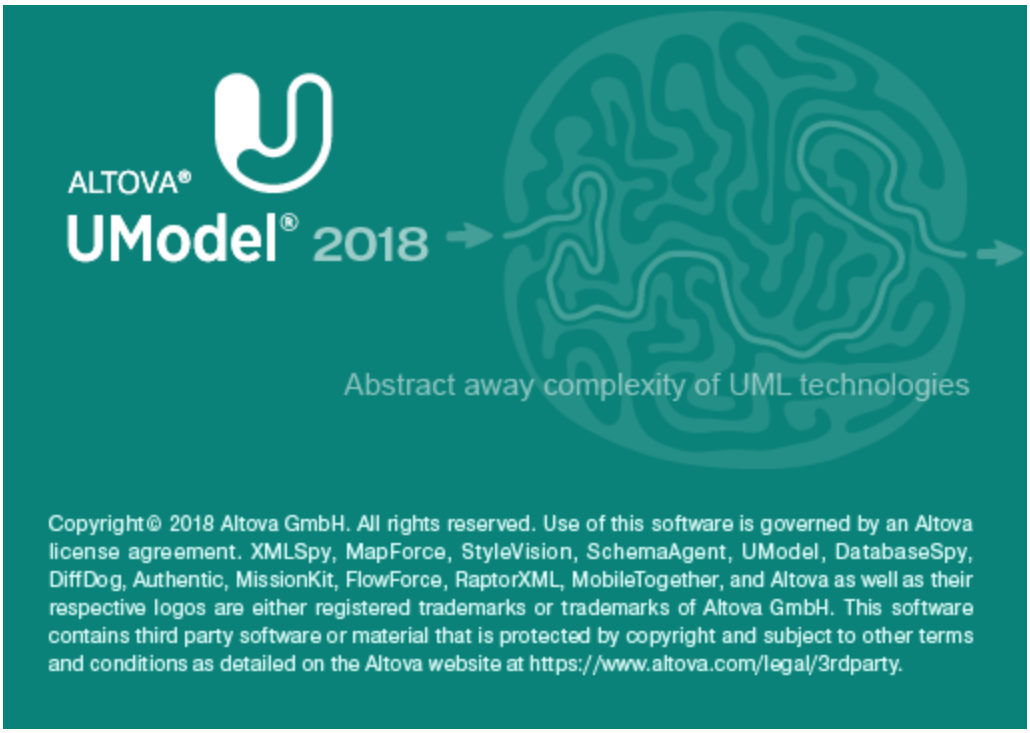
Manual del usuario y referencia



Copyright© 2018 Altova GmbH. All rights reserved. Use of this software is governed by an Altova license agreement. XMLSpy, MapForce, StyleVision, SchemaAgent, UModel, DatabaseSpy, DiffDog, Authentic, MissionKit, FlowForce, RaptorXML, MobileTogether, and Altova as well as their respective logos are either registered trademarks or trademarks of Altova GmbH. This software contains third party software or material that is protected by copyright and subject to other terms and conditions as detailed on the Altova website at https://www.altova.com/legal/3rdparty.

Manual del usuario y referencia de Altova UModel 2018

Todos los derechos reservados. Ningún fragmento de esta publicación podrá ser reproducido de manera alguna (ya sea de forma gráfica, electrónica o mecánica, fotocopiado, grabado o reproducido en sistemas de almacenamiento y recuperación de información) sin el consentimiento expreso por escrito de su autor/editor.

Los productos a los que se hace referencia en este documento pueden ser marcas registradas de sus respectivos propietarios. El autor y editor no afirman ser propietarios de dichas marcas registradas.

Durante la elaboración de este documento se tomaron todas las precauciones necesarias para prevenir errores. Sin embargo, el autor y editor no se responsabilizan de los errores u omisiones que pudiese contener el documento ni de los posibles daños o perjuicios derivados del uso del contenido de este documento o de los programas y código fuente que vengan con el documento. Bajo ninguna circunstancia se podrá considerar al autor y editor responsables de la pérdida de beneficios ni de cualquier otro daño y perjuicio derivado directa o indirectamente del uso de este documento.

Fecha de publicación: 2018

© 2018 Altova GmbH

Contenido

1 UModel 3

2 Introducción a UModel 6

2.1 Notas sobre compatibilidad ... 7

3 Tutorial de UModel 10

3.1 Introducción ... 11

3.2 Casos de uso ... 14

3.3 Diagramas de clases ... 25

3.3.1 Crear clases derivadas ... 34

3.4 Diagramas de objetos ... 39

3.5 Diagramas de componentes ... 46

3.6 Diagramas de implementación ... 52

3.7 Ingeniería directa (del modelo al código) ... 57

3.8 Ingeniería inversa (del código al modelo) ... 67

4 Interfaz gráfica del usuario de UModel 76

4.1 Ventana Estructura del modelo ... 78

4.2 Ventana Árbol de diagramas ... 82

4.3 Ventana Favoritos ... 83

4.4 Ventana Propiedades ... 84

4.5 Ventana Estilos ... 85

4.6 Ventana Jerarquía ... 86

4.7 Ventana Vista general ... 88

4.8 Ventana Documentación ... 89

4.9 Ventana Mensajes ... 90

4.10 Ventana de diagramas ... 92

4.11 Panel Diagramas ... 94

5 Interfaz de la línea de comandos 98

5.1 Archivo: Nuevo / Cargar / Opciones al guardar 102

6 Cómo modelar 106

6.1 Elementos 107
6.1.1 Crear elementos 107
6.1.2 Insertar elementos del modelo en un diagrama 108
6.1.3 Renombrar, mover y copiar elementos 110
6.1.4 Borrar elementos 111
6.1.5 Buscar y reemplazar texto 112
6.1.6 Comprobar si se están usando ciertos elementos y dónde 113
6.1.7 Restricción de elementos 114
6.1.8 Agregar hipervínculos a elementos 116
6.1.9 Documentar elementos 118
6.1.10 Cambiar el estilo de los elementos de un diagrama 120
6.2 Diagramas 122
6.2.1 Crear diagramas 122
6.2.2 Generar diagramas 123
6.2.3 Abrir diagramas 125
6.2.4 Borrar diagramas 126
6.2.5 Cambiar el estilo de diagramas 127
6.2.6 Alinear y ajustar el tamaño de elementos de modelado 128
6.2.7 Finalización automática en clases 130
6.2.8 Acercar y alejar diagramas 132
6.3 Relaciones 133
6.3.1 Crear relaciones entre elementos 133
6.3.2 Cambiar el estilo de las líneas y relaciones 135
6.3.3 Ver las relaciones de los elementos 137
6.3.4 Associations 138
6.3.5 Asociación de colecciones 141
6.3.6 Contención 144

7 Proyectos e ingeniería de código 146

7.1 Administrar proyectos de UModel 147
7.1.1 Crear, abrir y guardar proyectos 147
7.1.2 Abrir proyectos desde una URL 149
7.1.3 Mover proyectos a un directorio nuevo 152
7.1.4 Incluir otros proyectos de UModel 153
7.1.5 Compartir paquetes y diagramas 155
7.1.6 Consejos para mejorar el rendimiento 158

7.2 Generar código de programa ... 159
7.2.1 Revisar la sintaxis del proyecto ... 161
7.2.2 Opciones de generación de código ... 164
7.2.3 Ejemplo: generar código Java desde UModel ... 165
7.2.4 Plantillas SPL ... 174
7.3 Importar código fuente ... 176
7.3.1 Opciones de importación de código ... 178
7.3.2 Ejemplo: importar un proyecto C# ... 181
7.4 Importar binarios Java, C# y VB.NET ... 187
7.5 Sincronizar el modelo y el código fuente ... 193
7.5.1 Consejos prácticos ... 194
7.5.2 Refactorización de código y sincronización ... 196
7.5.3 Configurar la sincronización del código ... 197
7.6 Correspondencias con elementos de UModel ... 200
7.6.1 Correspondencias con C# ... 200
7.6.2 Correspondencias con VB.NET ... 220
7.6.3 Correspondencias con Java ... 234
7.6.4 Correspondencias con XML Schema ... 240
7.7 Combinar proyectos de UModel ... 250
7.7.1 Fusión de proyectos a 3 bandas ... 251
7.7.2 Ejemplo de fusión manual a 3 bandas ... 253
7.8 Plantillas UML ... 256
7.8.1 Firmas de plantilla ... 257
7.8.2 Enlace de plantilla ... 258
7.8.3 Usar plantillas en operaciones y propiedades ... 259

8 Generar documentación UML 262

8.1 Con una hoja de estilos SPS predeterminada ... 269
8.2 Con hojas de estilos predefinidas por el usuario ... 271

9 Diagramas UML 274

9.1 Diagramas de comportamiento ... 275
9.1.1 Diagrama de actividades ... 275
Insertar elementos ... 276
Crear bifurcaciones y convergencias ... 278
Elementos ... 281
9.1.2 Diagrama de máquina de estados ... 291
Insertar elementos ... 292

Crear estados, actividades y transiciones ... 293
Estados compuestos ... 299
Generar código a partir de diagramas de máquina de estados ... 302
Trabajar con código de máquina de estados ... 305
Elementos de diagramas de máquinas de estados ... 310
9.1.3 Diagrama de máquina de estados de protocolos ... 313
Insertar elementos ... 314
Elementos ... 315
9.1.4 Diagrama de casos de uso ... 317
9.1.5 Diagrama de comunicación ... 318
Insertar elementos ... 319
9.1.6 Diagrama global de interacción ... 322
Insertar elementos ... 323
9.1.7 Diagrama de secuencia ... 326
Insertar elementos ... 327
Líneas de vida ... 329
Fragmentos combinados ... 330
Usos de interacción ... 334
Puertas ... 334
Invariantes de estado ... 335
Mensajes ... 335
Generar diagramas de secuencia a partir de código fuente ... 340
Generar varios diagramas de secuencia a partir de propiedades ... 343
Generar código a partir de diagramas de secuencia ... 345
Agregar código a diagramas de secuencia ... 349
9.1.8 Diagrama de ciclo de vida ... 352
Insertar elementos ... 353
Línea de vida ... 353
Marca de graduación ... 356
Evento/estímulo ... 357
Restricción de duración ... 358
Restricción de tiempo ... 359
Mensaje ... 359
9.2 Diagramas de estructura ... 361
9.2.1 Diagrama de clases ... 361
Personalizar diagramas de clases ... 361
Invalidar operaciones de clases base e implementar operaciones de interfaz ... 368
Crear métodos getter y setter ... 369
Notaciones de forma esférica (Ball and socket) ... 371

Agregar excepciones emitidas a los métodos de una clase ... 372
Generar diagramas de clases ... 373
9.2.2 Diagrama de estructura de un compuesto ... 376
Insertar elementos ... 376
9.2.3 Diagrama de componentes ... 378
9.2.4 Diagrama de implementación ... 378
9.2.5 Diagrama de objetos ... 379
9.2.6 Diagrama de paquetes ... 379
Insertar elementos ... 381
Generar diagramas de paquetes al importar código o binarios ... 383
9.2.7 Diagrama de perfil y estereotipos ... 385
Agregar estereotipos y definir valores etiquetados ... 386
Estereotipos y enumeraciones ... 390
Estilos de estereotipo definidos por el usuario ... 392
Asignar iconos de estereotipo personalizados ... 394
9.3 Otros diagramas ... 398
9.3.1 Diagramas de esquema XML ... 398
Importar esquemas XML ... 399
Insertar elementos ... 406
Crear y generar un esquema XML ... 409

10 XMI: intercambio de metadatos XML 414

11 Trabajo en equipo con UModel 418
11.1 Crear y editar subproyectos ... 420

12 Control de código fuente 428
12.1 Sistemas de control de código fuente compatibles ... 430
12.2 Comandos de control de código fuente ... 432
12.2.1 Abrir desde el control de código fuente ... 432
12.2.2 Habilitar control de código fuente ... 435
12.2.3 Obtener la versión más reciente ... 436
12.2.4 Obtener ... 436
12.2.5 Obtener carpetas ... 438
12.2.6 Desproteger ... 439
12.2.7 Proteger ... 440

12.2.8 Anular desprotección ... 441
12.2.9 Agregar al control de código fuente ... 443
12.2.10 Quitar del control de código fuente ... 445
12.2.11 Compartir desde el control de código fuente ... 446
12.2.12 Mostrar historial ... 447
12.2.13 Mostrar diferencias ... 449
12.2.14 Mostrar propiedades ... 450
12.2.15 Actualizar estado ... 451
12.2.16 Administrador del control de código fuente ... 451
12.2.17 Cambiar control de código fuente ... 451
12.3 Control de código fuente con Git ... 453
12.3.1 Habilitar Git con el complemento de control de código fuente ... 454
12.3.2 Agregar un proyecto al control de código fuente de Git ... 454
12.3.3 Clonar un proyecto desde el control de código fuente de Git ... 455

13 Iconos en los diagramas de UModel 460

13.1 Diagramas de actividades ... 461
13.2 Diagramas de clases ... 463
13.3 Diagramas de comunicación ... 464
13.4 Diagramas de estructura de un compuesto ... 465
13.5 Diagramas de componentes ... 466
13.6 Diagramas de implementación ... 467
13.7 Diagramas global de interacción ... 468
13.8 Diagramas de objetos ... 469
13.9 Diagramas de paquetes ... 470
13.10 Diagramas de perfil ... 471
13.11 Diagramas de máquina de estados de protocolos ... 472
13.12 Diagramas de secuencia ... 473
13.13 Diagramas de máquina de estados ... 474
13.14 Diagramas de ciclo de vida ... 475
13.15 Diagramas de casos de uso ... 476
13.16 Diagramas de esquema XML ... 477

14 Referencia del usuario 480

14.1 Menú Archivo ... 481
14.2 Menú Edición ... 483
14.3 Menú Proyecto ... 485

14.4 Menú Diseño ... 488
14.5 Menú Vista ... 490
14.6 Menú Herramientas ... 491
14.6.1 Herramientas definidas por el usuario ... 491
14.6.2 Personalizar ... 491
Comandos ... 491
Barras de herramientas ... 493
Herramientas ... 494
Teclado ... 497
Menú ... 499
Opciones ... 500
14.6.3 Restaurar barras de herramientas y ventanas ... 501
14.6.4 Opciones ... 501
Configuración del proxy de red ... 507
14.7 Menú Ventanas ... 510
14.8 Menú Ayuda ... 511

15 Generador de código 518

15.1 SPL: el lenguaje de programación Spy ... 519
15.1.1 Variables ... 519
15.1.2 Operadores ... 527
15.1.3 Condiciones ... 528
15.1.4 Colecciones y foreach ... 529
15.1.5 Subrutinas ... 530
Declaración de subrutinas ... 530
Invocación de subrutinas ... 531
15.2 Códigos de error ... 533

16 Información sobre licencias 536

16.1 Distribución electrónica de software ... 537
16.2 Activación del software y medición de licencias ... 538
16.3 Derechos de propiedad intelectual ... 540
16.4 Contrato de licencia para el usuario final ... 541

Índice

Altova UModel 2018

UModel

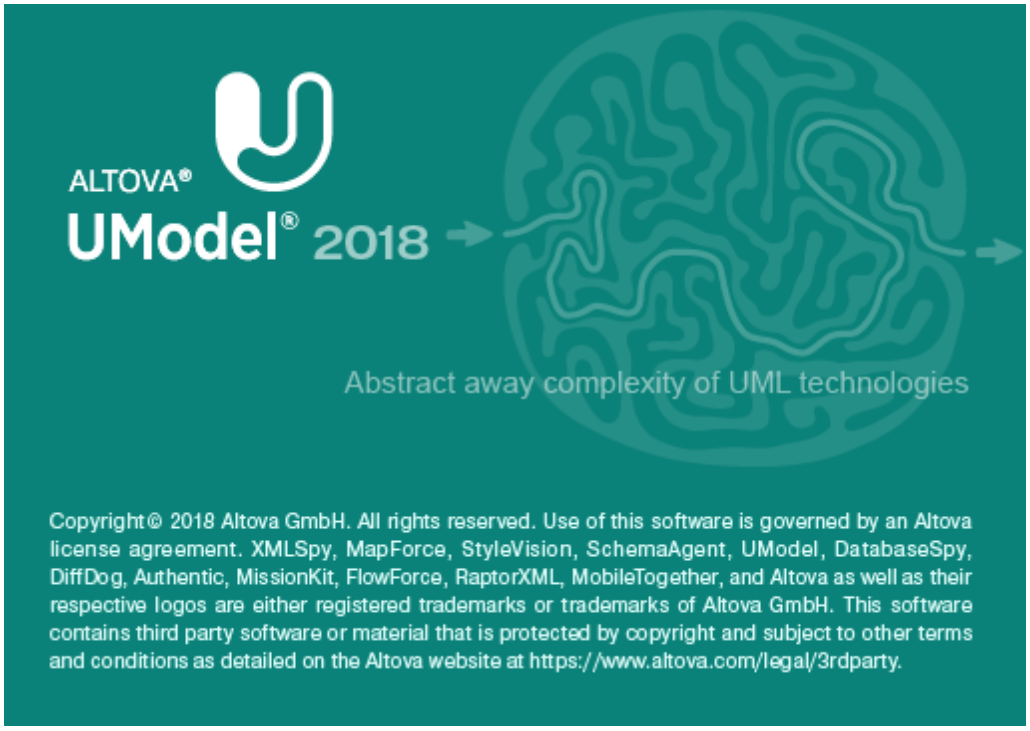
1 UModel

UModel® 2018 Basic Edition es una asequible 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 2.4. UModel es una aplicación de 32/64 bits para Windows compatible con Windows 7 SP1 con actualización de la plataforma, Windows 8, Windows 10 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.

Principales características de UModel® 2018:

- Compatible con los 14 tipos de diagrama de modelado UML 2.4
- Compatible con procesos de arquitectura dirigida por modelos (MDA), que permite realizar conversiones entre diferentes lenguajes de programación (solo en la edición Enterprise Edition)
- Opción para importar y exportar bases de datos SQL en UModel (solo en las ediciones Enterprise y Professional)
- Funciones para trabajar en equipo y editar proyectos de forma simultánea
- Fusión de proyectos a 3 bandas
- Máquinas de estado de protocolos
- Compatible con diagramas SysML (solo en las ediciones Enterprise y Professional)
- Generación de diagramas de secuencia a partir del código fuente
- Generación de código a partir de diagramas de máquina de estados
- API y complemento de UModel
- Entorno de scripting y editor de formularios integrados (solo en las ediciones Enterprise y Professional)
- Integración en Eclipse
- Compatible con sistemas de control de versiones
- Compatible con diagramas de XML Schema
- Compatible con BPMN (Business Process Modeling Notation) 1.0 y 2.0
- Opción para usar varias capas en el diagrama UML
- Opción para importar binarios Java, C# y Visual Basic
- Opción para crear hipervínculos entre diagramas y elementos de modelado
- Color de sintaxis en los diagramas
- Estilos en cascada
- Número ilimitado de operaciones Deshacer/Rehacer
- Sofisticada generación de código C#, Java y Visual Basic a partir de los modelos
- Ingeniería inversa de código fuente C#, Java y VB.NET
- Ingeniería de ida y vuelta para combinar el código con el modelo
- Importación y exportación de modelos con XMI 2.4 para UML 2.0, 2.1, & 2.1.2, 2.2, 2.3, 2.4
- Generación de documentación de proyectos de UModel

Todas estas características le ayudarán a incrementar la productividad y optimizar resultados con UML.



UML®, OMG™, Object Management Group™ y Unified Modeling Language™ son marcas o marcas registradas de Object Management Group, Inc. en EE UU y otros países.

Última actualización: 6/22/2018

Altova UModel 2018

Introducción a UModel

2 Introducción a UModel

Sitio web de Altova: Herramienta de modelado UML

UML es un lenguaje de modelado completo pero no abarca una metodología para procesos de desarrollo, generación de código ni ingeniería de ida y vuelta. UModel está diseñado para ofrecer una total flexibilidad 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.
- 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.
- En UModel, la generación de código se realiza sobre plantillas SPL y, por tanto, se puede personalizar. Las personalizaciones son detectadas automáticamente durante la generación de código.
- La función de generación de código y de ingeniería inversa es compatible con estos lenguajes

Lenguaje	Versión
C#	1.2, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0
Java	1.4, 5.0 (1.5), 6, 7, 8, 9
Visual Basic .NET	7.1, 8.0, 9.0

- Cada proyecto puede ser compatible con Java, C# y VB simultáneamente.
- UModel es compatible con plantillas y genéricos UML.
- Intercambio de metadatos XML (XMI 2.1 para UML 2.0, 2.1.2, 2.2, 2.3 y XMI 2.4 para UML 2.4.1)
- Cuando se añaden propiedades u operaciones, UModel ofrece ayudantes de entrada contextuales donde el usuario puede elegir tipos, nivel de protección y otras propiedades normalmente disponibles en entornos IDE como XMLSpy, Visual Studio o Eclipse.
- 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.
- Puede definir actores personalizables en diagramas de casos de uso para ilustrar terminales y otros símbolos.
- Puede buscar elementos de modelado por su nombre en la ventana de diagramas, en las ventanas *Estructura del modelo* y *Documentación* y en la ventana *Mensajes*.
- Puede buscar/resaltar clases, asociaciones de objetos, dependencias, generalizaciones, etc. con ayuda del menú contextual de los diagramas.
- Las operaciones de Deshacer/Rehacer no solo abarcan cambios en el contenido sino también cambios de estilo realizados en los elementos del modelo.

Nota: el objetivo de la presente documentación no es describir ni explicar el lenguaje UML. Su objetivo es explicar al usuario cómo utilizar la herramienta de modelado Altova UModel para modelar código y completar procesos de ingeniería de ida y vuelta correctamente.

© 2018 Altova GmbH

2.1 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

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

Altova UModel 2018

Tutorial de UModel

3 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\UModel2018\UModelExamples\Tutorial**:

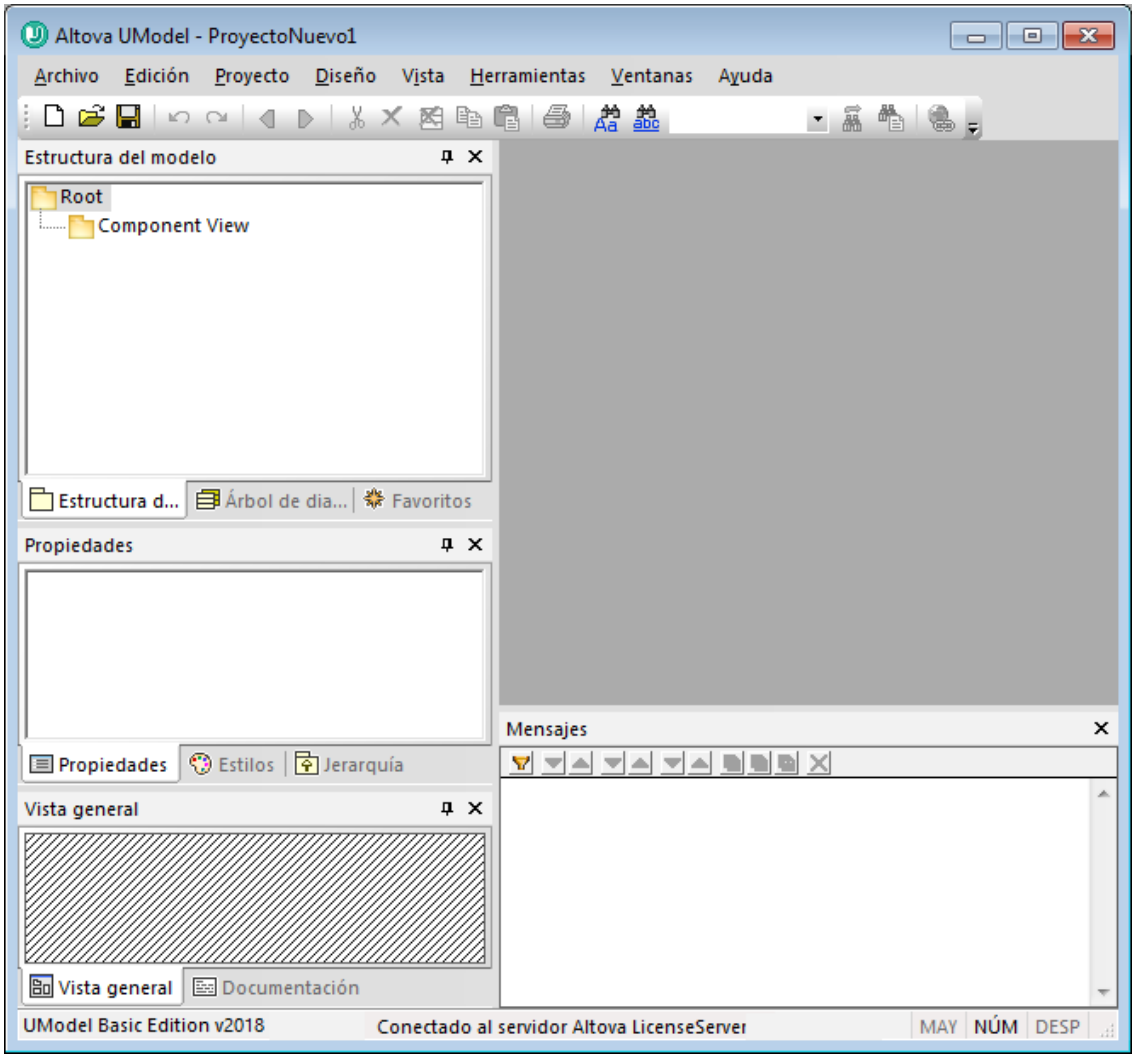
BankView-start.ump	Se trata del archivo de proyecto de UModel que representa el estado inicial del tutorial. En este proyecto existen varios diagramas de modelado, así como clases, objetos y otros elementos de modelado. A lo largo del tutorial se añadirán elementos y diagramas nuevos y se editarán elementos del proyecto. **Nota:** el hecho de que proyecto esté incompleto es deliberado. 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\UModel2018**. Cuando se inicia la aplicación por primera vez, los archivos de ejemplo se copian en el directorio **C:\Usuarios\<usuario>\Documentos\Altova\UModel2018\UModelExamples**. Por tanto, evite mover, editar o eliminar los archivos de ejemplo del directorio inicial.

