632
Project,
modularize, 165
split into subprojects, 165
Property,
coloring, 438
Propiedad,
con tipo (ver), 299
crear valores iniciales, 524
de tipo línea de vida, 398
volver a utilizar, 41
Propiedades,
agregar, 32
control de código fuente, 706
Propiedades de los bloques,
como nodos, 521
Proteger, 695
Proveedor,
de control de código fuente, 682
seleccionar, 686
Proyecto, 739, 762
abrir último proyecto al iniciar UModel, 762
actualizar archivo de proyecto, 231
agregarlo al control de código fuente, 697

Proyecto, 739, 762
añadir o eliminar elementos, 82
código predeterminado, 762
combinar, 290
crear, 157
enviar por correo electrónico, 735
estilos, 89
explorar, 82
flujo de trabajo, 157
fusión a 3 bandas, 291
fusión a 3 bandas manual, 292
generar documentación, 327
guardar los diagramas abiertos, 762
guardar para pretty print, 157
incluir proyecto de UModel, 168
insertar paquete, 157
mover, 157
quitarlo del control de código fuente, 700
revisión de la sintaxis, 739
Proyecto local, 686
Puerta,
diagrama de secuencia, 403
Puerto, 524
conjugado (isConjugated), 524
crear atómico, 524
crear puerto de flujo, 524
crear puerto de flujo conjugado atómico, 524
insertar un puerto de flujo, 524
unir dos puertos, 524
Puerto de flujo,
crear puerto de flujo atómico conjugado, 524
definir la dirección de, 524
insertar, 524
Punto de entrada,
agregar a submáquina, 365
Punto de salida,
agregar a submáquina, 365
Puntos de vista,
diagrama de paquetes SysML, 530
PVCS Version Manager, 682

Q

Quitar,
del control de código fuente, 700

R

Raíz,
catálogo, 762
como paquete, 112
sincronizar raíz / paquetes / clases, 231
Rama,
crear en diagrama de actividades, 342
Realización,
componente, 53
Realizaciones,
generar RealizacionesDeComponente, 232
RealizacionesDeComponente,
generación automática, 232
Rechazar cambios, 696
Recuperar archivo,
control de código fuente, 690
Recursos,
acelerar actualización de estado en segundo plano, 682
Referencia, 734
Región,
agregar a estado compuesto, 365
nombre de región (ver/ocultar), 365
Registrar,
complemento, 824
Relaciones,
agregación, 139
asociación, 110, 139
cambiar el estilo de, 141
composición, 139
dependencia, 139
generalización, 110, 139
realización, 139
ver, 142
Relativos,
vínculos relativos en la documentación, 327
Rendimiento,
mejorar, 172
Repositorios, 682
Respuesta,
mensaje (generar automáticamente la respuesta), 404
Restaurar,
acceso directo, 758
barras de herramientas y ventanas, 745
comandos de la barra de herramientas, 753

Restaurar,
comandos de menú, 760
Restricción,
definir parámetros, 529
revisar sintaxis, 739
RestricciónDeDuración,
diagrama de ciclo de vida, 430
RestricciónDeTiempo,
diagrama de ciclo de vida, 431
Restringir,
clasificadores, 296
Revisar,
sintaxis del proyecto, 739
Ruta de acceso,
cambiar ubicación del proyecto, 157
carpeta de ejemplos, 17
plantillas SPL, 1350