© 2018 Altova GmbH

3.1 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:

- La ventana *Estructura del modelo* muestra todos los elementos de modelado del

proyecto de UModel. Los elementos se pueden manipular en esta pestaña directamente, usando las teclas de edición estándar y operaciones de arrastrar y colocar.

- La ventana *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.
- La ventana *Favoritos* es un repositorio de elementos de modelados 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 izquierdo ofrece varias vistas de determinadas propiedades del modelo:

- La ventana *Propiedades* muestra las propiedades del elemento que está seleccionado en la *Estructura del modelo* o en el *de diagramas*. Las propiedades de los elementos se pueden definir y actualizar en esta pestaña.
- La ventana *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.
- La ventana *Jerarquía* muestra todas las relaciones que tiene el elemento de modelado seleccionado. El elemento de modelado puede estar seleccionado en el diagrama, en la *Estructura del modelo* o en *Favoritos*.

La ventana inferior izquierda está compuesta por:

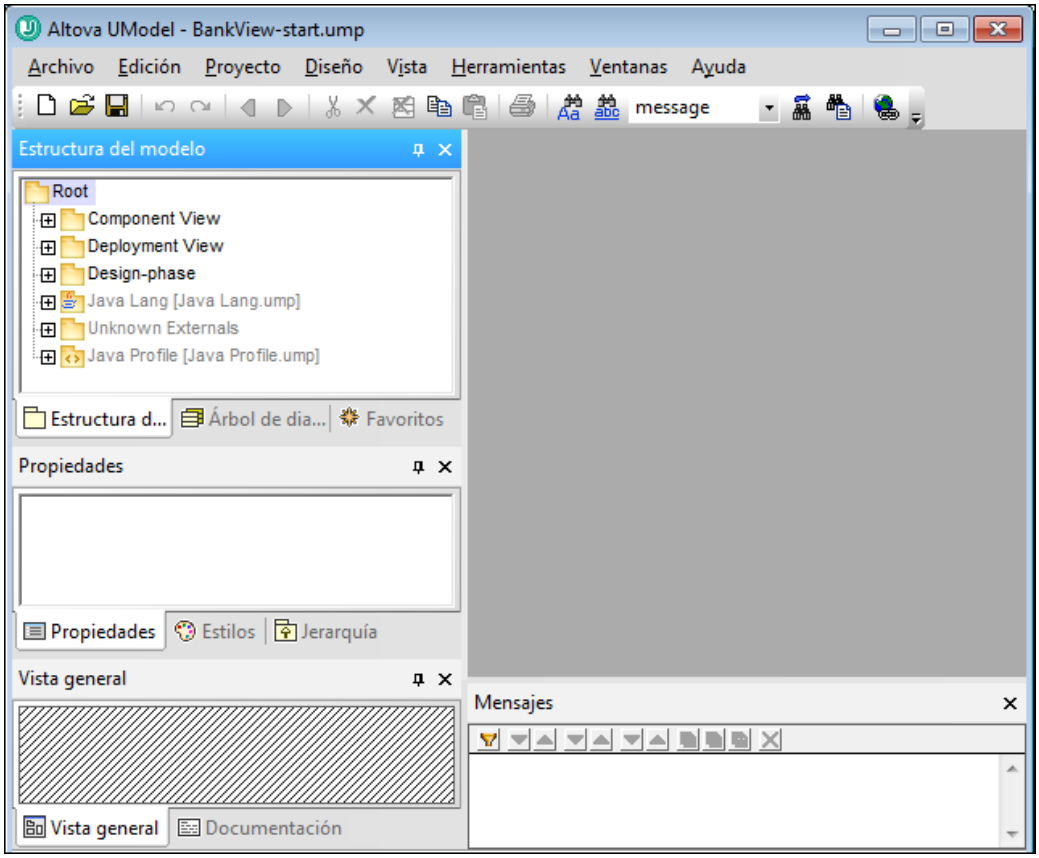
- La ventana *Vista general*, que muestra la estructura general del diagrama activo.
- La ventana *Documentación*, que sirve para documentar las clases una a una.

En este tutorial se trabajará sobre todo en las ventanas *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.

© 2018 Altova GmbH



Proyecto BankView-start.ump

3.2 Casos de uso

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

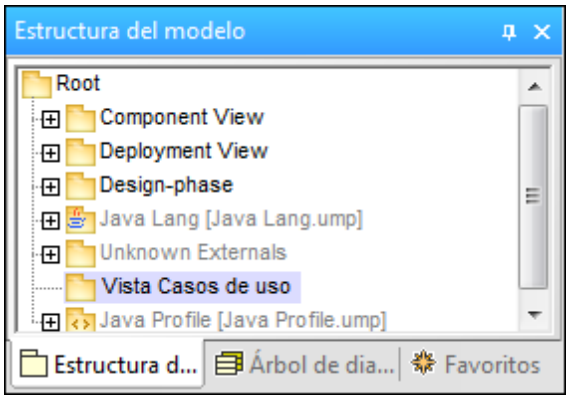
- agregar un **paquete** nuevo al proyecto;
- agregar un **diagrama** de casos de uso nuevo al proyecto;
- agregar **elementos** al diagrama y definir dependencias entre ellos;
- alinear los elementos y ajustar su tamaño;
- cambiar el estilo de todos los diagramas de un proyecto de UModel.

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. Recuerde que UModel no exige que los diagramas residan en paquetes. Sin embargo, se recomienda organizar los diagramas en paquetes.

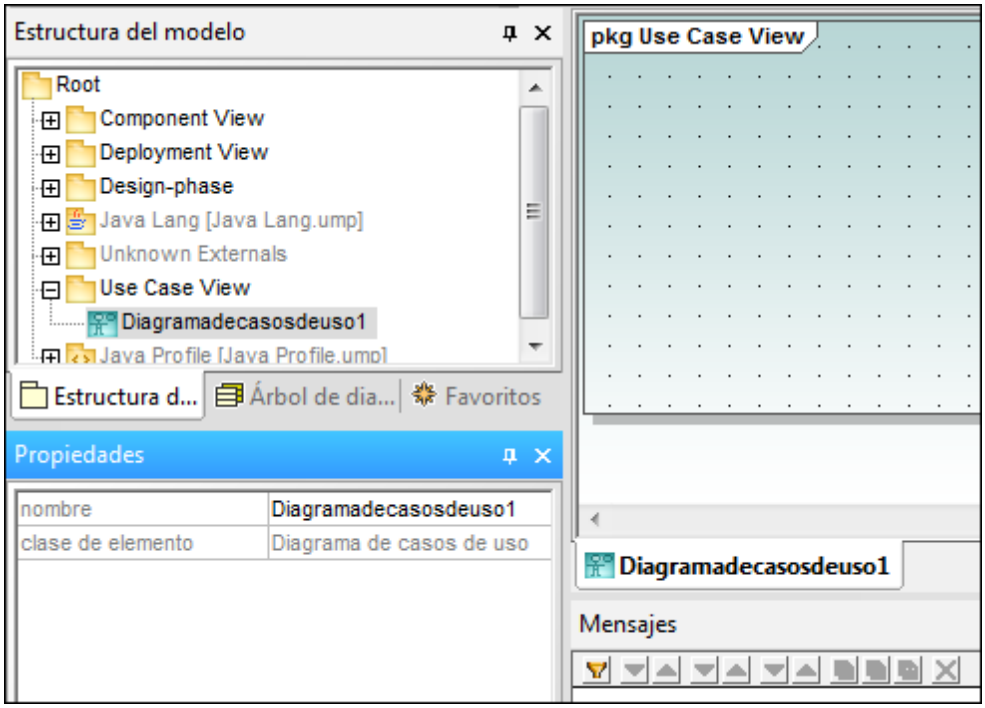
1. En la ventana *Estructura del modelo* haga clic con el botón derecho en el paquete `Root` y seleccione **Elemento nuevo | Paquete**.
2. Escriba el nombre del nuevo paquete (p. ej. `Vista Casos de uso`) y pulse **Entrar**.



Agregar un diagrama de caso de uso a un proyecto

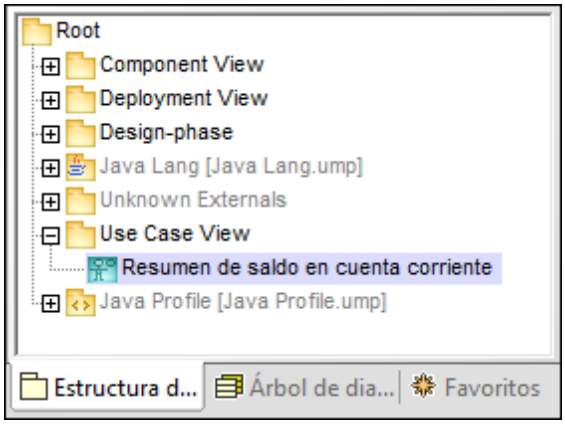
1. Haga clic con el botón derecho en el paquete `Vista Casos de uso` que acaba de crear.
2. Seleccione el comando **Diagrama nuevo | Diagrama de casos de uso**.

 © 2018 Altova GmbH



Al paquete se añade un diagrama de casos de uso (en la *Estructura del modelo*) y en el panel *Diagramas* aparece la pestaña *Diagramacasosdeuso1*, que lleva el nombre predeterminado creado automáticamente para el diagrama.

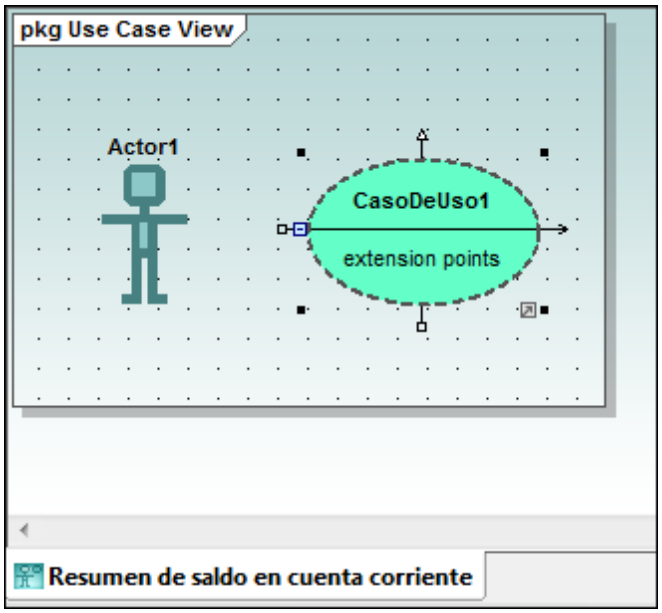
3. En la *Estructura del modelo* haga doble clic en el nombre del diagrama, escriba `Resumen de saldo en cuenta corriente` y pulse **Entrar** para confirmar.



Agregar elementos al diagrama de casos de uso

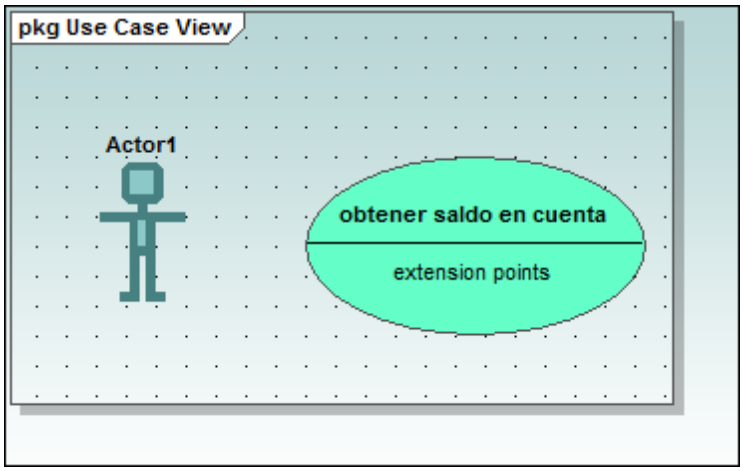
1. Haga clic con el botón derecho dentro de la ventana del diagrama recién creado y seleccione **Nuevo/a | Actor**.
   El elemento `actor` se inserta en la posición donde se hizo clic.
2. Haga clic en el icono **Caso de uso** de la barra de herramientas y después en la ventana del diagrama para insertar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

El elemento `CasoDeUso1` se inserta en el diagrama. Observe que el elemento y su nombre están seleccionados y, por tanto, 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.



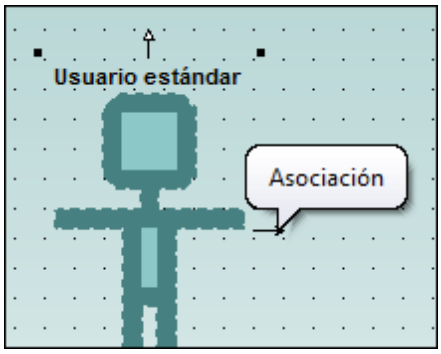
   Use **Ctrl+Entrar** para añadir un salto de línea en el nombre del caso de uso.

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.

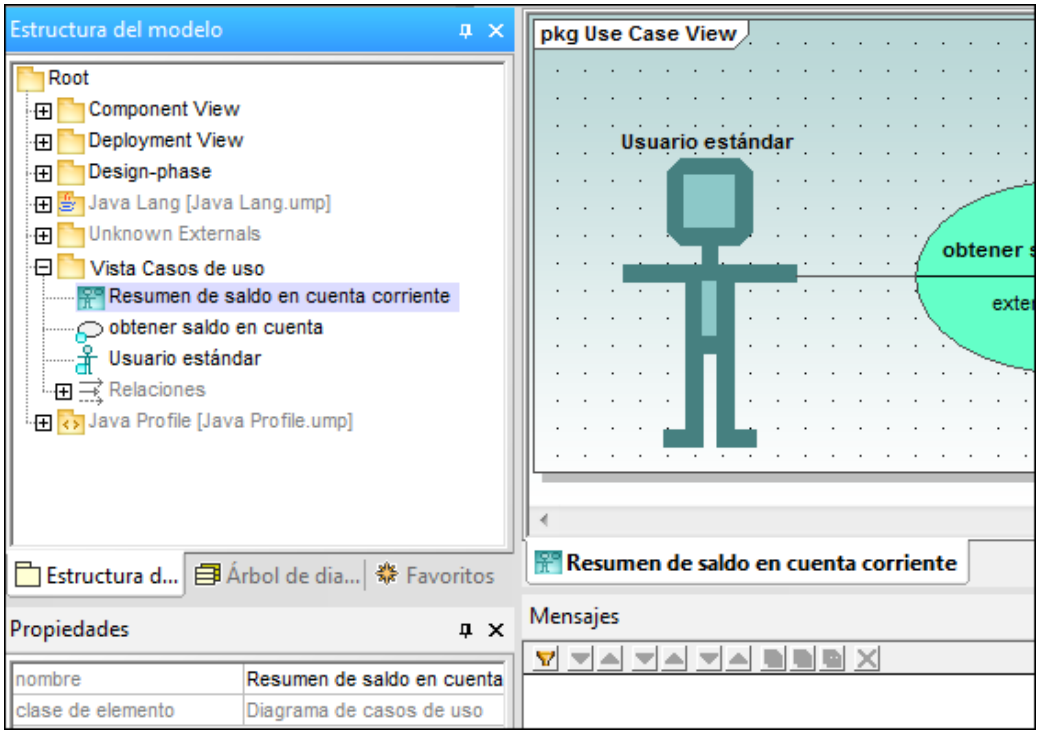
 © 2018 Altova GmbH

1. Haga doble clic en el texto **`Actor1`** del elemento `actor`, escriba el nuevo nombre **`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`.

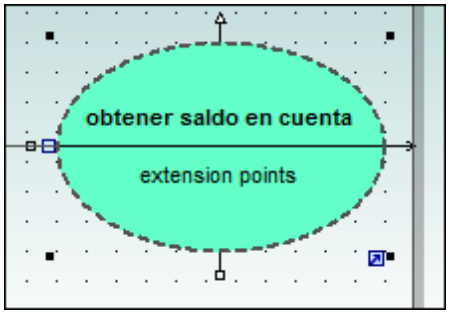


3. Haga clic en el controlador, arrastre la línea de asociación hacia la derecha y suéltela en el caso de uso **`obtener saldo en cuenta`**.

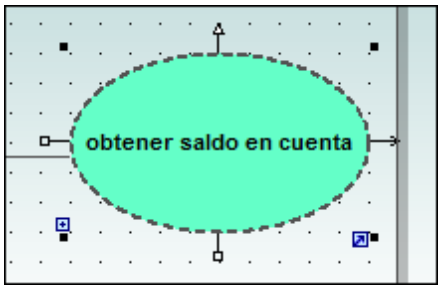
   Ahora se crea una asociación entre el actor y el caso de uso. Las propiedades de la asociación se pueden ver en la ventana *Propiedades*. La nueva asociación también se añade a la *Estructura del modelo*, bajo el elemento `Relaciones` del paquete `Vista Casos de uso`.

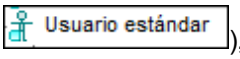


4. Haga clic en el caso de uso y arrástrelo hacia la derecha.
   Las propiedades de la asociación están visibles en el objeto asociación.
5. Haga clic en el caso de uso para seleccionarlo y después haga clic en el icono contraer situado en el borde izquierdo del caso de uso.

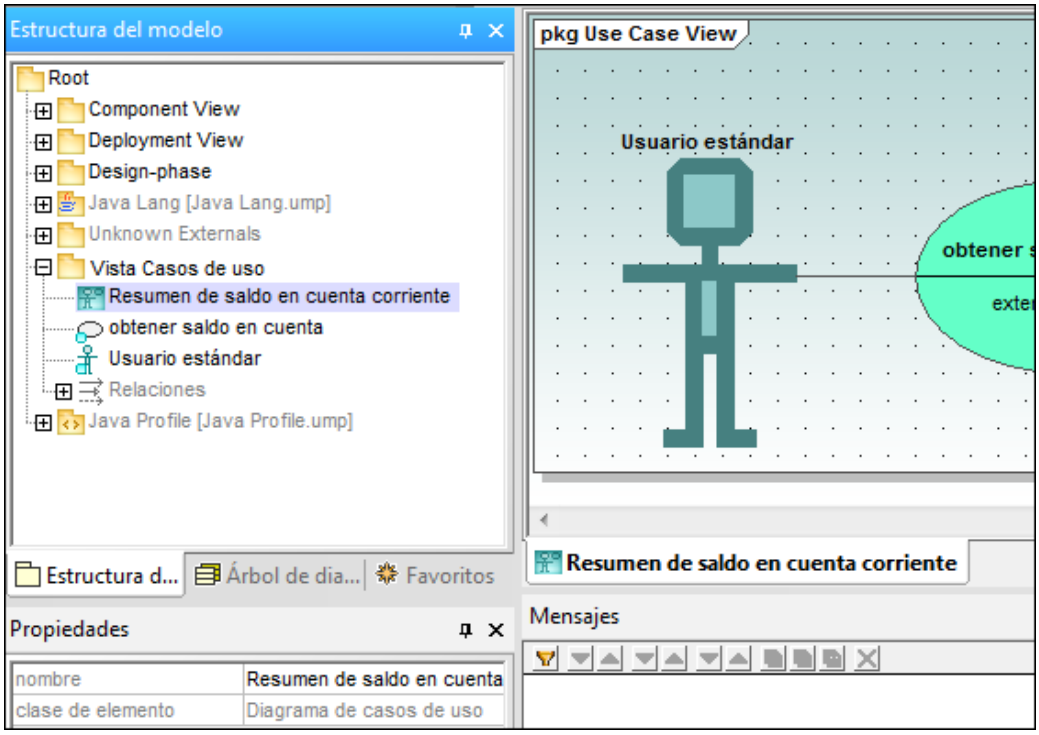


Esto oculta el compartimento `puntos de extensión`.



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*:

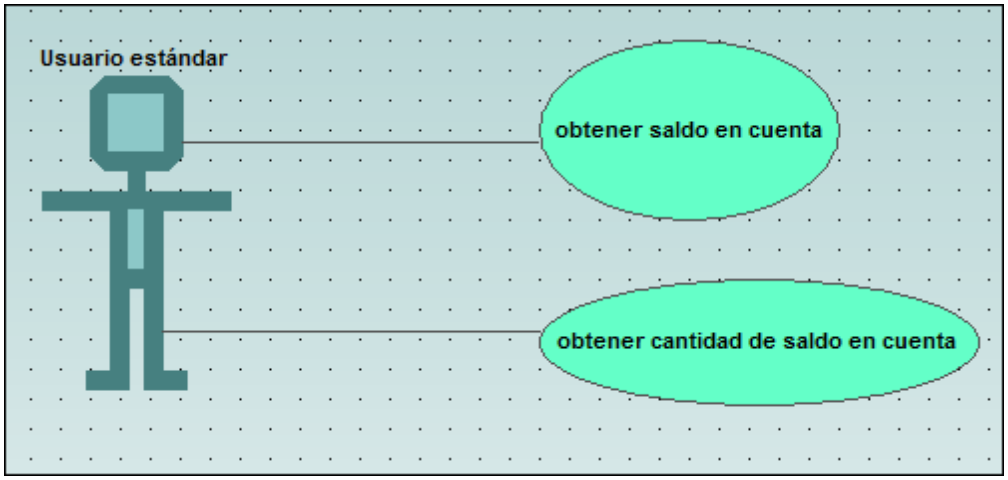


 © 2018 Altova GmbH

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

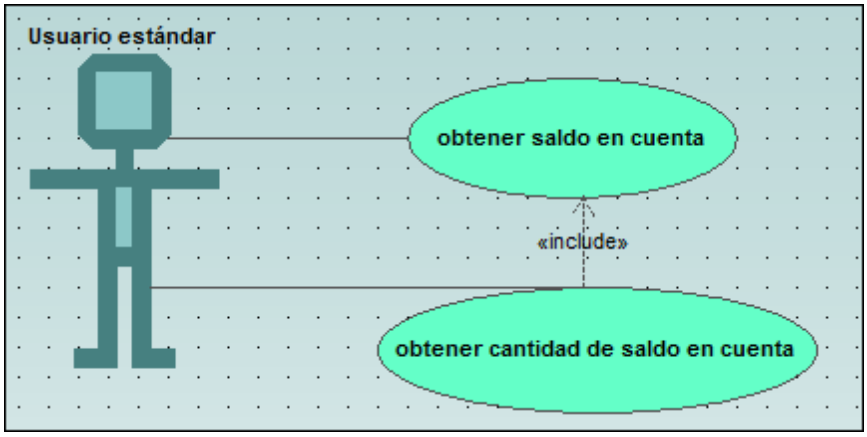
Terminar el diagrama de casos de uso

1. Haga clic en el icono **Caso de uso** de la barra de herramientas mientras pulsa la tecla **Ctrl**.
2. Haga clic en dos posiciones verticales distintas de la ventana del diagrama para añadir dos casos de uso más. Cuando termine puede soltar la tecla **Ctrl**.
3. Al primer caso de uso lo llamamos **`obtener cantidad de saldo en cuenta`** y al segundo **`generar informe mensual de ingresos`**.
4. Haga clic en el icono contraer de ambos casos de uso para ocultar el compartimento de los puntos de extensión.
5. 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)

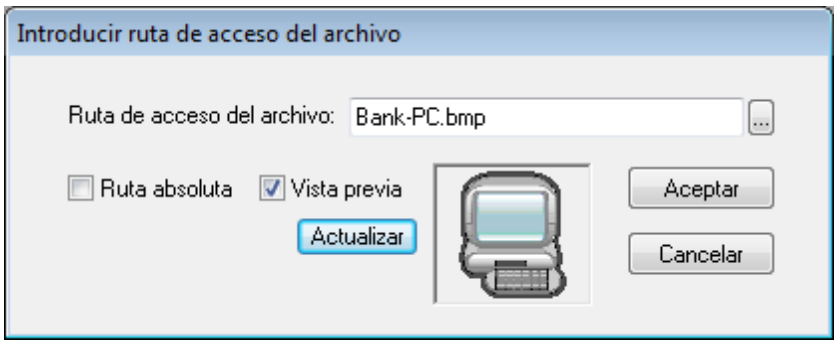
1. Haga clic en el controlador *Inclusión* del caso de uso **`obtener cantidad saldo en cuenta`**, situado en el borde inferior, y arrástrelo hasta el caso de uso **`obtener saldo en cuenta`**. Se crea la dependencia de inclusión y el estereotipo `include` aparece en la flecha de línea discontinua.



Insertar actores definidos por el usuario

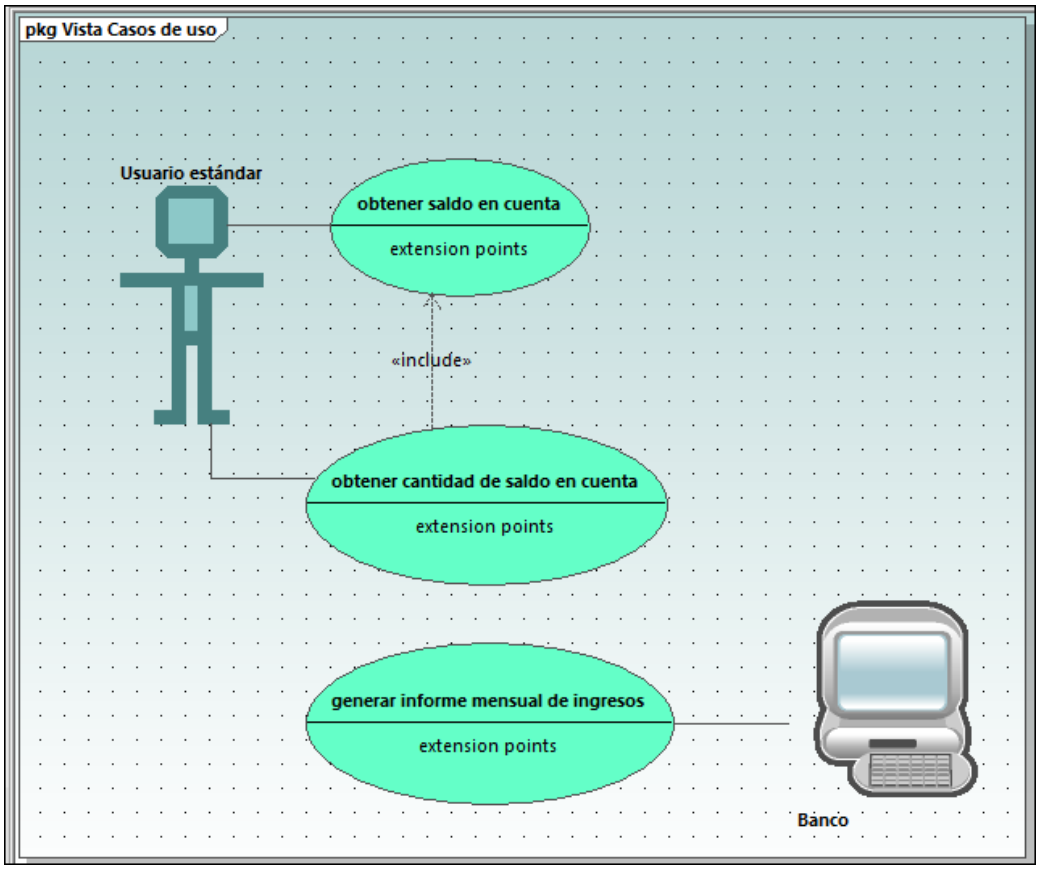
El actor del caso de uso **`generar informe mensual de ingresos`** no es una persona, sino un trabajo automatizado de procesamiento por lotes que ejecuta una computadora del banco. A continuación explicamos cómo añadir un actor nuevo al diagrama y cómo usar una imagen personal para el nuevo actor.

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** situado 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.

 © 2018 Altova GmbH



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

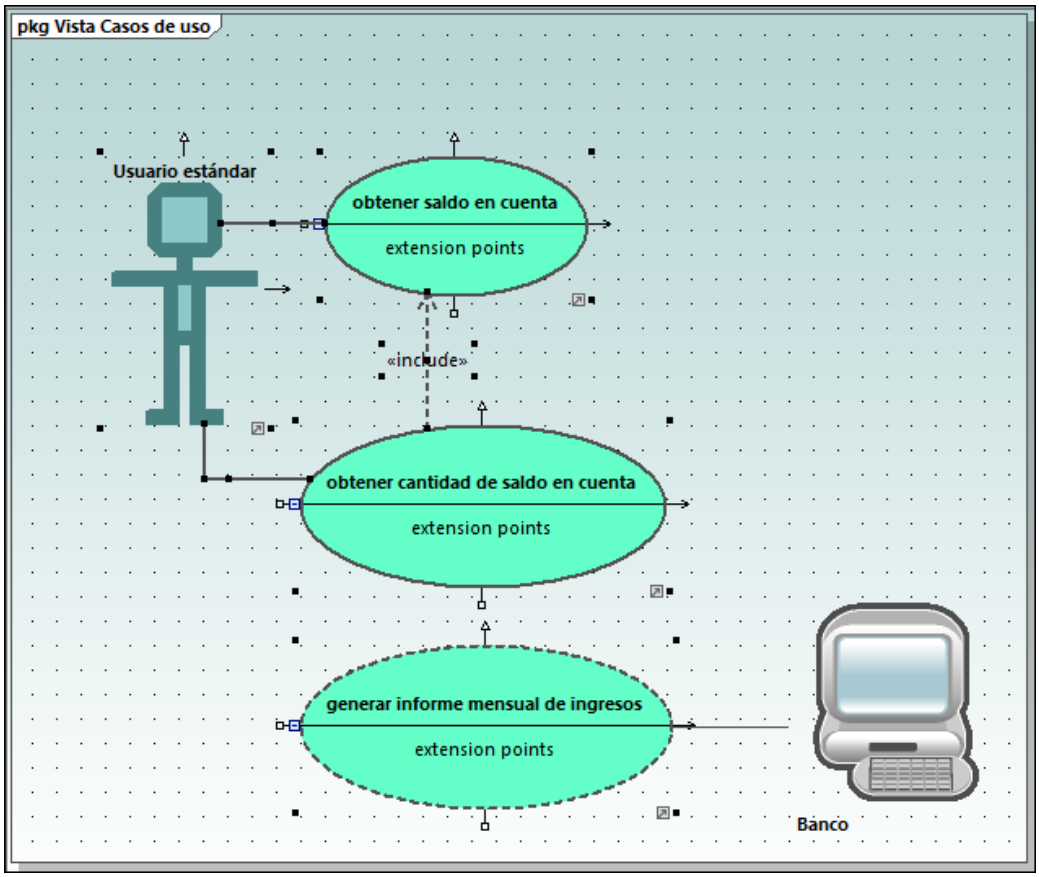
Alinear los elementos del diagrama y ajustar su tamaño

Cuando se arrastran los componentes del diagrama, aparecen líneas de 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 se pueden alinear varios elementos a la vez y ajustar su tamaño de una sola vez:

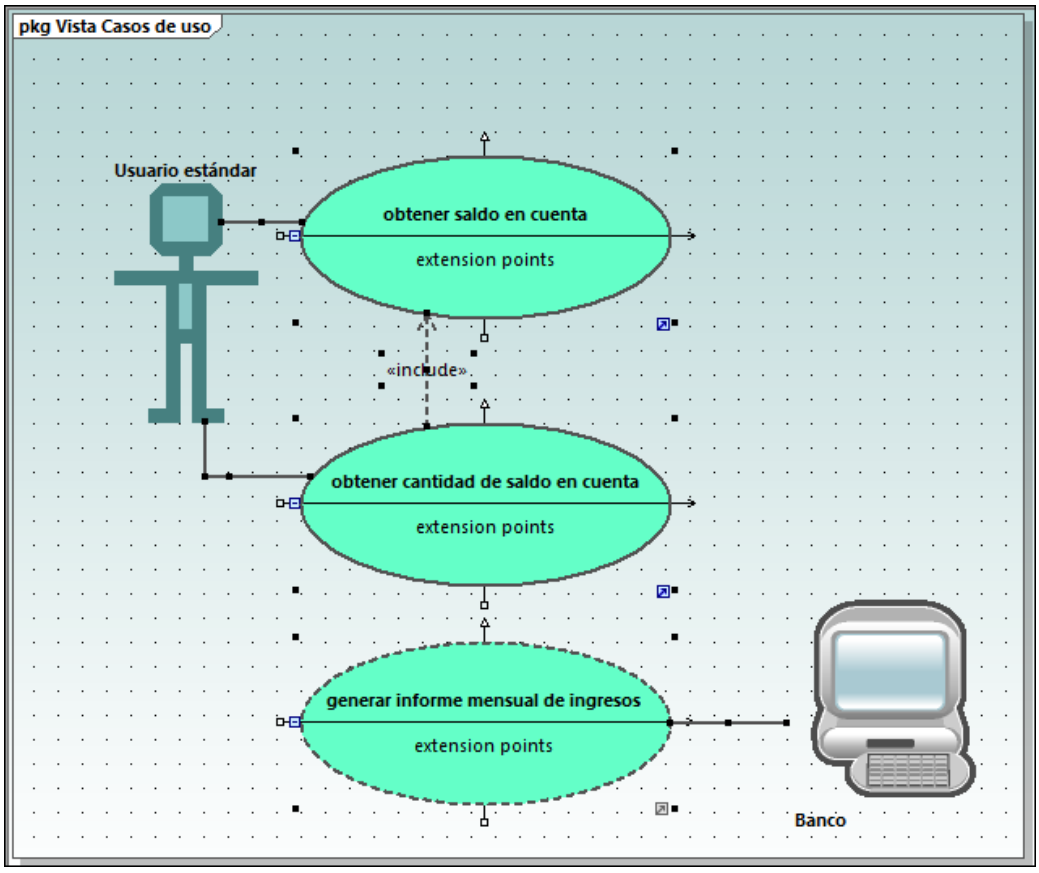
1. Seleccione los tres casos de uso del diagrama (arrastrando el puntero por el diseño para crear un recuadro de selección que abarque los tres casos de uso). También puede seleccionar varios elementos haciendo clic en ellos mientras mantiene pulsada la tecla **Ctrl**. Observe que el último caso de uso 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.

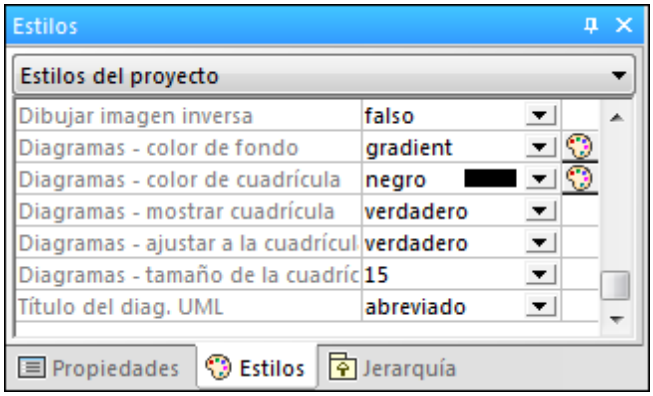
 © 2018 Altova GmbH



Cambiar el estilo de los diagramas en un proyecto

Por defecto todos los diagramas del proyecto de tutorial tienen un color de fondo degradado y una cuadrícula de fondo. 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.

© 2018 Altova GmbH

3.3 Diagramas de clases

Este apartado del tutorial explica:

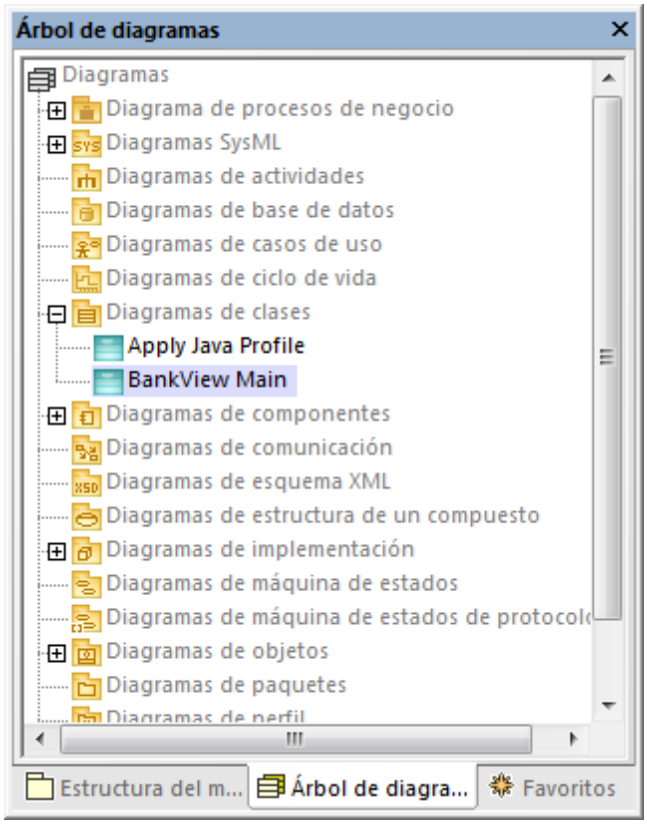
- cómo añadir una clase abstracta a un diagrama de clases,
- cómo añadir propiedades de clase y operaciones y definir parámetros, su dirección y su tipo,
- cómo añadir un tipo devuelto a una operación,
- cómo cambiar iconos por símbolos UML estándar,
- cómo eliminar y ocultar propiedades de clase y operaciones 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 debe 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.

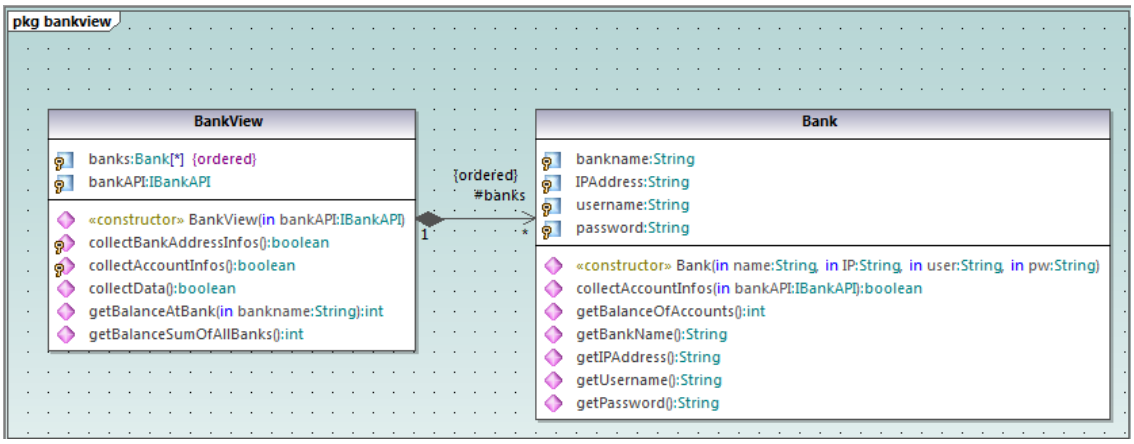
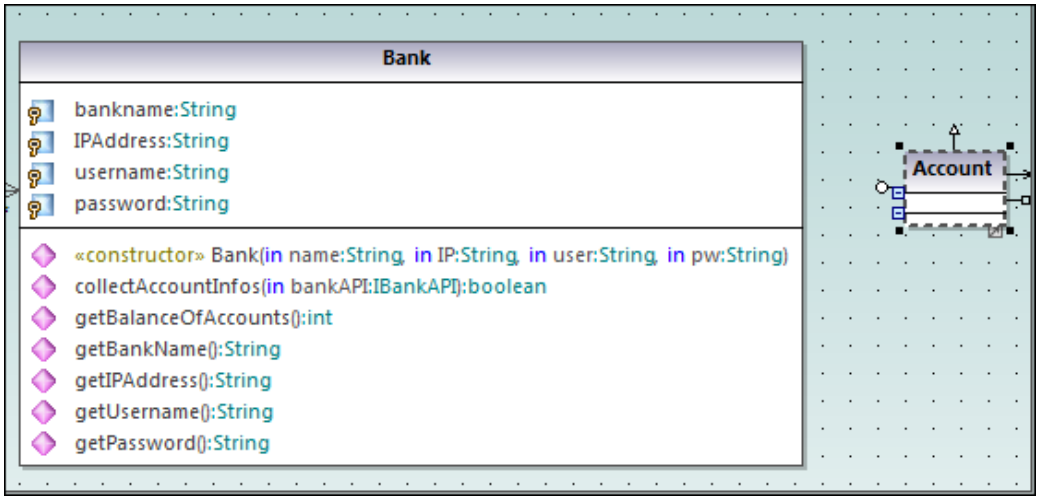


Diagrama "BankView Main"

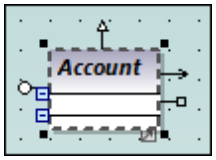
Para añadir la nueva clase abstracta:

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`.

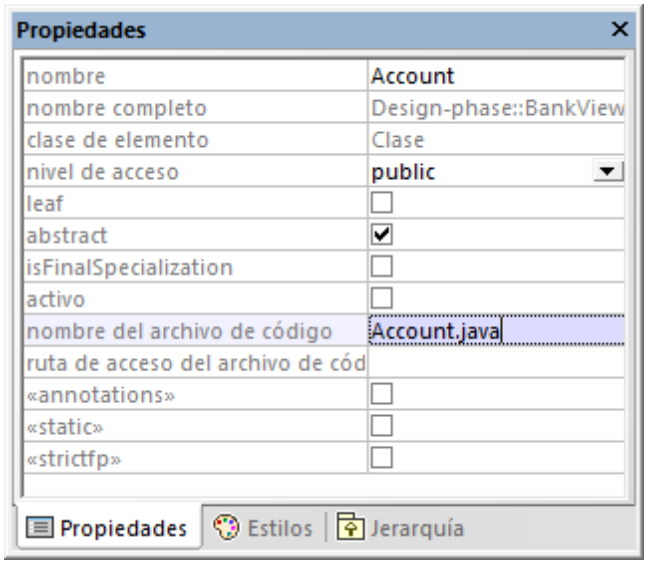


 © 2018 Altova GmbH

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.

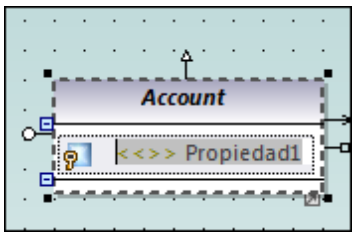


4. En la casilla *nombre del archivo de código* introduzca "Account.java" para definir la clase Java.

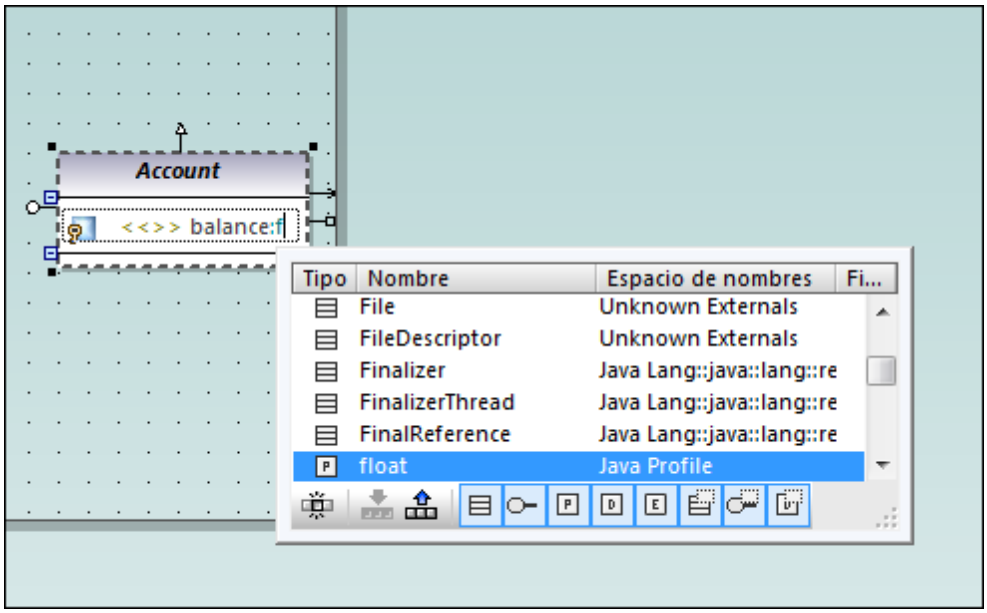


Para agregar propiedades a una clase:

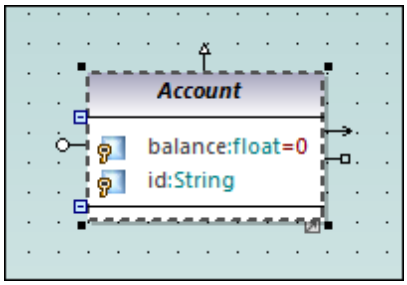
1. Haga clic con el botón derecho en la clase "Account" y seleccione **Nuevo/a | Propiedad** o pulse **F7**. Esto inserta una propiedad predeterminada llamada `Property1` con los identificadores de estereotipo `<< >>`.



2. Cambie el nombre de la propiedad por `balance` y después introduzca un carácter de dos puntos (:). Ahora 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.

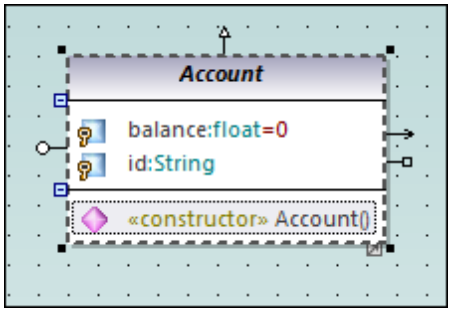


4. Siga en la misma línea y anexe "=0" para definir el valor predeterminado.
5. Para terminar (y usando el mismo método) cree una propiedad nueva llamada `id` de tipo `String`.



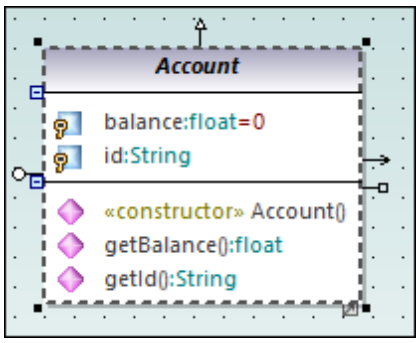
Para agregar operaciones a una clase:

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.