S

Salida,
archivo XMI de salida, 643
Salto de línea,
en el texto del actor, 21
SC,
syntax coloring, 438
Secuencia,
diagrama de, 394
diagramas SysML, 534
iconos, 726
Segundo plano,
actualización de estado - aumentar el intervalo, 682
Sequence diagram,
generate from code, 841
generate manually from code, 842
Set,
métodos getter / setter, 433
Setter / Getter,
generar diagrama de secuencia a partir de métodos setter/getter, 416
Símbolos,
iconos de visibilidad, 433
Sincronización,
cambiar nombres de las clases, 234
opciones de configuración, 231
Sincronizar,
en la ubicación nueva, 157
nombre de las clases y el nombre del archivo de código, 234
raíz / paquetes / clases, 231
Sintaxis,
archivo de procesamiento por lotes, 101
revisar sintaxis del proyecto, 739
Sintaxis del proyecto,
revisar, 95
Slot,
mostrar / ocultar, 433
Sobrescribir,
código con el modelo, 739
modelo con el código, 739
Socket,
Ball and socket (notación de forma esférica), 433
SPL, 1348
bloques de código, 1349
condiciones, 1359
foreach, 1360
plantillas SPL definidas por el usuario, 231
subrutinas, 1362
SQL,
importar en UModel, 538
SQL Server,
conectarse por ADO, 566
conectarse por ADO.NET, 571
SQLite,
configurar una conexión (Windows), 587
StarTeam, 682
Subproceso,
contraído, 501
expandido, 500
Subproject,
create from main project, 165
reintegrate into main project, 165
Sybase,
conectarse por JDBC, 635
Syntax coloring, 438
SysML, 520
diagrama de actividades, 533
diagrama de bloque interno, 524
diagrama de casos de uso, 536
diagrama de definición de bloques, 521
diagrama de máquina de estados, 535
diagrama de paquetes, 530
diagrama de requisitos, 532
diagrama de secuencia, 534
diagrama paramétrico, 529

SysML, 520
insertar un diagrama SysML, 520
introducción, 520

T

Tecla de acceso rápido, 758
Teradata,
conectarse por JDBC, 636
conectarse por ODBC, 638
Tipo,
propiedad (ver), 299
propiedad de tipo línea de vida, 398
Tipo de diagrama,
identificar, 97
Todos,
expandir / contraer, 433
Transformación,
opciones de configuración, 303
Transformar,
C# en Java, 312
Transformar modelos, 300
Transición, 358
agregar diagrama de actividades a, 358
definir disparador, 358
definir entre estados, 358
Tres bandas,
fusión del proyecto, 291
fusión manual del proyecto, 292
Tutorial, 17
archivos de ejemplo, 17
carpeta de ejemplos, 17
objetivos, 17

U

Ubicación,
mover proyecto, 157
UML,
compartir diagrama UML, 169
diagramas, 338
iconos de visibilidad, 433
plantillas, 296
variables, 1350
UModel,
complemento para VS .NET, 645
importar XMI generado, 643
iniciar, 18
UModel API,
overview of, 834
UModel Type Library,
adding reference to, 853
UMP,
cambiar ubicación del proyecto, 157
extensión de archivo, 157
URL,
abrir archivo desde, 735
Uso,
dependencia, 53
Uso de interacción, 403
Usuario,
carpeta de ejemplos multiusuario, 17
UUID,
identificadores únicos universales, 643

V

Valor general de la línea de vida,
diagrama de ciclo de vida, 425
Valores,
etiquetados (mostrar), 475
Valores etiquetados,
como enumeraciones, 463, 466
crear, 152, 463
definición, 151
ejemplo, 466
ejemplos, 151
mostrar, 475
mostrar u ocultar, 154
Valores iniciales,
agregar a propiedad (SysML), 524
Variables,
argumentos de herramientas externas, 755
UML, 1350
VB.NET,
generar código, 173
importar código fuente, 202
opciones de generación de código, 179
opciones de importación de código, 205
Velocidad,

Velocidad,
acelerar actualización de estado en segundo plano, 682
Ventanas, 773
restaurar, 745
Ver,
propiedad como asociación, 299
u ocultar nombre de región, 365
varias instancias del elemento, 433
Vínculos,
relativos a la documentación, 327
Visibilidad,
iconos (seleccionar), 433
Vista, 744
Vista de elementos,
como paquete, 112
Visual Basic,
error handling, 851
Visual Studio,
cargar/descargar proyectos de UModel, 652
funciones de UModel para soluciones, 648
sincronización automática de código y modelo, 653
sincronizar el código con el modelo, 653
VS .NET,
complemento de UModel, 645

X

XMI, 643
extensiones, 643
pretty print, 643
XML Schema,
creating diagrams, 481
declare namespace, 481
generating from model, 483
modeling, 481, 483
XMLSPY,
complemento, 815
registrar el complemento, 824