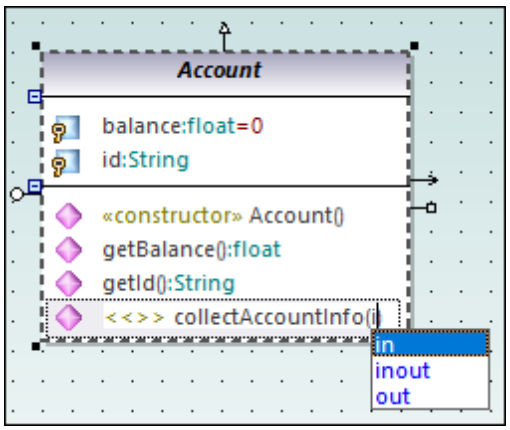
 © 2018 Altova GmbH

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

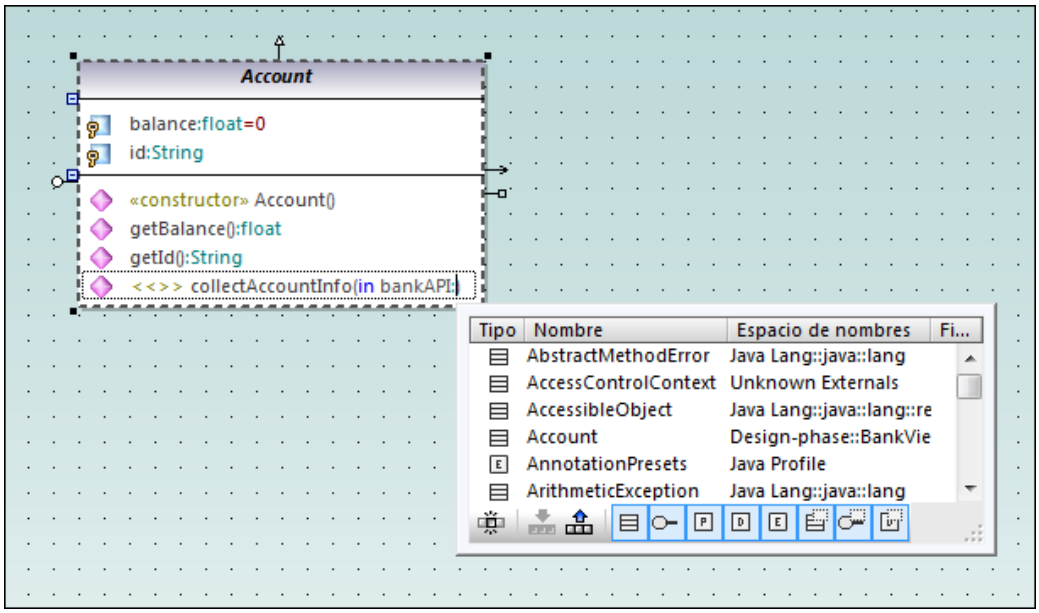


Ahora añadiremos una operación nueva que toma un parámetro. Además especificaremos la dirección y el tipo de este parámetro.

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 un carácter de dos puntos (:). Aparece una lista desplegable donde puede seleccionar el tipo del parámetro.

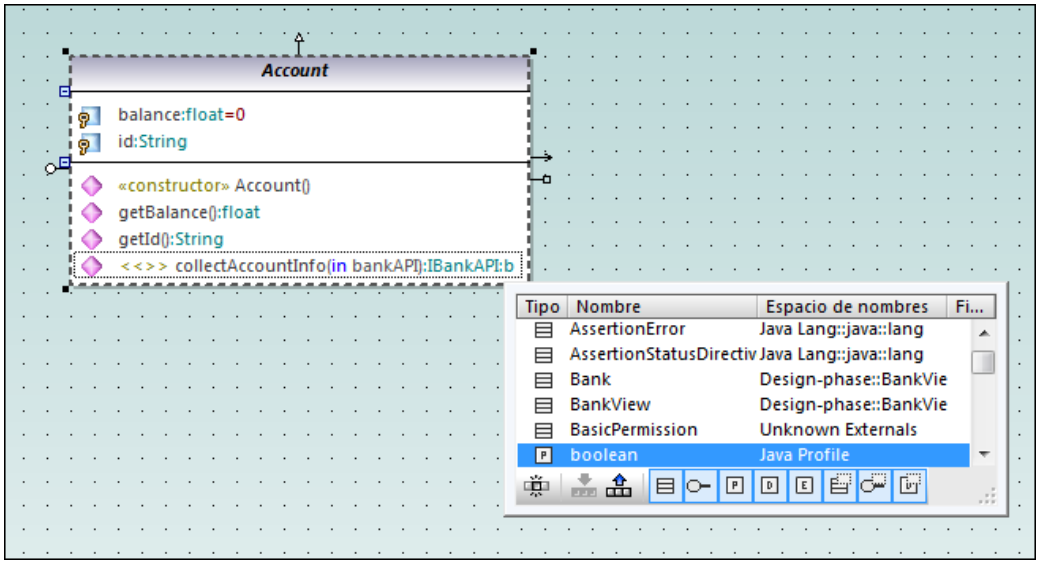


5. Seleccione **IBankAPI** en la lista desplegable.

Para agregar un tipo devuelto a una operación:

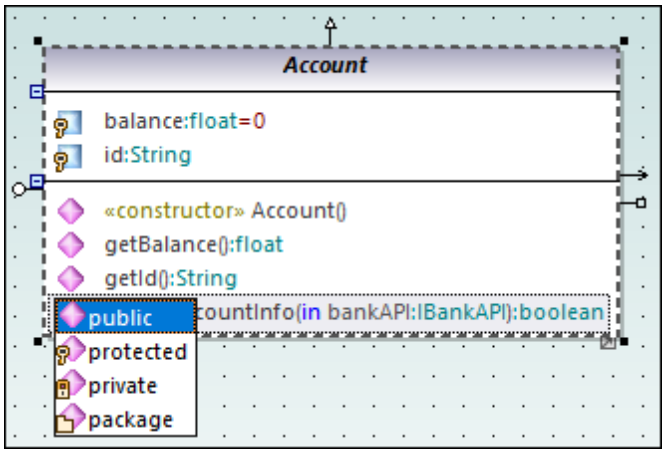
Llegados a este punto se añadió el parámetro de la operación pero todavía no tiene un tipo devuelto.

1. Ponga el cursor del ratón detrás del paréntesis de cierre ")" e introduzca un carácter de 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:

 © 2018 Altova GmbH



Para cambiar los iconos por símbolos UML estándar:

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

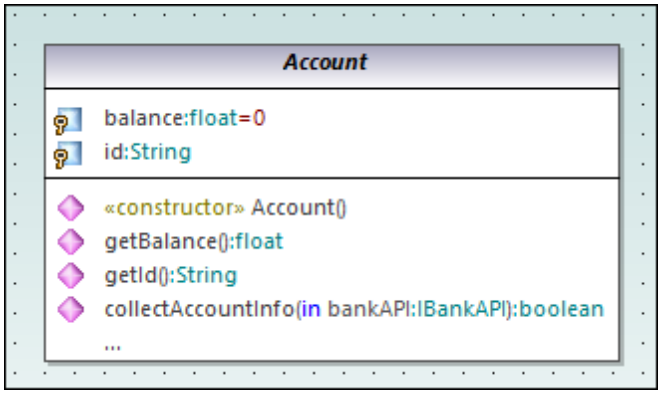
1. En la ventana *Estilos* seleccione **Estilos del proyecto** en la lista desplegable.
2. Desplácese hasta la opción *Mostrar nivel de acceso* y seleccione **Estilo de UML**.

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`.

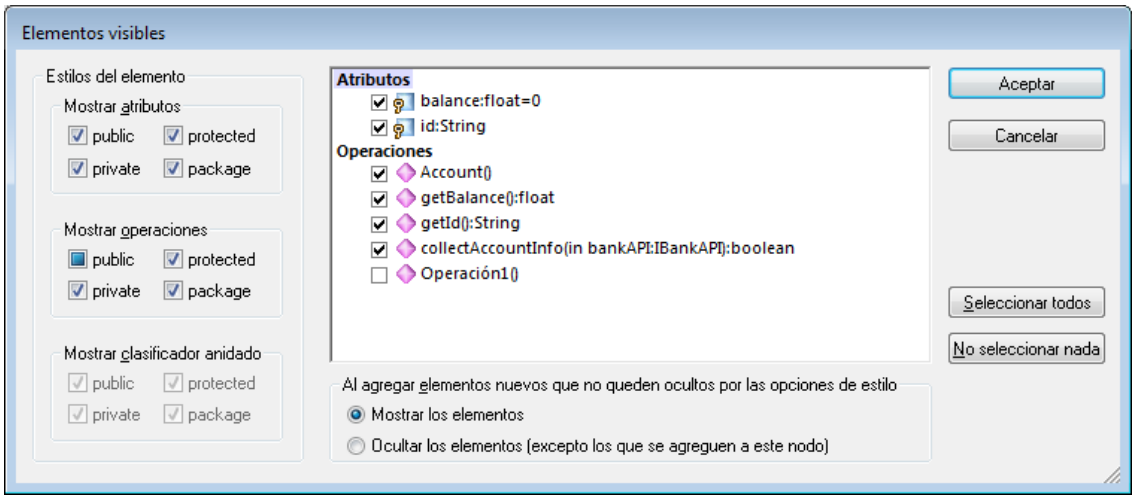
Para eliminar esta operación de ejemplo basta con seleccionarla y pulsar **Supr**. También puede hacer clic con el botón derecho en la operación y seleccionar **Eliminar** en el menú contextual. Aparecerá un cuadro de mensaje preguntando si desea eliminar el elemento del proyecto. Haga clic en **Sí**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

Para eliminar una operación de una clase pero no del proyecto pulse **Ctrl+Supr**. Esto oculta la operación del diagrama pero la operación 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"

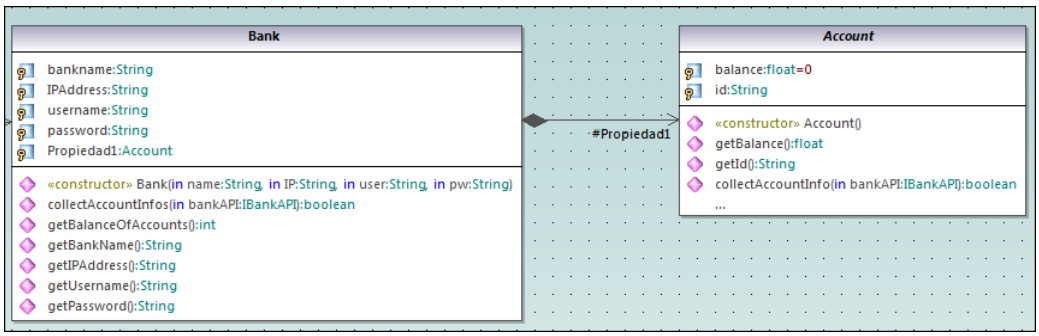
Si lo prefiere, puede configurar UModel para que no muestre un cuadro de mensaje cada vez que se intenta eliminar un objeto del diagrama:

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*.

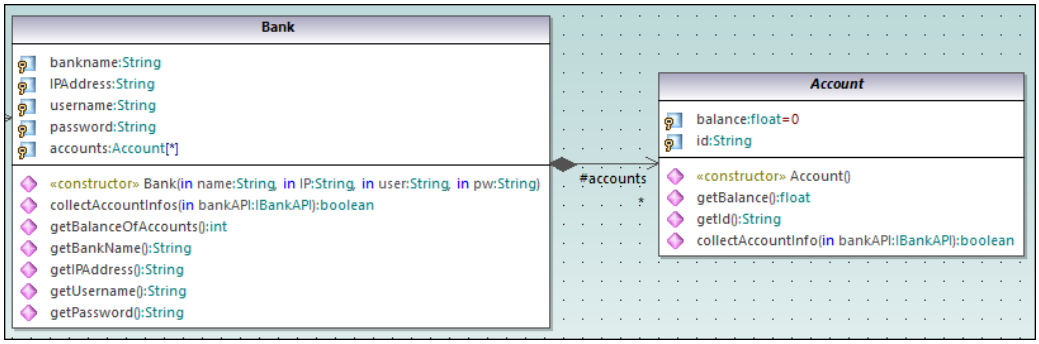
Para crear una asociación de composición entre las clases Bank y Account:

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

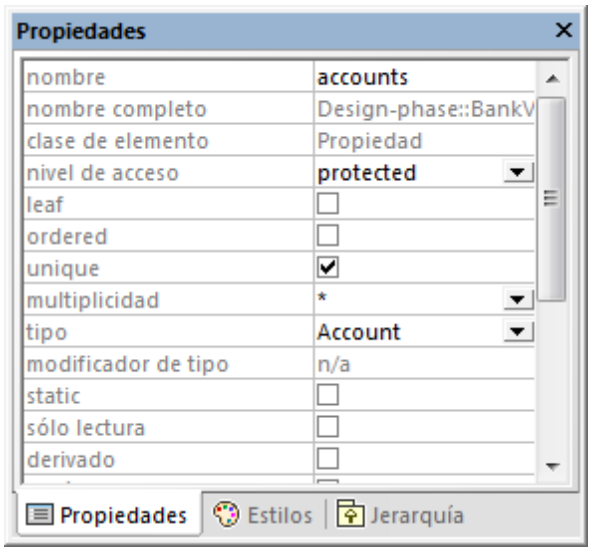
asociación se puede crear. Como resultado se crea una propiedad nueva llamada `Propiedad1:Account` en la clase `Bank` y una flecha de asociación compuesta une las dos clases.



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*:



3.3.1 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`.

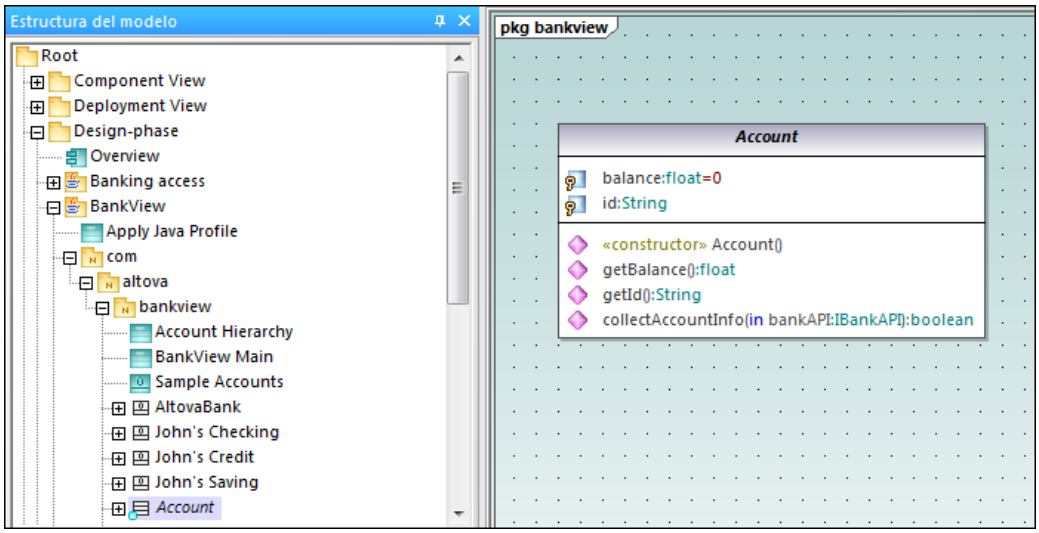
Crear un diagrama de clases nuevo

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.

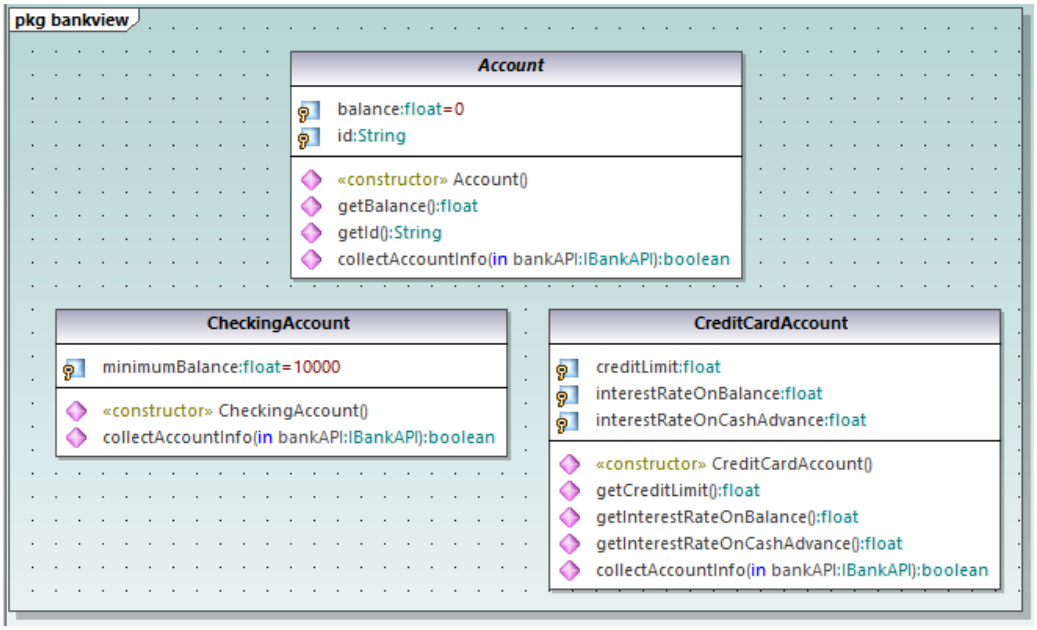
Agregar clases que ya existen a un diagrama

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.

 © 2018 Altova GmbH



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`.

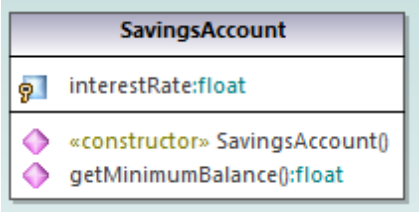


Agregar una clase nueva

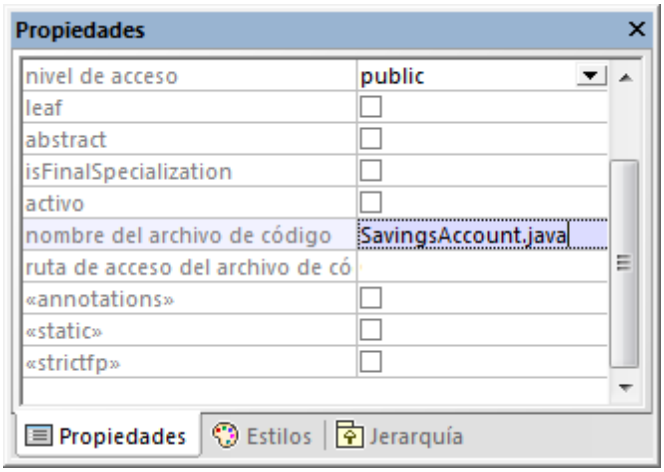
La tercera clase derivada (`SavingsAccount`) se añadirá manualmente al diagrama.

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:

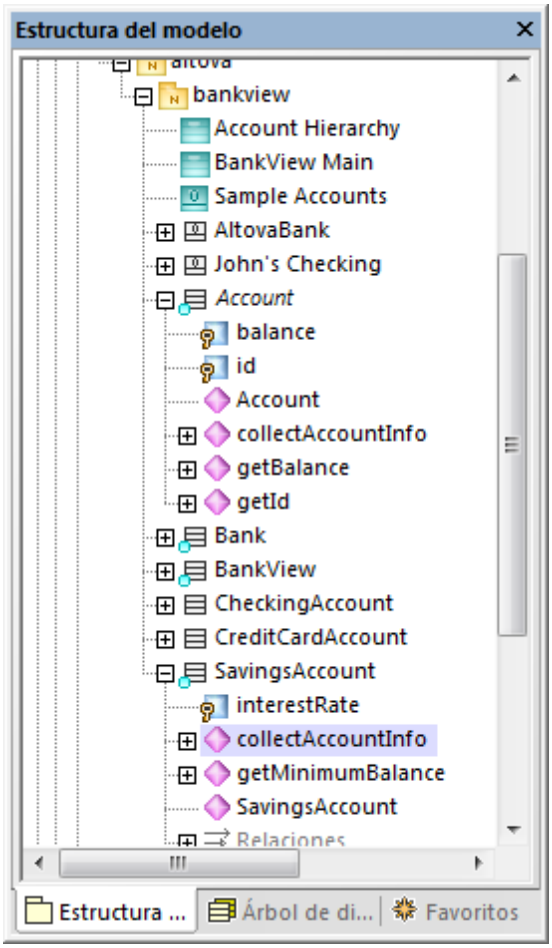
- propiedades y operaciones de una clase del diagrama actual,
- propiedades y operaciones de clases distintas del mismo diagrama,
- propiedades y operaciones de la ventana *Estructura del modelo*,
- propiedades y operaciones de diagramas UML distintos (colocando los datos copiados en un diagrama diferente).

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.

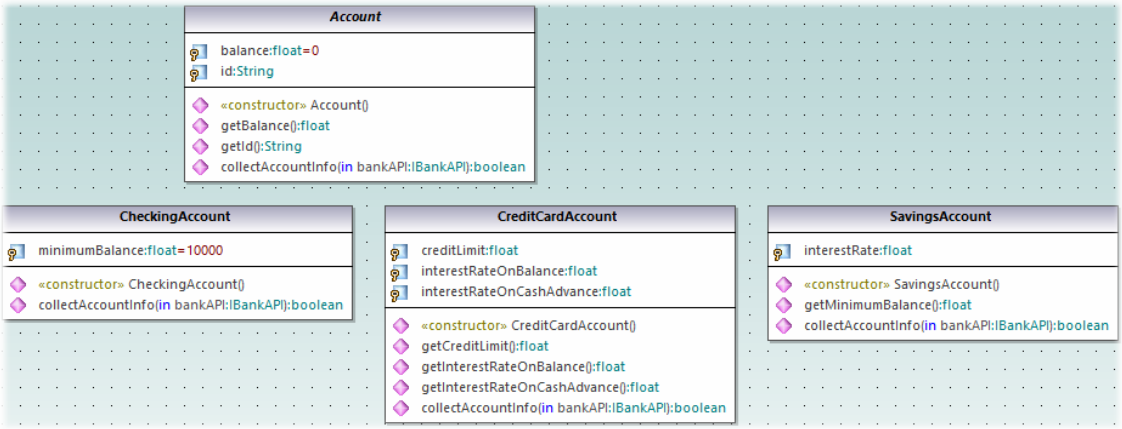
 © 2018 Altova GmbH



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

Crear clases derivadas mediante generalización/especialización

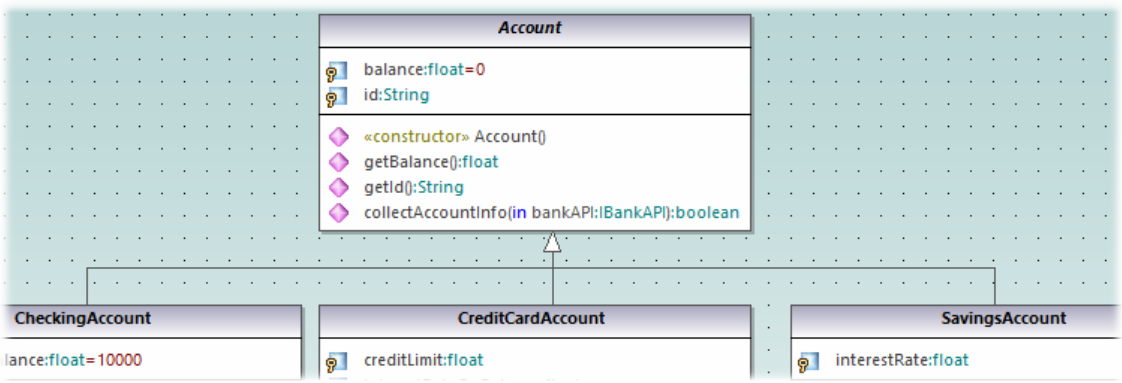
Llegados a este punto el diagrama de clases contiene la clase abstracta `Account` y tres clases específicas.



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`.



 © 2018 Altova GmbH

3.4 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".

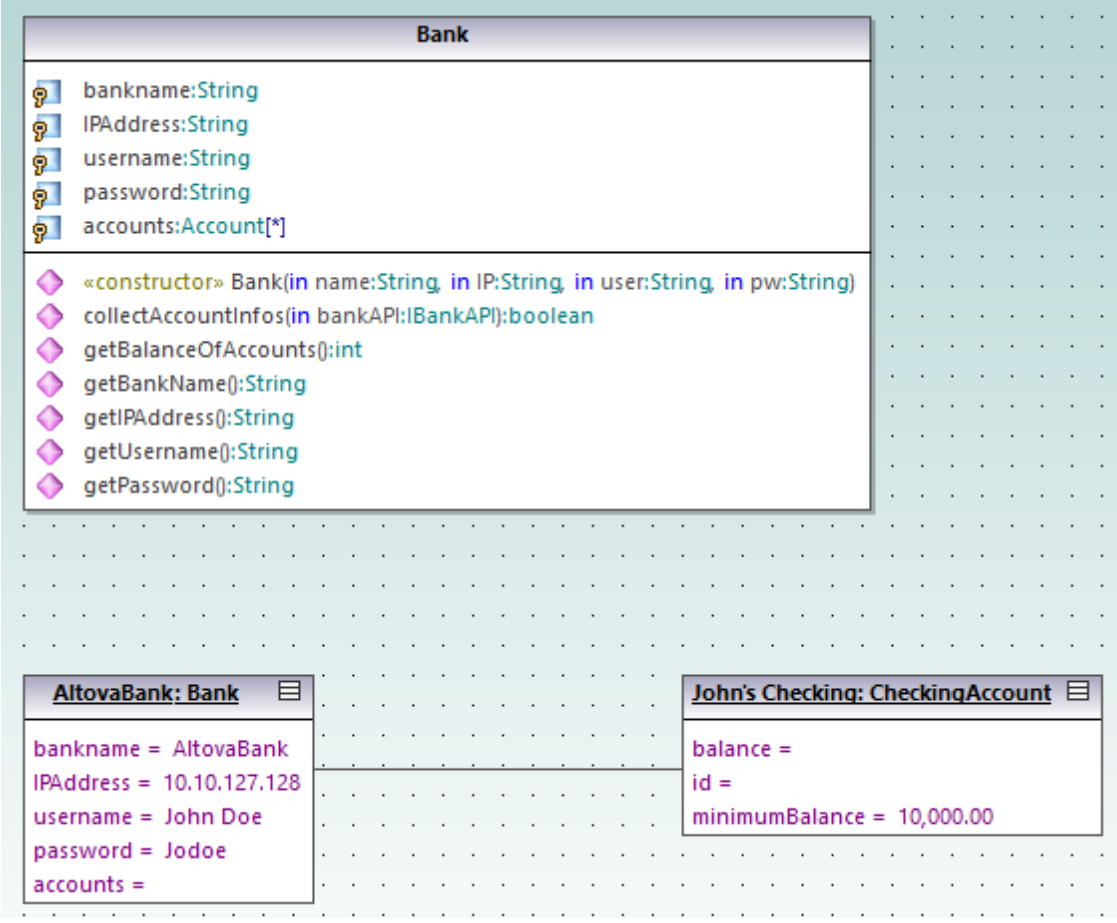
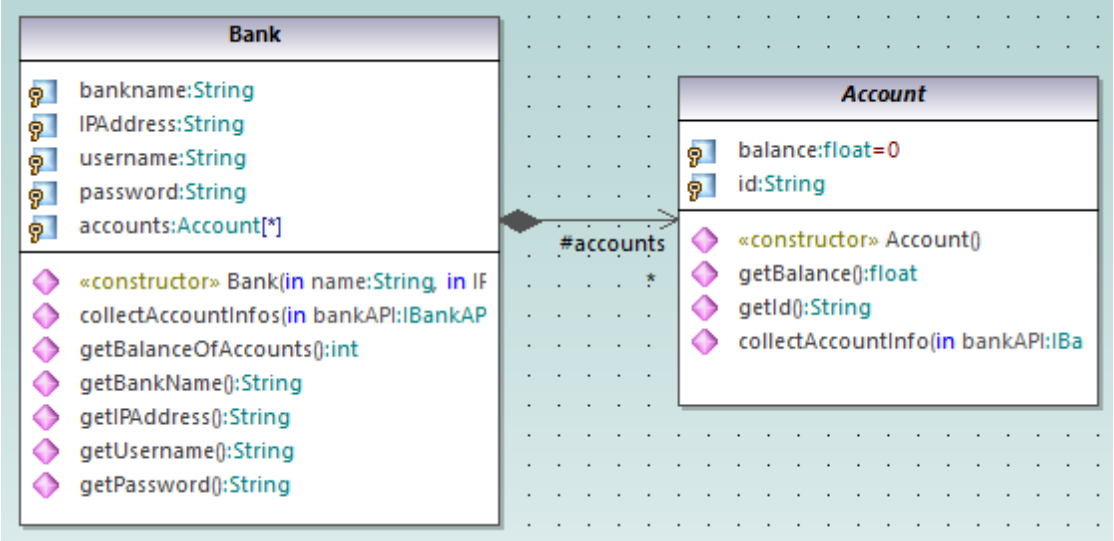


Diagrama "Sample Accounts"

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).

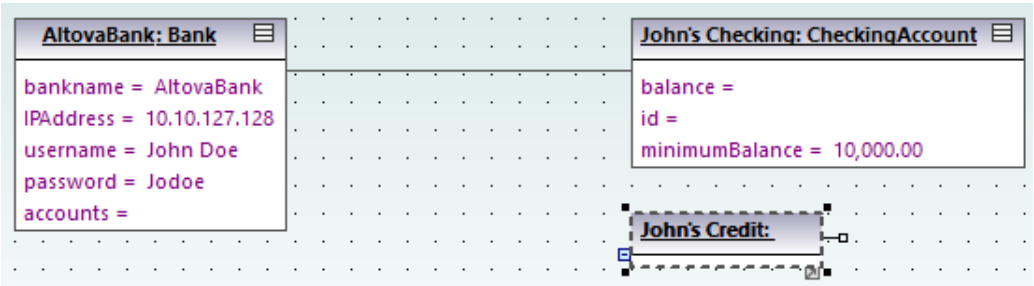
Ahora vamos a agregar la clase `Account` que falta. Para ello la seleccionamos y la arrastramos desde la *Estructura del modelo* hasta el diagrama. Observe que aparece automáticamente la asociación compuesta entre `Bank` y `Account` (esta asociación se definió en una de las fases anteriores del tutorial,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



Agregar un objeto/una instancia nuevos (Método 1)

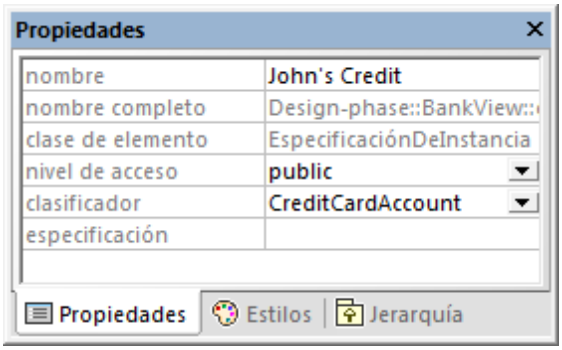
Ahora vamos a agregar un objeto nuevo llamado `John's Credit` al diagrama. Este objeto creará una instancia de la clase `CreditCardAccount`.

1. Haga clic en el botón **EspecificaciónDeInstancia** de la barra de herramientas y después haga 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**.

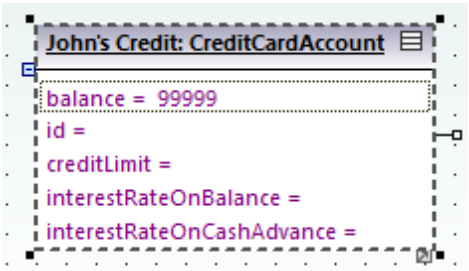


3. Seleccione la instancia nueva para ver sus propiedades en la ventana *Propiedades*.
4. En la ventana *Propiedades*, junto al campo `clasificador`, seleccione

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:

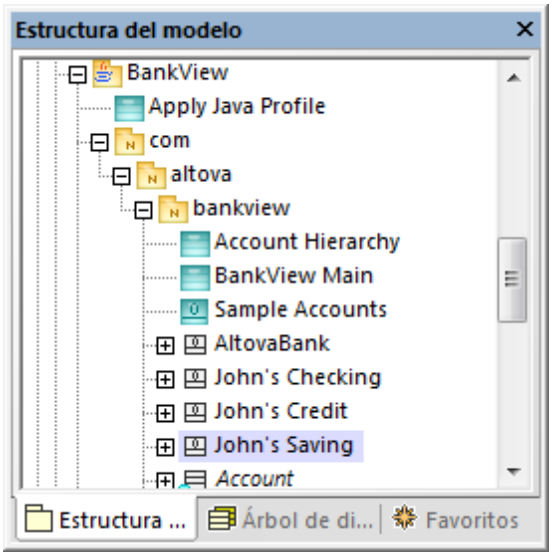


Para mostrar u ocultar nodos concretos haga clic con el botón derecho en la instancia y seleccione **Mostrar u ocultar el contenido del nodo** (**Ctrl+Mayús+H)** del menú contextual.

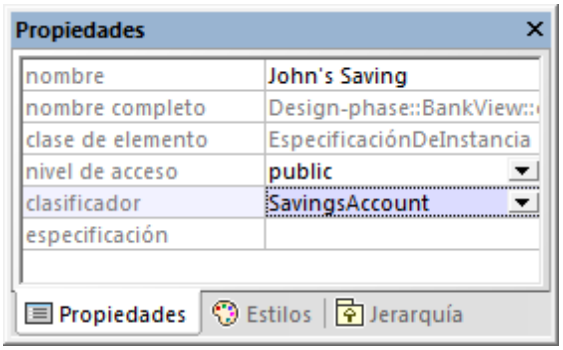
Agregar un objeto/una instancia nuevos (Método 2)

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

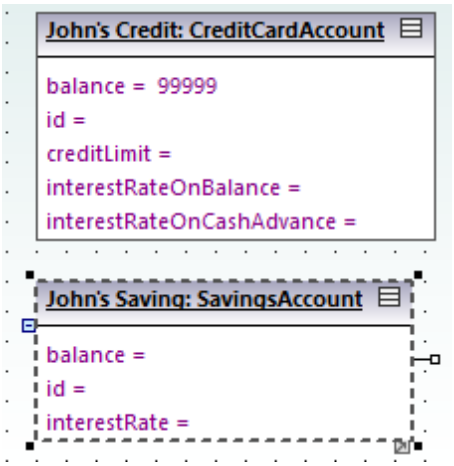
1. En la ventana *Estructura del modelo* haga clic con el botón derecho en el paquete `bankview` y seleccione **Elemento nuevo | EspecificaciónDeInstancia**.
2. Cambie el nombre de la instancia nueva por `John's Saving` y pulse **Entrar** para confirmar. El objeto nuevo se añade al paquete y se coloca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



3. Estando el objeto todavía seleccionado en la ventana *Estructura del modelo*, ahora seleccione **SavingsAccount** 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`.

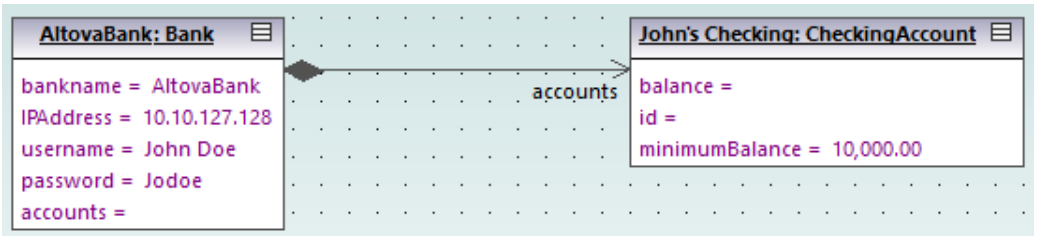


 © 2018 Altova GmbH

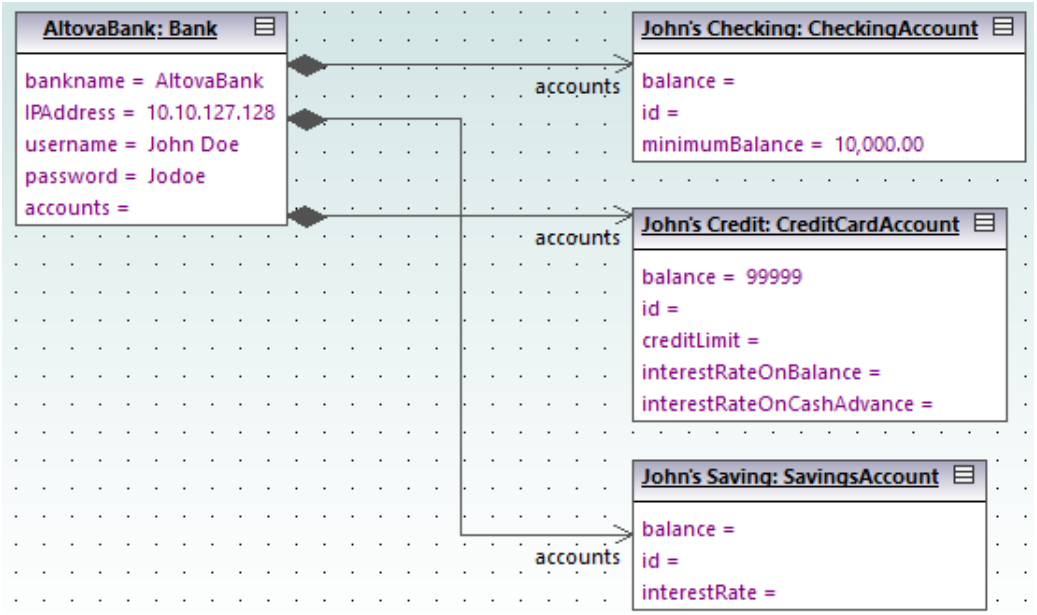
Crear vínculos entre objetos

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.



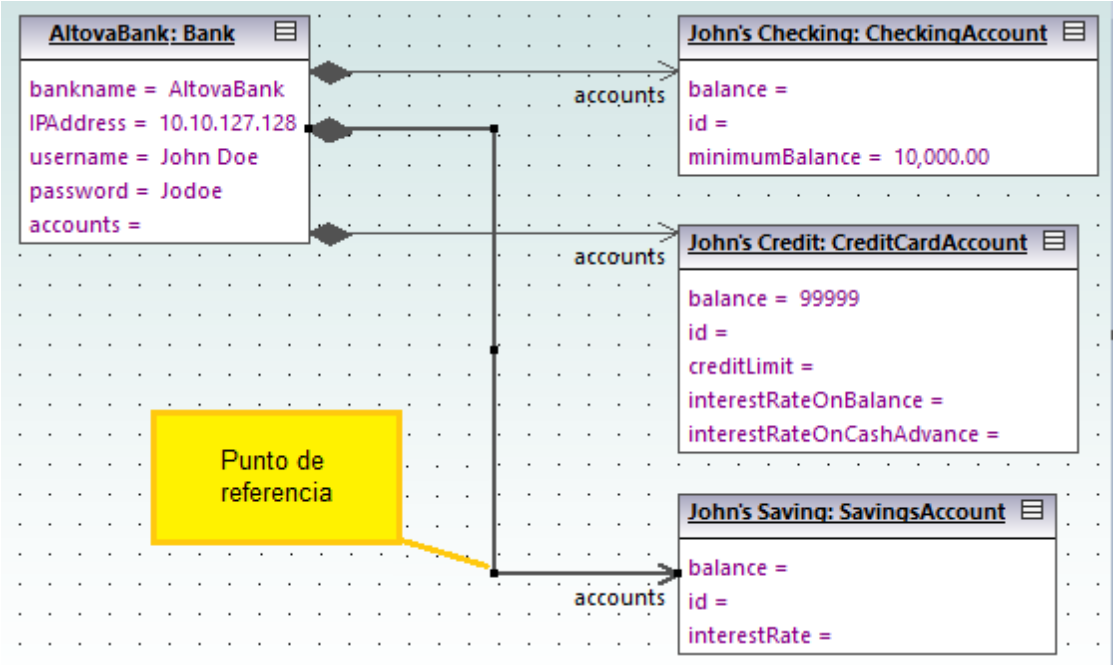
3. Haga clic en el botón **EspecificaciónDeInstancia** de la barra de herramientas y pase el puntero por encima del objeto `John's Credit: CreditAccount`. El puntero adopta ahora la forma del signo +.
4. Arrastre el puntero desde el objeto `John's Credit: CreditAccount` hasta `AltovaBank: Bank` para 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 realizados que afectan 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*).

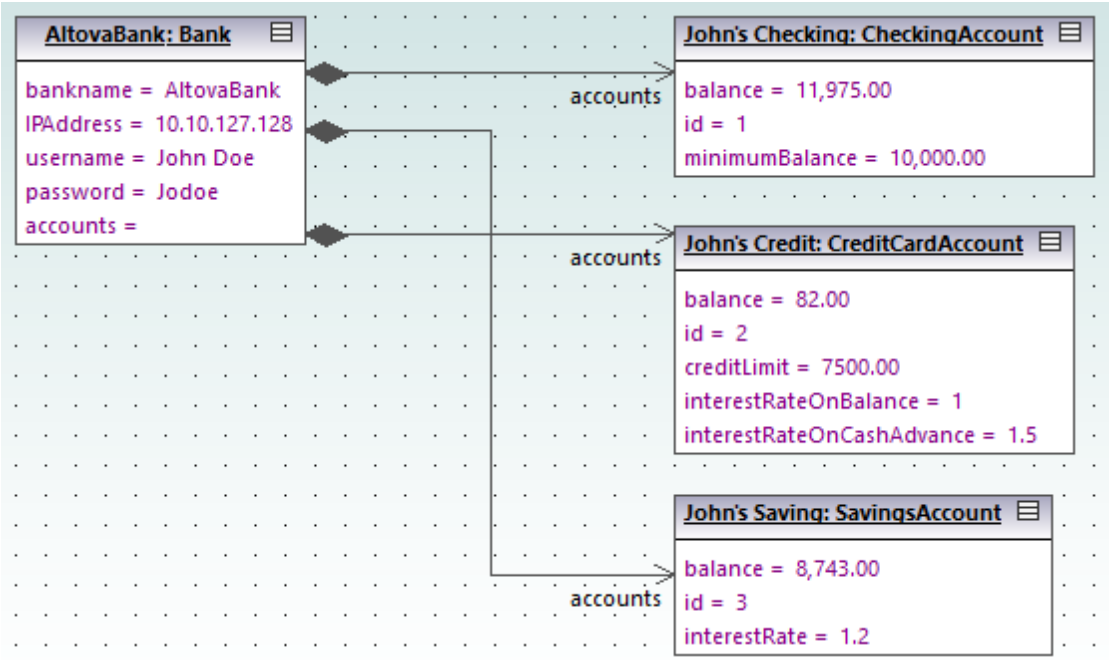


Vínculos en un diagrama de objetos

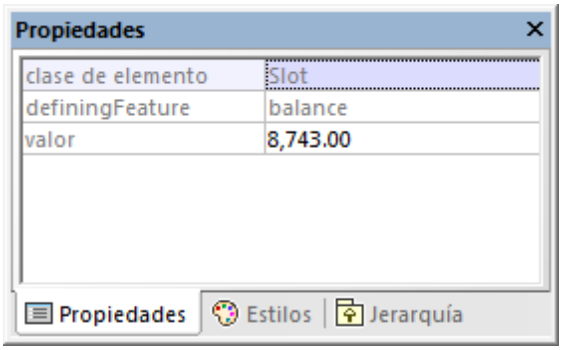
Introducir datos de muestra en objetos

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:

© 2018 Altova GmbH



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*.



3.5 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 esférica.

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.

© 2018 Altova GmbH

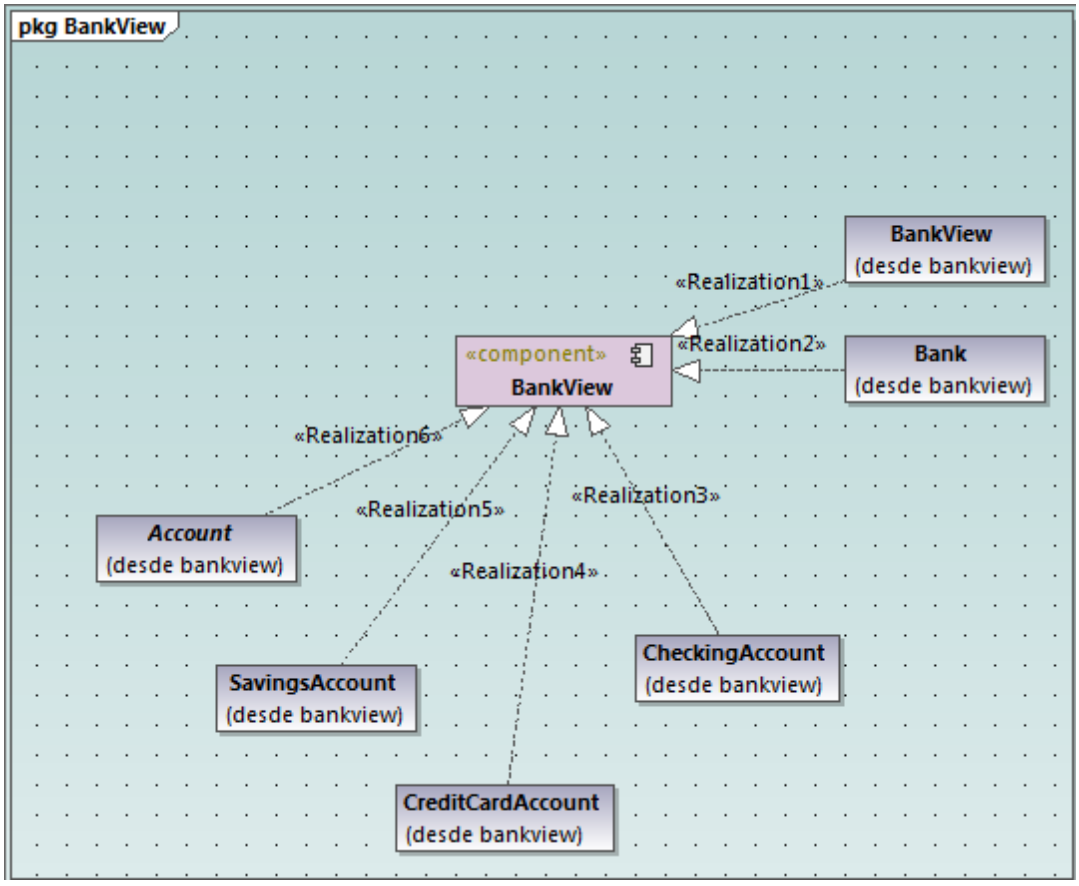
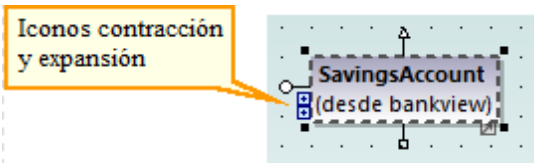


Diagrama "BankView realization"

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

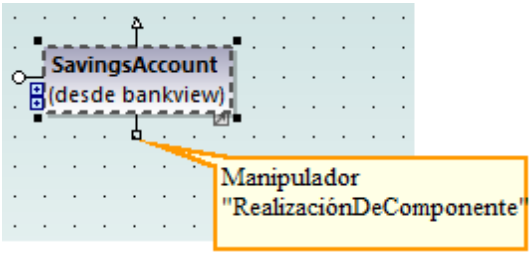
1. En la ventana *Estructura del modelo* navegue hasta la clase `SavingsAccount` del paquete `bankview`. Si falta esta clase, entonces debe completar las instrucciones del apartado Crear clases derivadas del tutorial para crearla.
2. Arrastre la clase `SavingsAccount` desde la ventana *Estructura del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

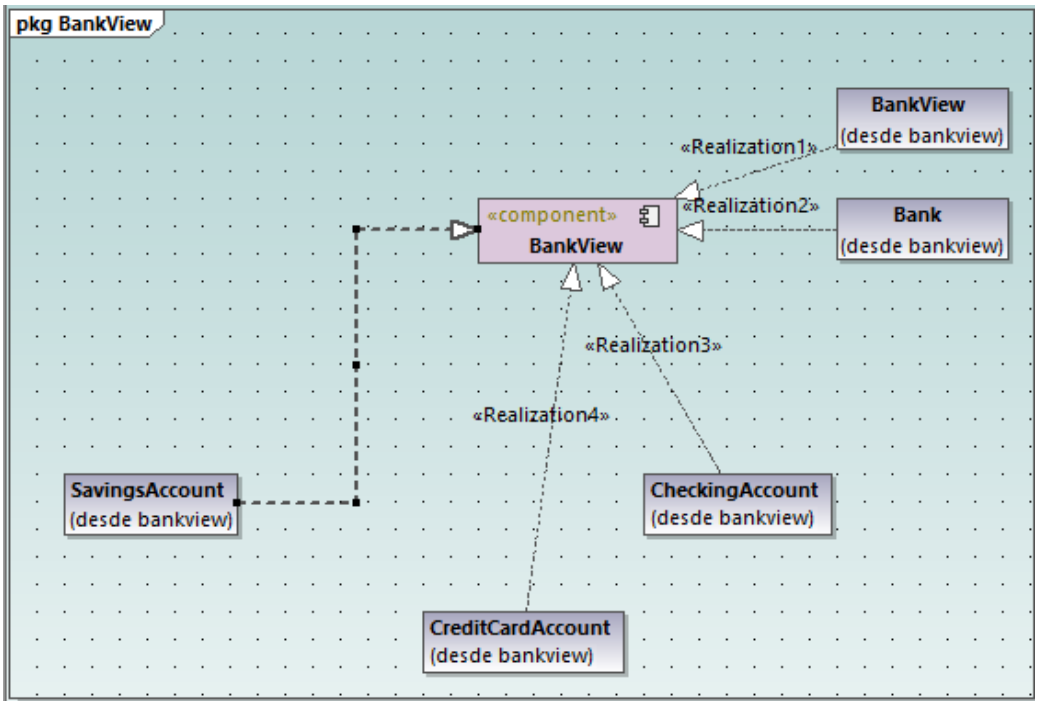


Hay dos maneras de crear una dependencia de realización entre la clase y el componente:

- 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

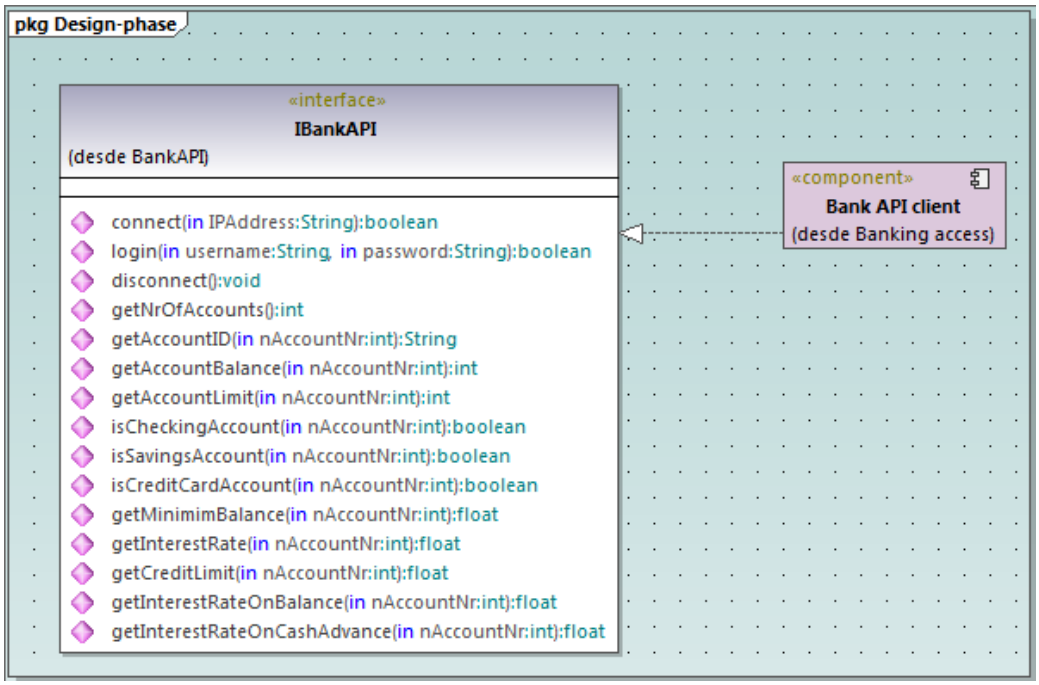
Ahora aprenderemos a cambiar el aspecto de las líneas, que pueden ser curvas o rectas:

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.

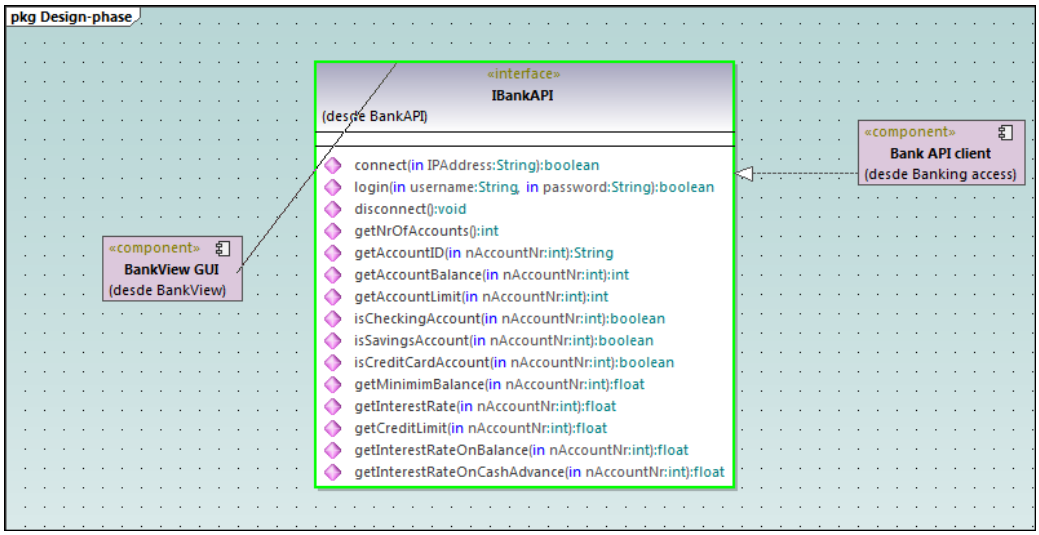
© 2018 Altova GmbH

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.

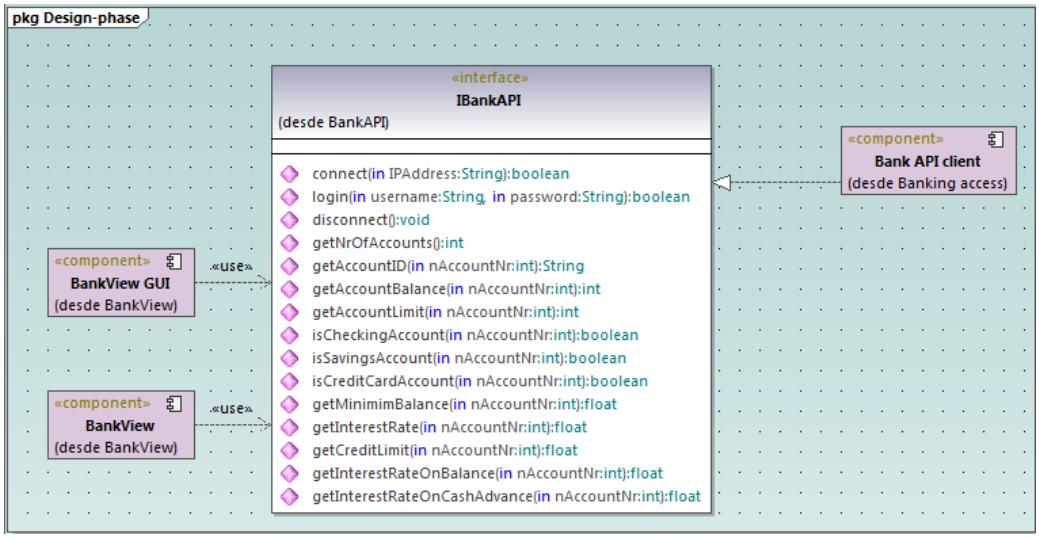


2. En la ventana *Estructura del modelo* navegue hasta **Root | Component View | BankView** y arrastre el paquete `BankView GUI` hasta el diagrama de componentes "Overview".
3. Ahora arrastre también el paquete `BankView` hasta el diagrama de componentes "Overview".
4. Haga clic en el botón **Utilización**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

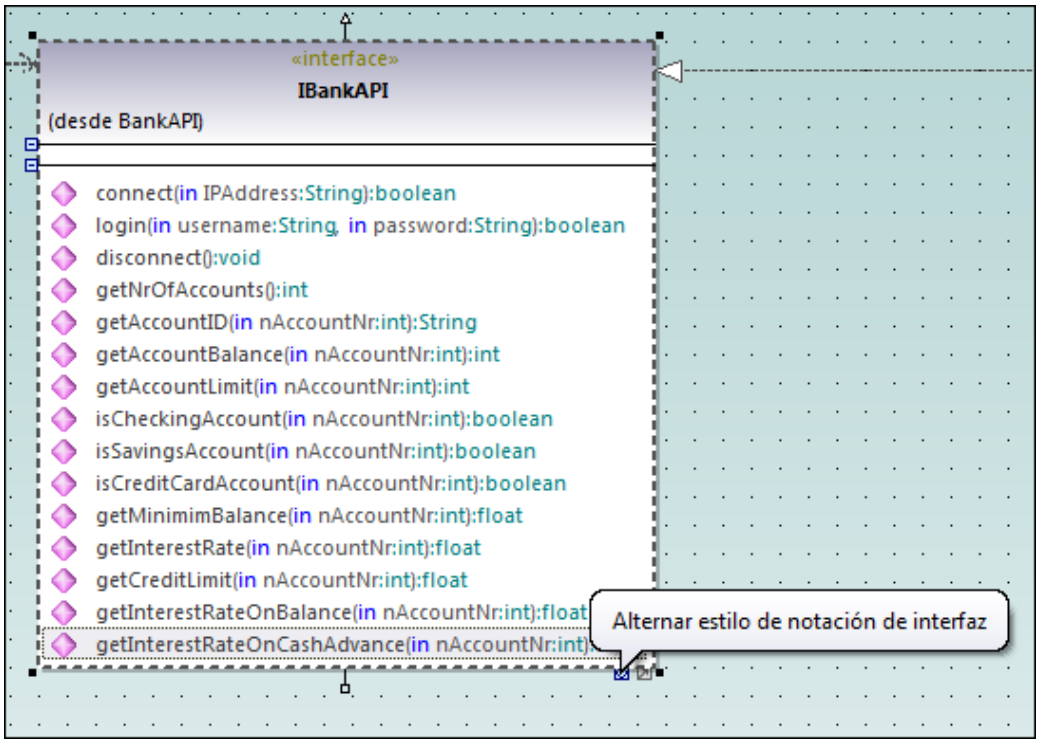
Como puede ver en la imagen siguiente los dos paquetes tienen ahora una dependencia de utilización con la interfaz. Es decir, los paquetes `BankView` y `BankView GUI` necesitan a la interfaz `IBankAPI`. Por su parte, el paquete `Bank API Client` aporta la interfaz.



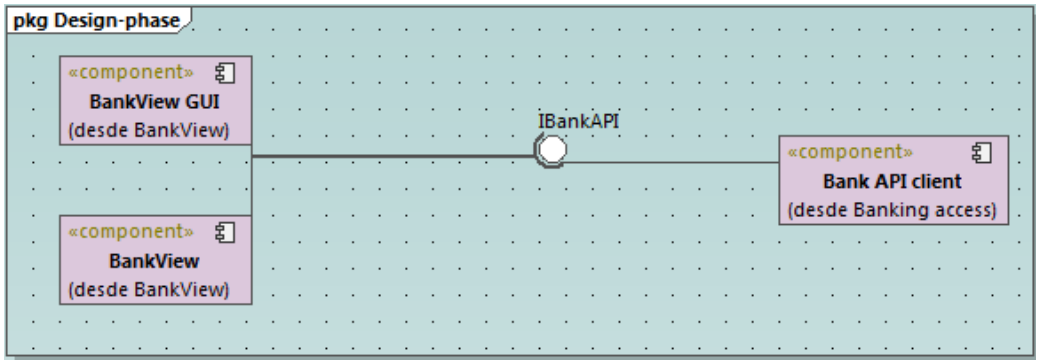
Usar notaciones esféricas

Si lo desea, puede convertir la notación del diagrama en una notación de tipo *esférico*. 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.

© 2018 Altova GmbH



Cuando se hace clic en ese botón, el diagrama se presenta con notación esférica.



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**.

3.6 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 abrir este diagrama. Como puede ver en la imagen siguiente, este diagrama muestra la manifestación de los componentes `Bank API client` y `BankView` a sus archivos Java .jar compilados correspondientes.

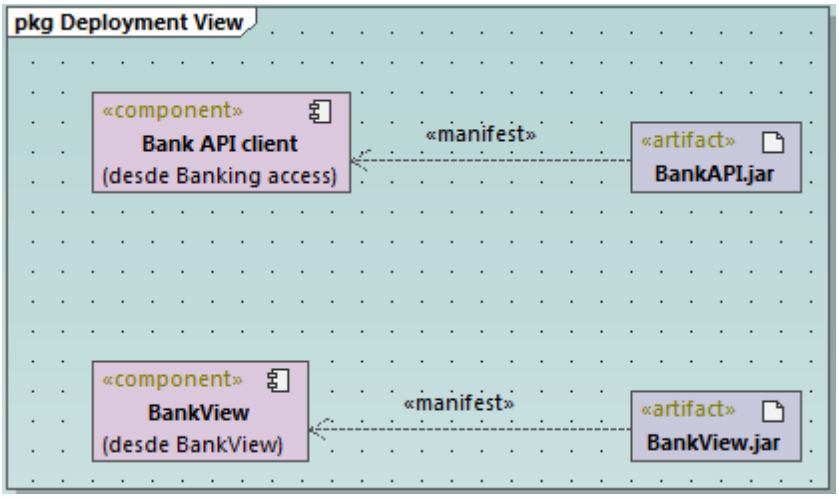


Diagrama "Artifacts"

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

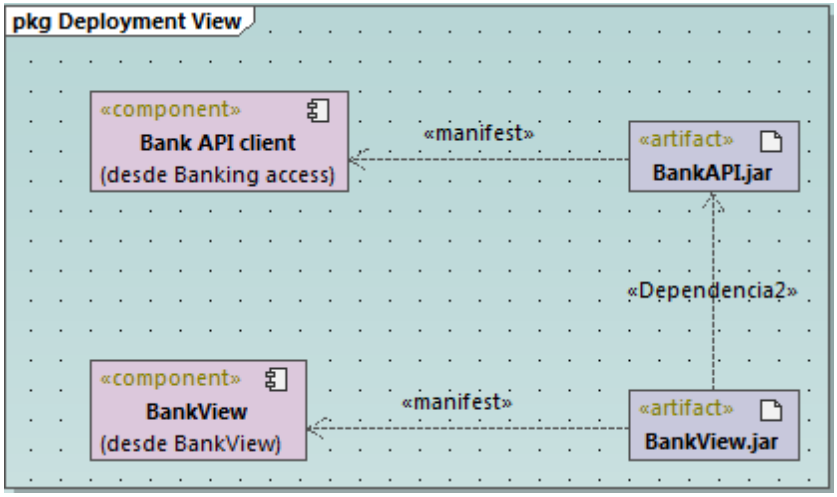
1. Haga clic en el botón Manifestación de la barra de herramientas.
2. Arrastre el puntero desde el artefacto hasta el componente.

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`.

© 2018 Altova GmbH

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.

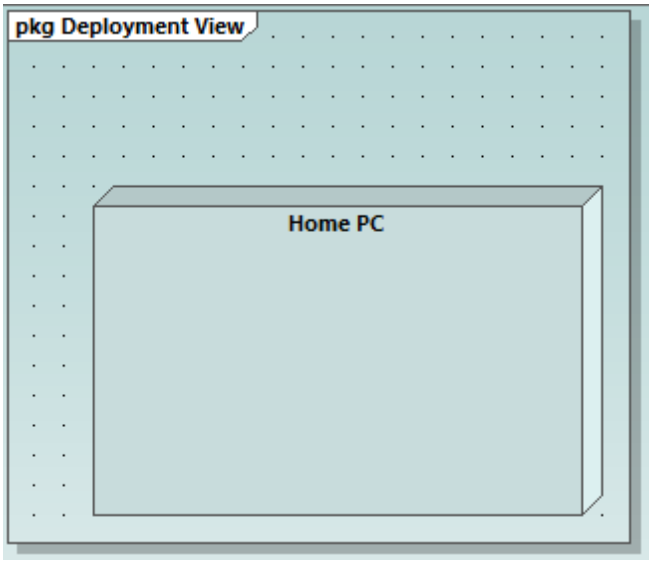
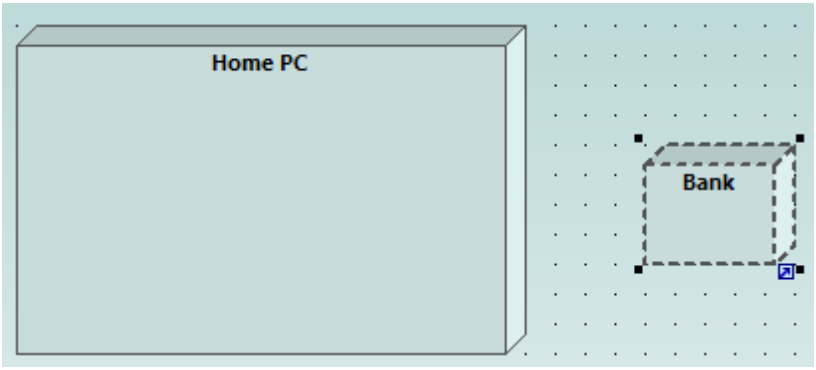


Diagrama "Deployment"

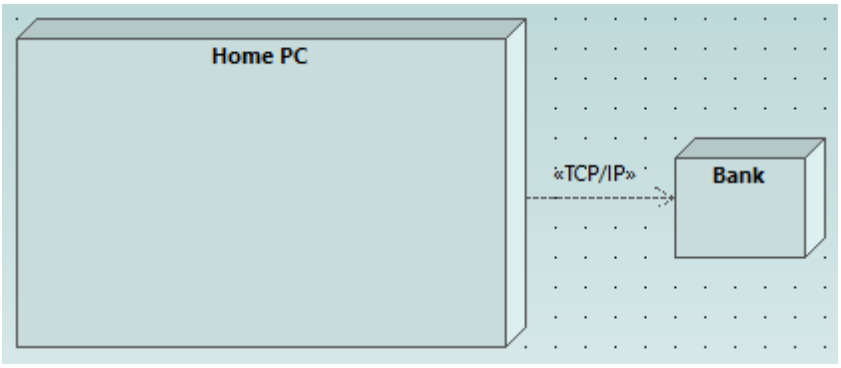
Imaginemos que nuestro objetivo es ilustrar una conexión TCP/IP entre el equipo doméstico y un banco. Para ello deberemos añadir varios elementos que son necesarios:

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.



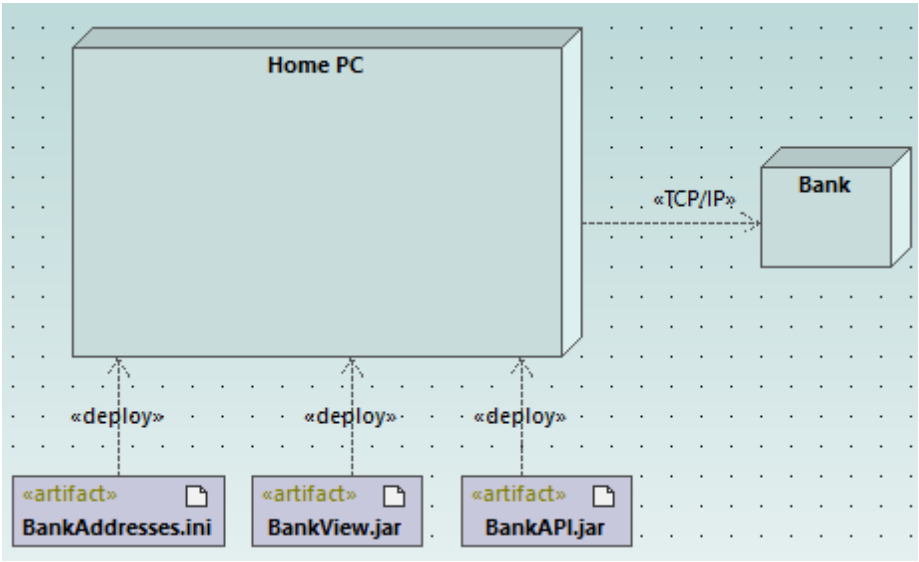
3. Haga clic en el botón **Dependencia** de la barra de herramientas y arrastre el puntero desde el nodo "Home PC" hasta el nodo "Bank". Esto crea una dependencia entre los dos nodos.
4. Seleccione la línea de dependencia y póngale el nombre "TCP/IP". (Esto también puede hacerse editando el valor de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



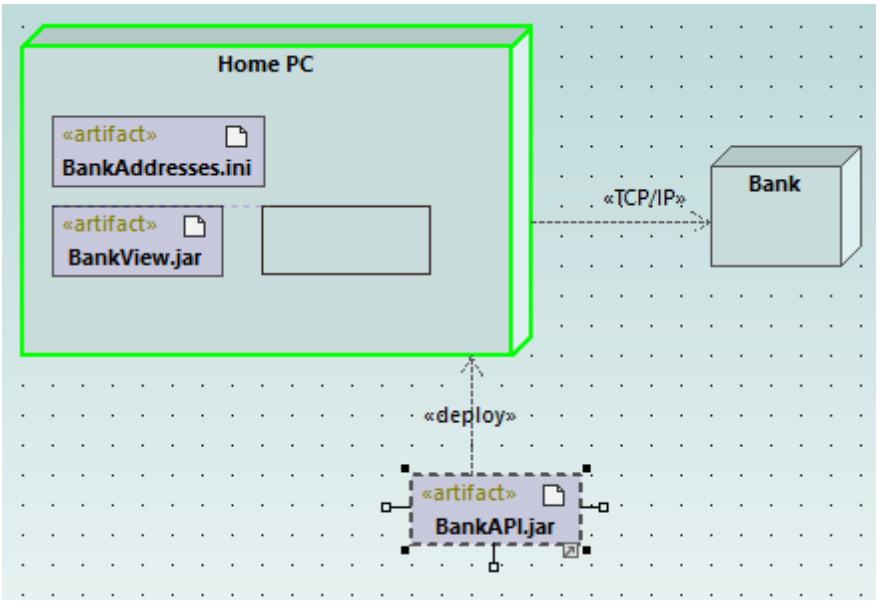
Incrustar artefactos

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 incluir dependencias de implementación entre estos artefactos y el nodo "Home PC" por lo que todas estas dependencias se pueden ver en el diagrama:

© 2018 Altova GmbH

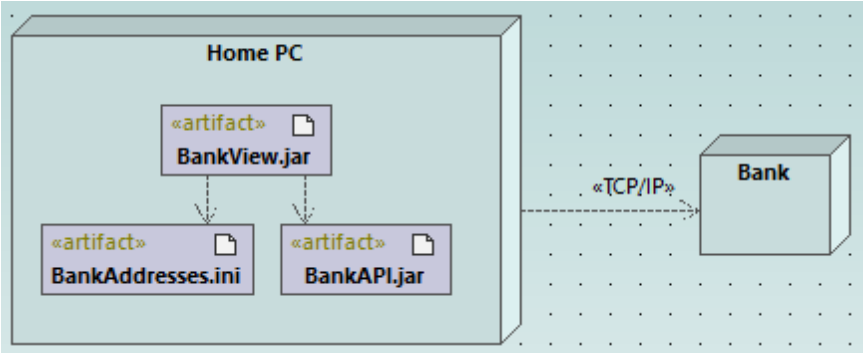


También puede incrustar los artefactos en el nodo "Home PC" (arrastrando uno a uno todos los artefactos hasta el nodo). Observe que las dependencias de implementación ya no se ven en el diagrama pero siguen existiendo lógicamente.



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 una dependencia de implementación automáticamente.

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*.

© 2018 Altova GmbH

3.7 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 las mismas instrucciones.

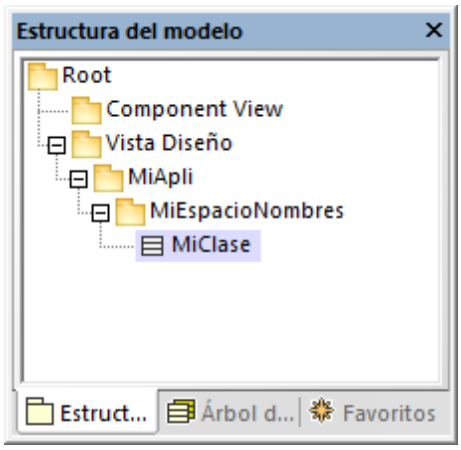
Crear un proyecto de UModel nuevo

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 "Vista Diseño".
2. Haga clic con el botón derecho en "Vista Diseño" y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "MiApli"
3. Haga clic con el botón derecho en "MiApli" y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "MiEspacioNombres".
4. Haga clic con el botón derecho en "MiEspacioNombres" 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:

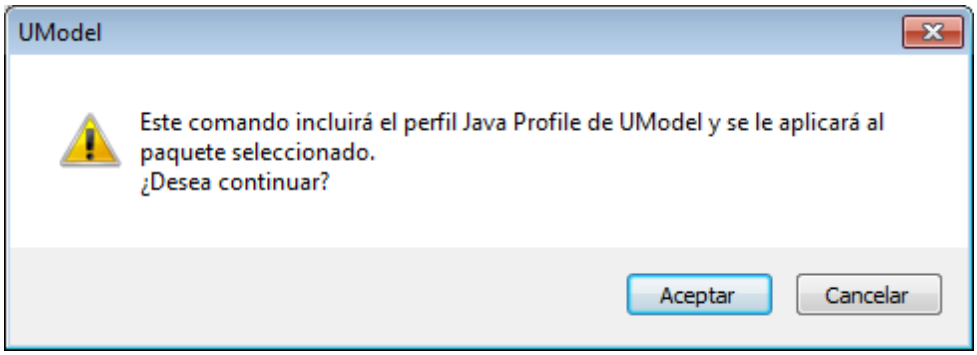
- Debe tener definido un paquete raíz de espacio de nombres Java, C# o VB.NET.
- Debe existir un componente que sea realizado por todas las clases o interfaces para las que se debe generar código.
- El componente debe tener asignada una ubicación física (un directorio). El código se generará en dicha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

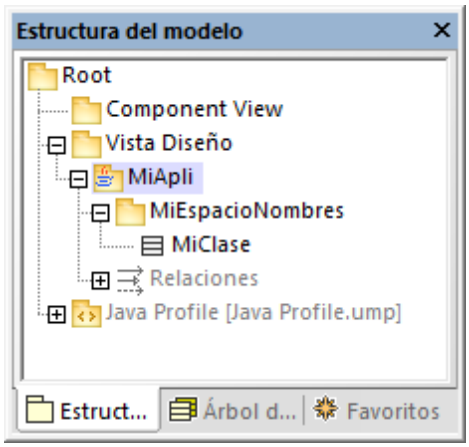
Si se valida el proyecto en este momento, la ventana Mensajes emite el error de validación "`No se encontró la raíz de espacio de nombres. Utilice el menú contextual (submenú "Ingeniería de código") en la estructura del modelo para definir un paquete como raíz de espacio de nombres`." Para resolver este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

1. Haga clic con el botón derecho en el paquete "MiApli" 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**.

 © 2018 Altova GmbH

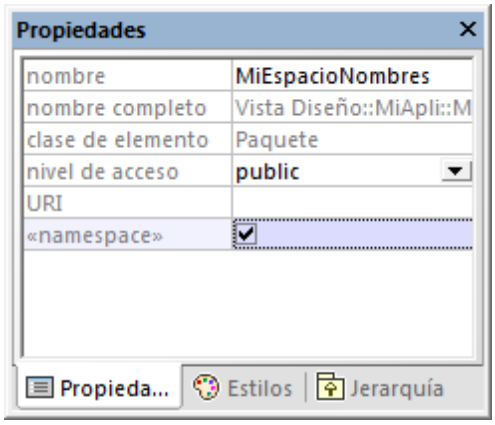


Observe que ahora el paquete tiene un icono distinto ( ) que nos indica que este paquete es una raíz de espacio de nombres de Java. Además, se añadió el perfil Java Profile al proyecto.



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

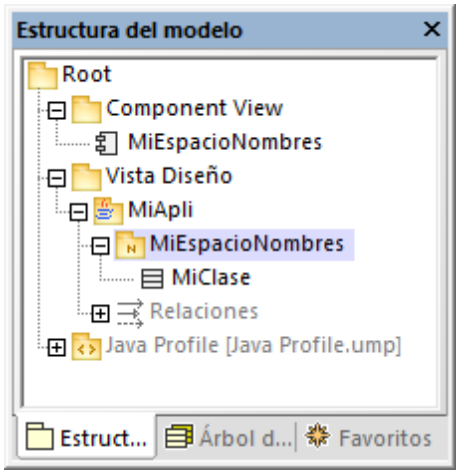
1. Seleccione el paquete "MiEspacioNombres" en la ventana *Estructura del modelo*.
2. En la ventana *Propiedades* habilite la propiedad `<<namespace>>`.



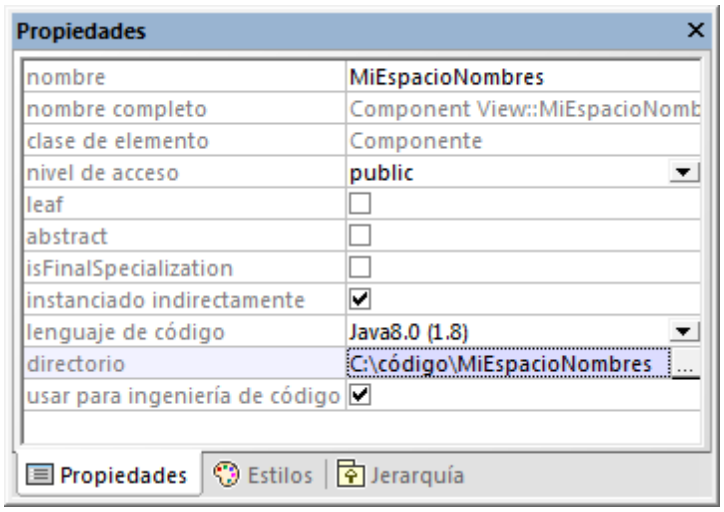
Observe que ahora el paquete tiene un icono distinto ( ) que nos indica que este paquete es un espacio de nombres.

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 "MiEspacioNombres".



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



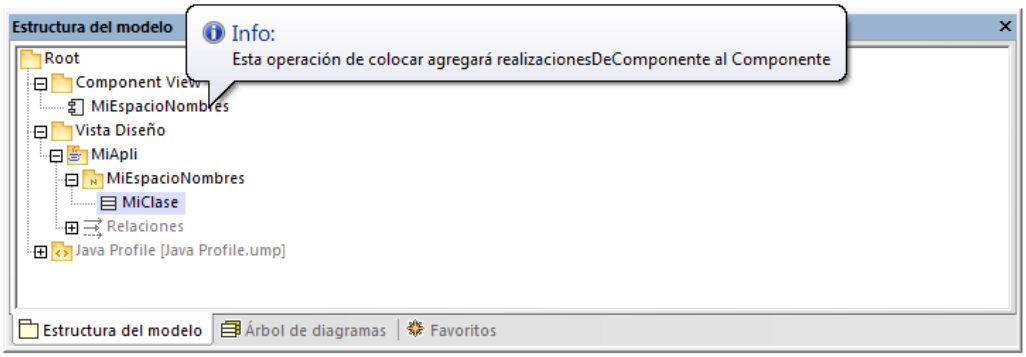
Nota: el directorio de generación de código también puede ser relativo al proyecto .ump. Por ejemplo, si se usa la ruta de acceso **código\MiEspacioNombres**, el directorio **código** se creará en el mismo directorio donde reside el proyecto de UModel.

 © 2018 Altova GmbH

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 "MiEspacioNombres" tiene la ruta de acceso "C:\código\MiEspacioNombres", 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*.

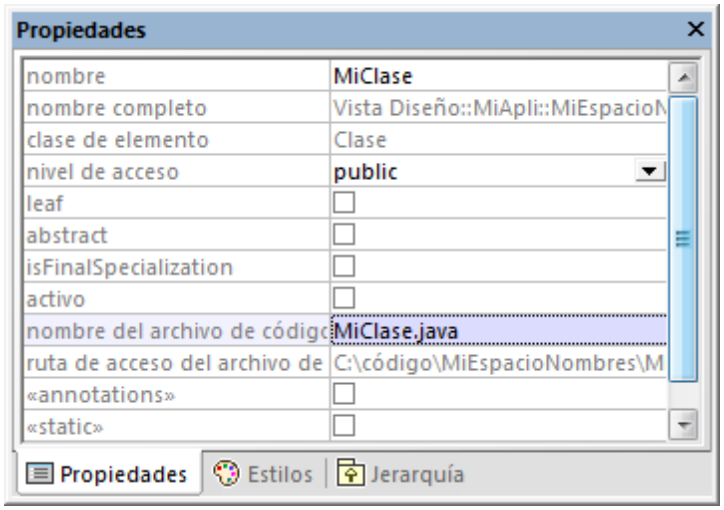
Ahora veamos cómo se puede crear la relación de realización de componente. En la ventana *Estructura del modelo* haga clic en la clase "MiClase" y arrástrela hasta el componente "MiEspacioNombres".



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:

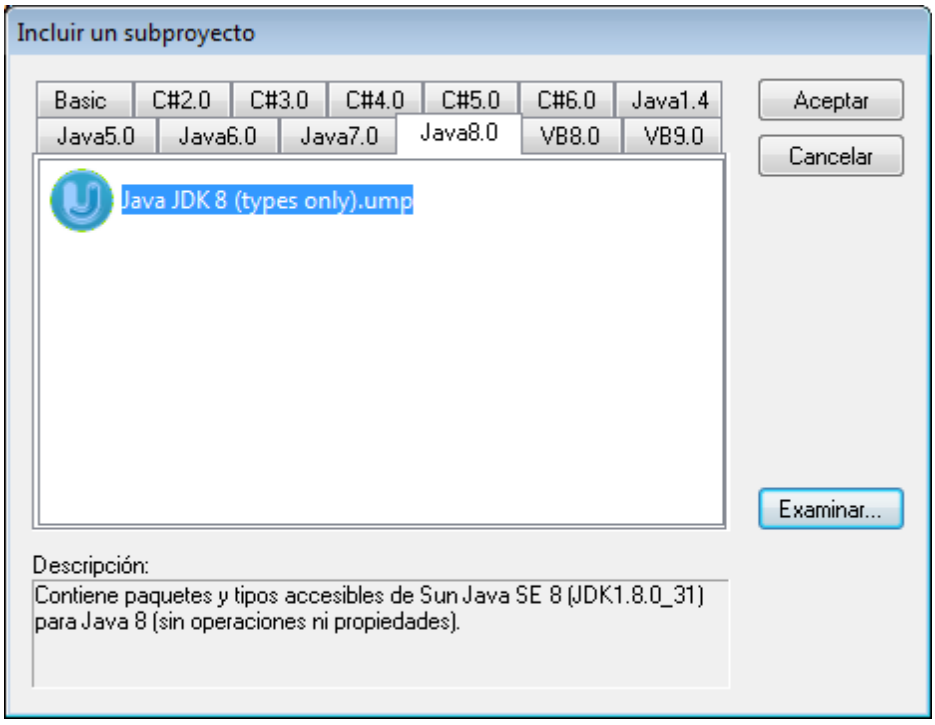
1. Seleccione la clase "MiClase" en la ventana *Estructura del modelo*.
2. En la ventana *Propiedades* cambie el valor de la propiedad `nombre del archivo de código` por el nombre correspondiente (p. ej. `MiClase.java`).



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# y VB.NET):

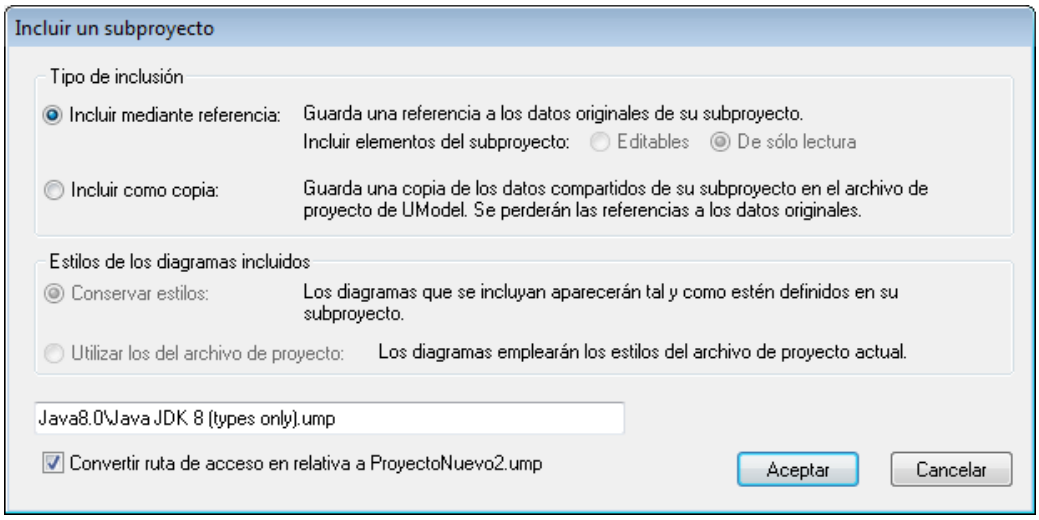
1. En el menú **Proyecto** haga clic en **Incluir un subproyecto**.
2. Haga clic en la pestaña **Java 8.0**, seleccione el proyecto **Java JDK 8 (types only)** y haga clic en **Aceptar.**



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

 © 2018 Altova GmbH

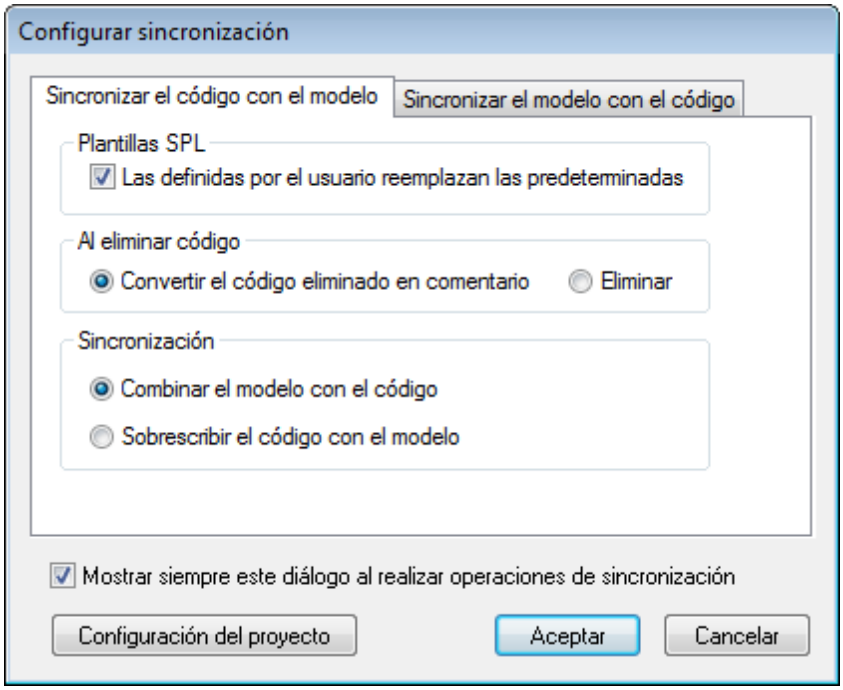
opción *Incluir mediante referencia*.



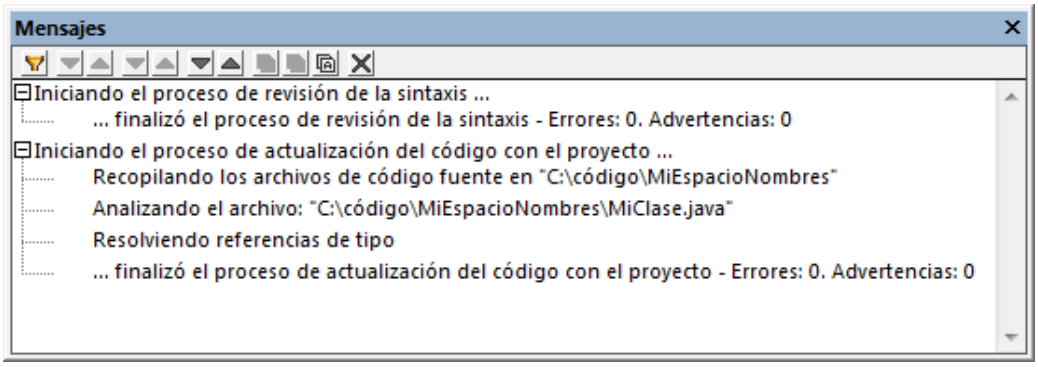
Generar código

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 (p. ej. `public float sum(float num1, float num2){}`). El archivo **MiClase.java** tendrá este aspecto:

```
package MiEspacioNombres;
public class MiClase
{
  public float sum(float num1, float num2){}
}
```

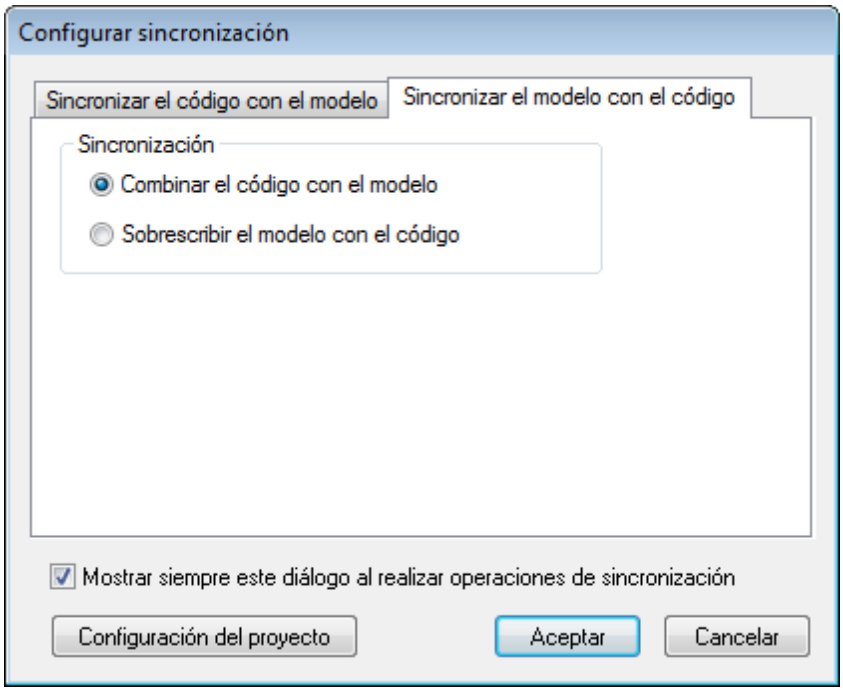
MiClase.java

Combinar cambios realizados en el código con el modelo original

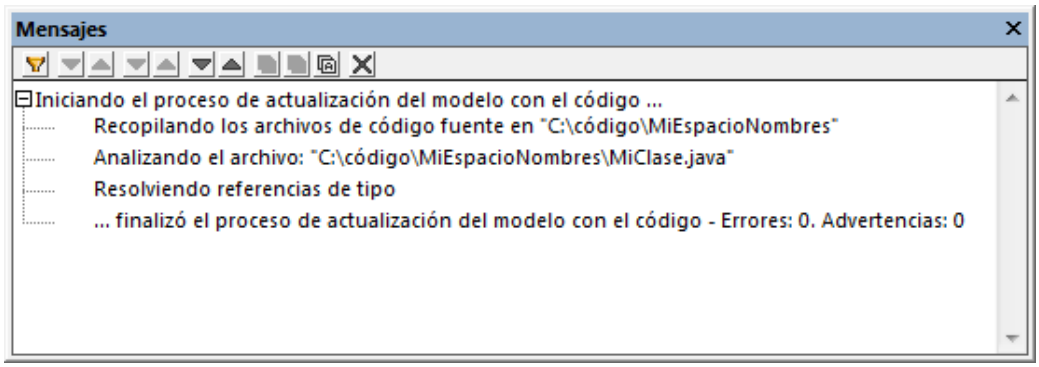
Ahora podemos combinar los cambios realizados en el código con el código original:

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

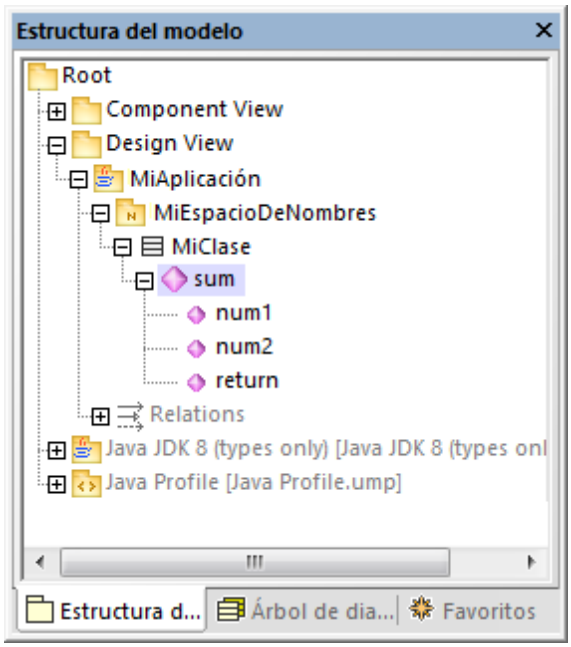
 © 2018 Altova GmbH



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*.



© 2018 Altova GmbH

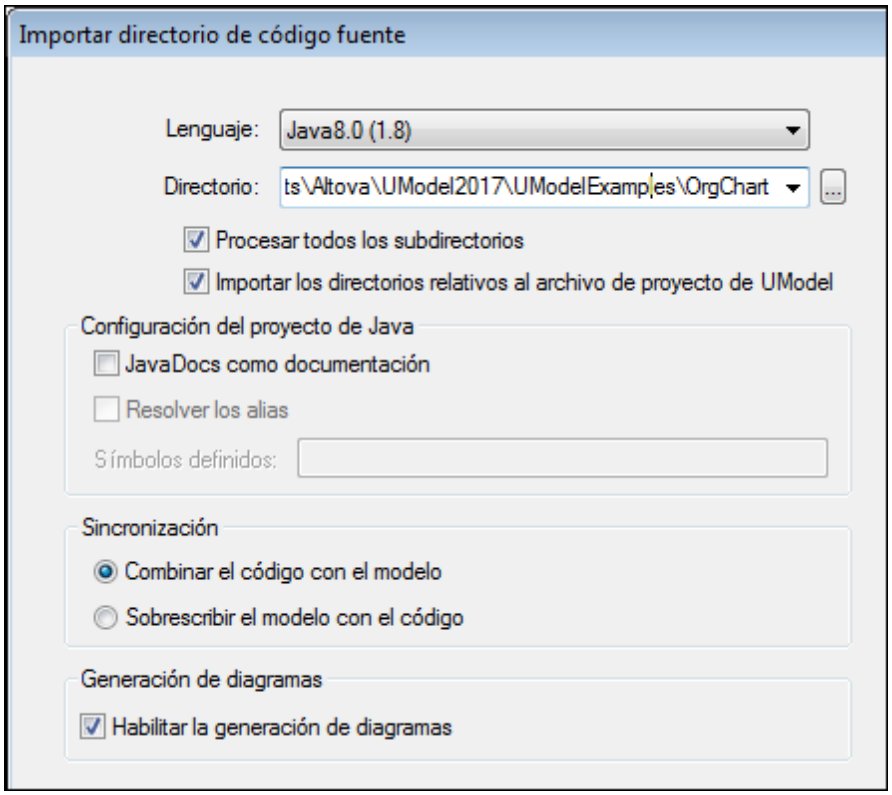
3.8 Ingeniería inversa (del código al modelo)

En este apartado del tutorial explicamos cómo importar código de programa desde un directorio hasta un proyecto de UModel nuevo (ingeniería inversa). También aprenderá a agregar una clase nueva al modelo, a prepararla para la generación de código y a 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\UModel2018\UModelExamples\OrgChart.zip**. Antes de empezar el tutorial deberá descomprimirlo en el mismo directorio.

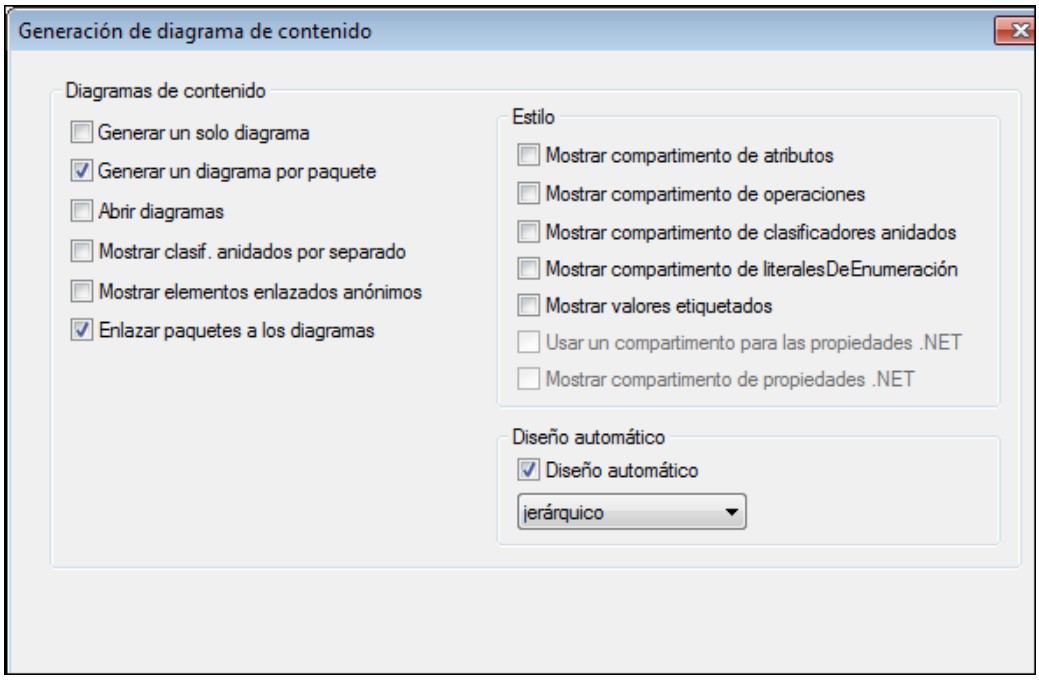
Importar código desde un directorio

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.

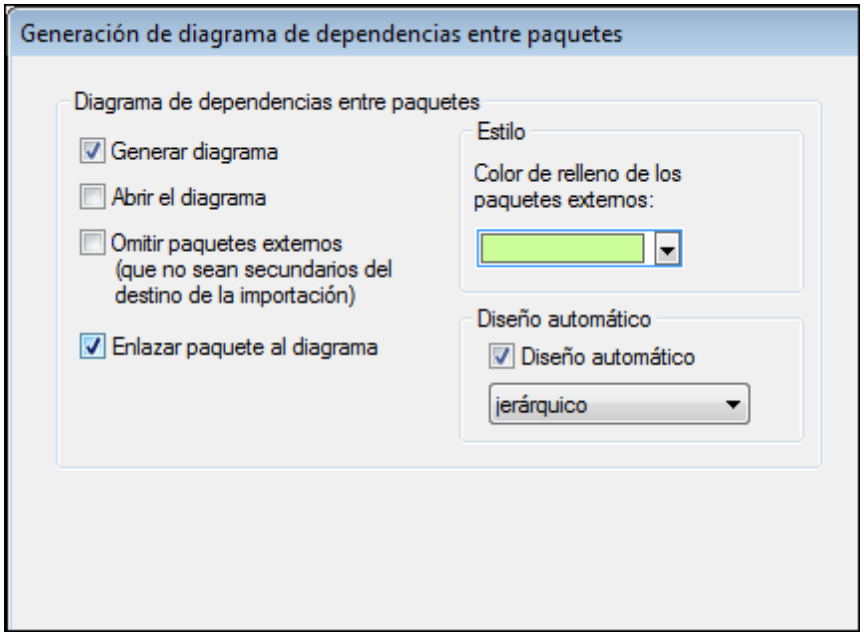


5. En la siguiente pantalla marque la casilla *Generar un diagrama por paquete*. Esto ordena a UModel que cree un diagrama nuevo por cada paquete. Las opciones de estilo de los

diagramas se pueden configurar aquí o más adelante.

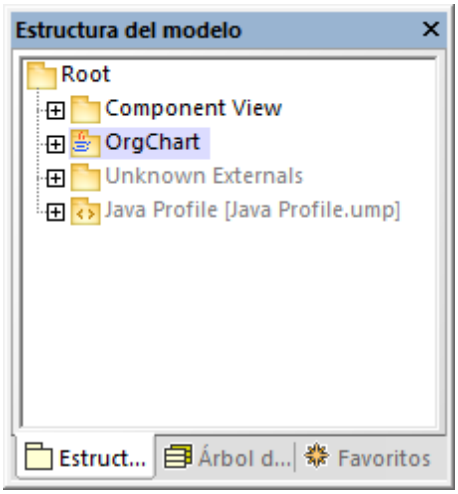


6. Haga clic en **Siguiente** para continuar. En la siguiente pantalla puede definir las opciones de generación de dependencias entre paquetes.

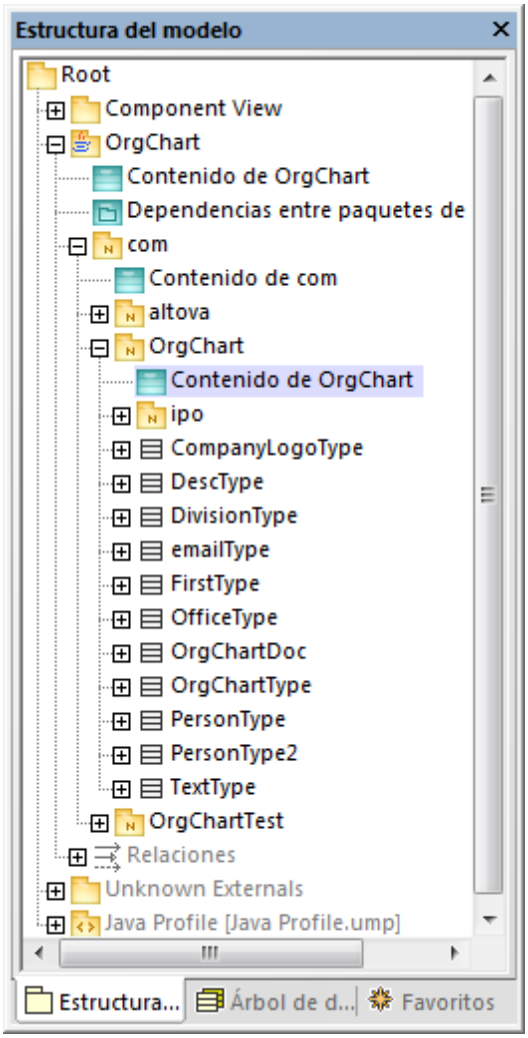


7. Haga clic en **Finalizar**. Cuando la aplicación lo solicite, guarde el modelo nuevo en un directorio del sistema. Los datos se analizan y se crea un paquete nuevo llamado "**OrgChart**".

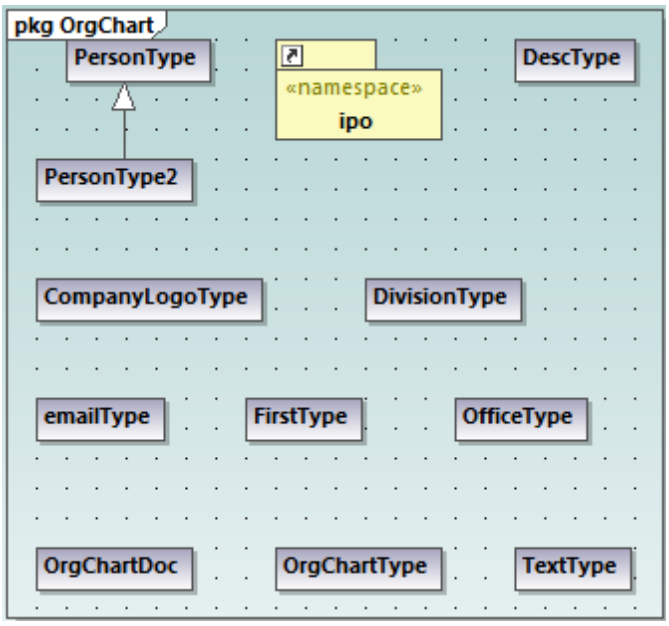
 © 2018 Altova GmbH



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.

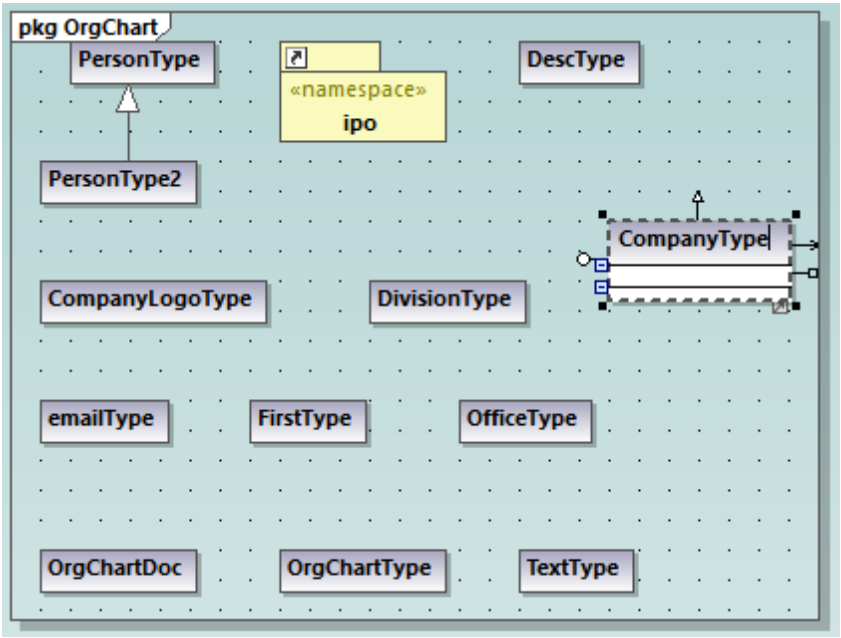


Agregar una clase nueva al diagrama OrgChart

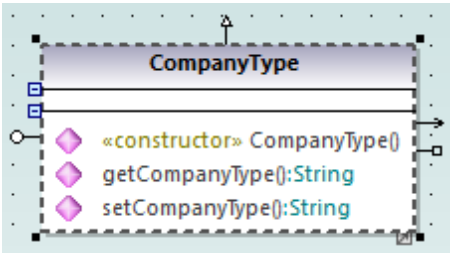
Hasta ahora hemos creado un modelo mediante ingeniería inversa a partir de código Java que ya teníamos. El modelo que hemos creado también incluye varios diagramas que se generaron automáticamente. Ahora vamos a ampliar el modelo con una clase nueva.

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**.

 © 2018 Altova GmbH



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

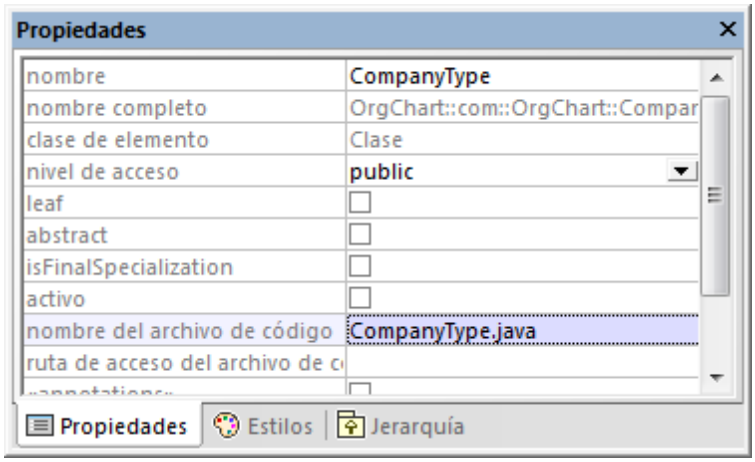


Nota: como el nombre de la clase es `CompanyType`, a la operación `CompanyType()` se le asigna automáticamente el estereotipo `<<constructor>>`.

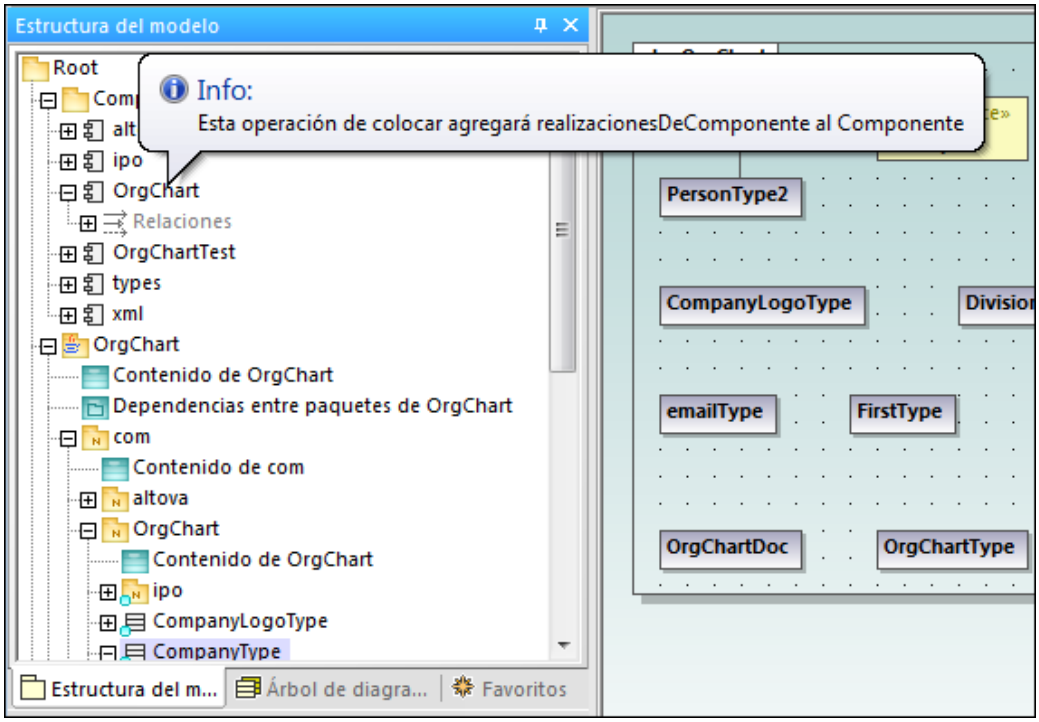
Habilitar la clase nueva para la generación de código

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

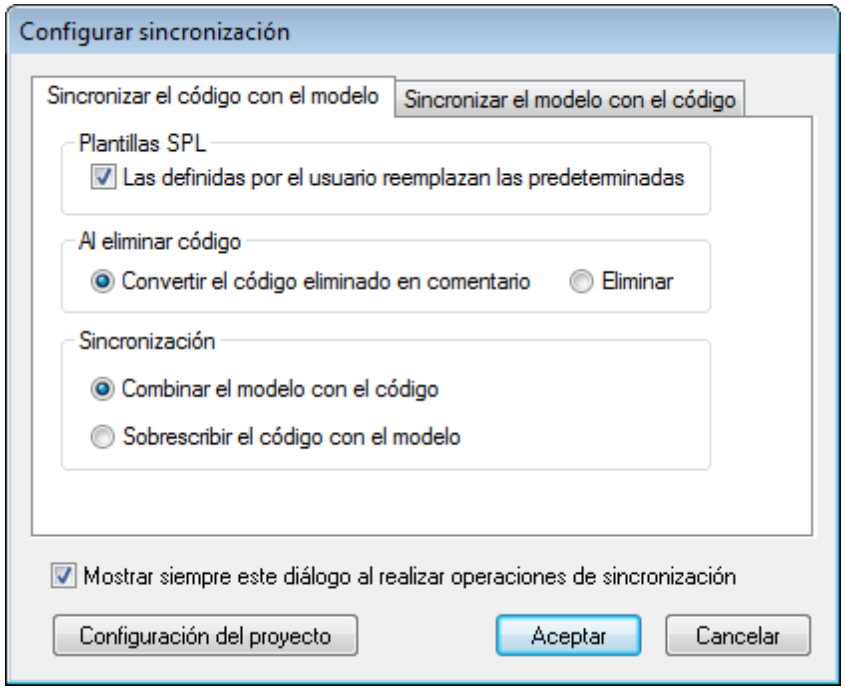


Esto crea una relación de tipo "RealizaciónDeComponente" entre una clase y un componente. También puede crear esta relación dibujando una línea de relación en el diagrama de componentes (*véase* *Diagramas de componentes*) Expanda el elemento `Relaciones` del componente **OrgChart** para ver la relación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

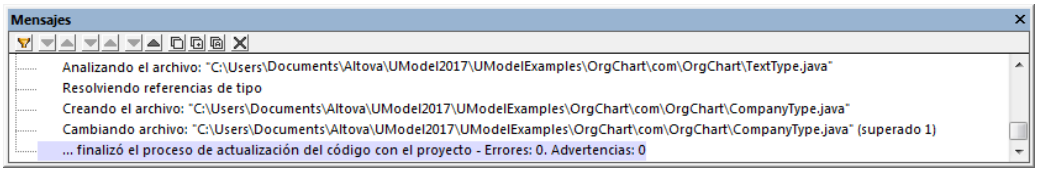
Combinar código de programa con un paquete

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.

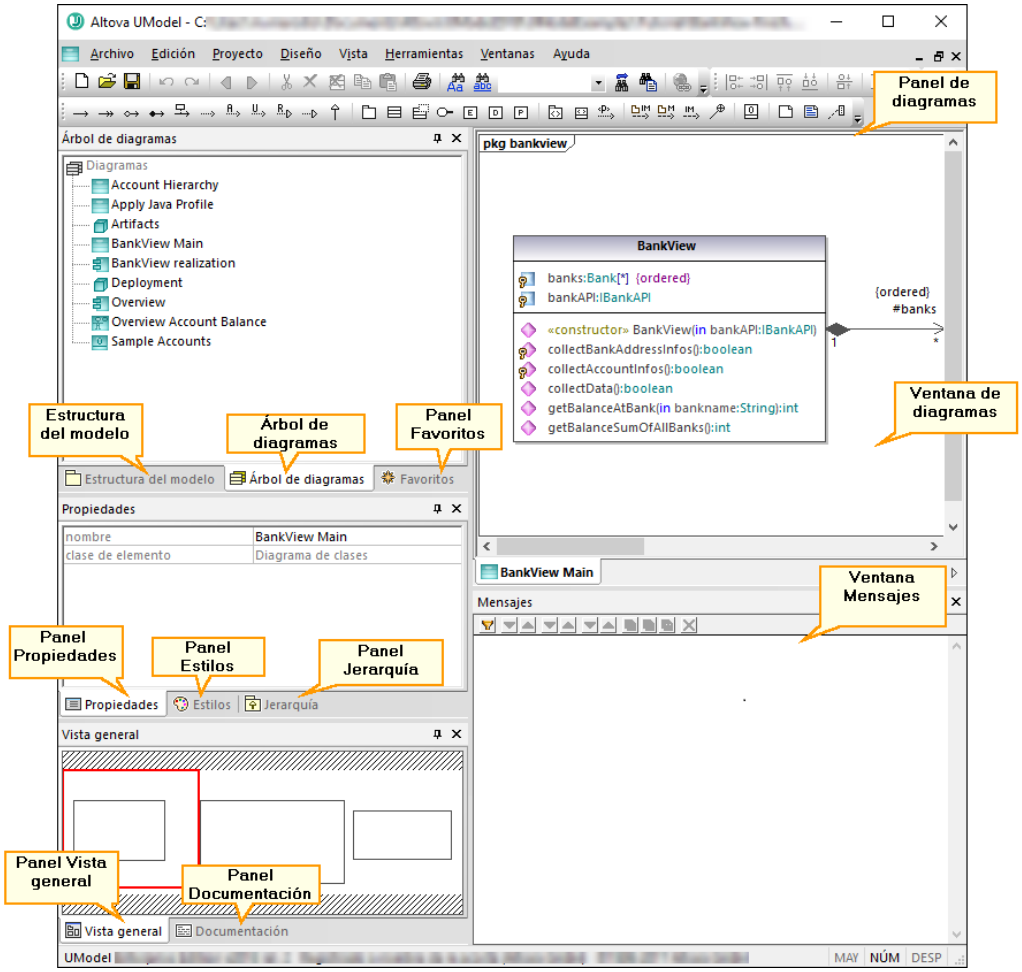
© 2018 Altova GmbH

Altova UModel 2018

Interfaz gráfica del usuario de UModel

4 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 diagramas 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.

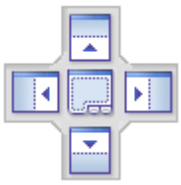
© 2018 Altova GmbH

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 desplegable.

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:

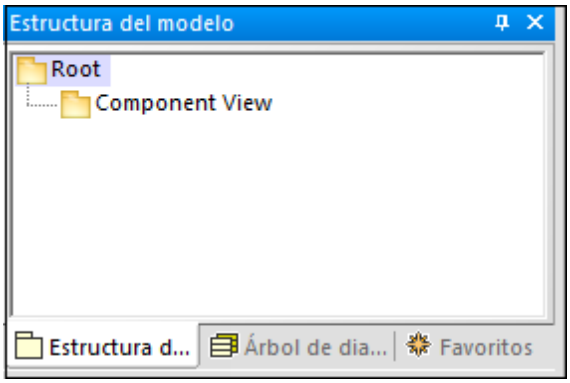
- En el menú **Herramientas** haga clic en **Restaurar barras de herramientas y ventanas**.

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

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

4.1 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\UModel2018\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.

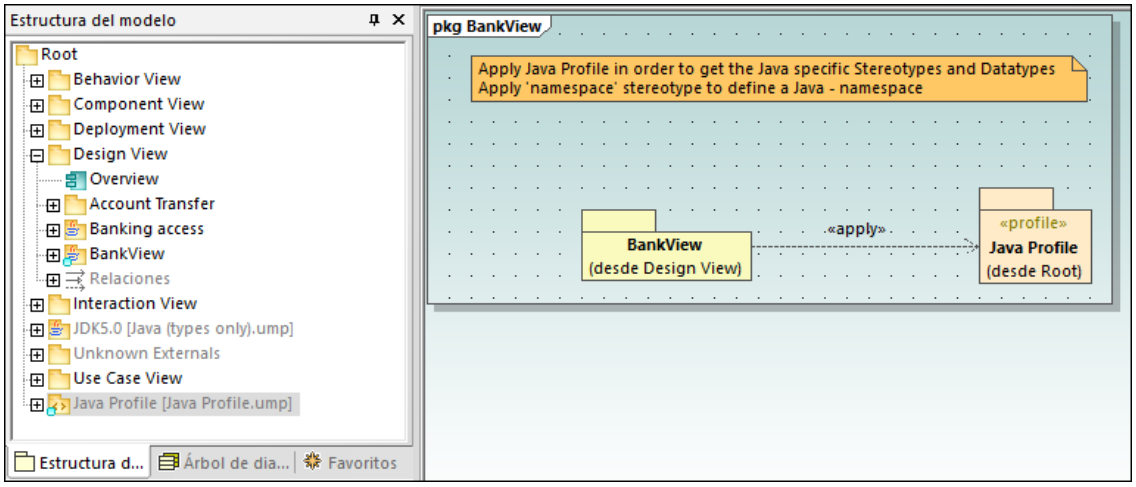
Para contraer los paquetes, pulse la tecla guion (-) del teclado. Para contraer todos los

 © 2018 Altova GmbH

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 "BankView" 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 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 de espacio de nombres (paquete al que se aplica el estereotipo

Icono	Descripción
	<<namespace>>)
	Perfil UML

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

Icono	Descripción
	Diagrama de actividades
	Diagrama de clases
	Diagrama de comunicación
	Diagrama de componentes
	Diagrama de estructura compuesta
	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
	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.

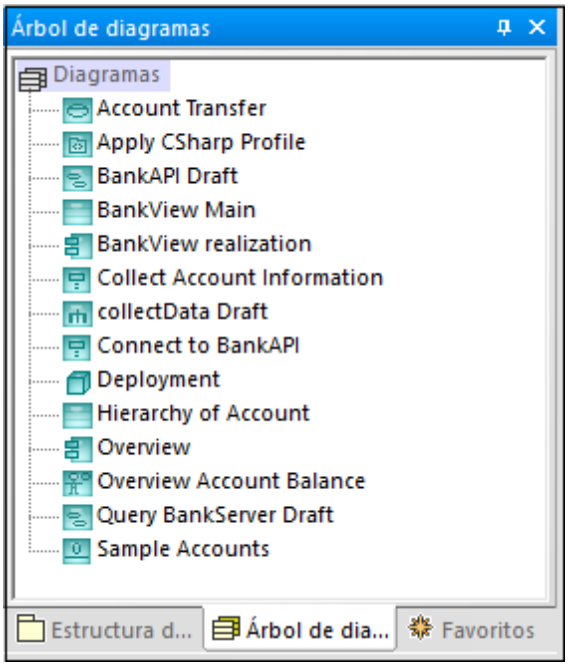
Icono	Descripción
	Clase

© 2018 Altova GmbH

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

4.2 Ventana Árbol de diagramas

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



Panel Árbol de diagramas

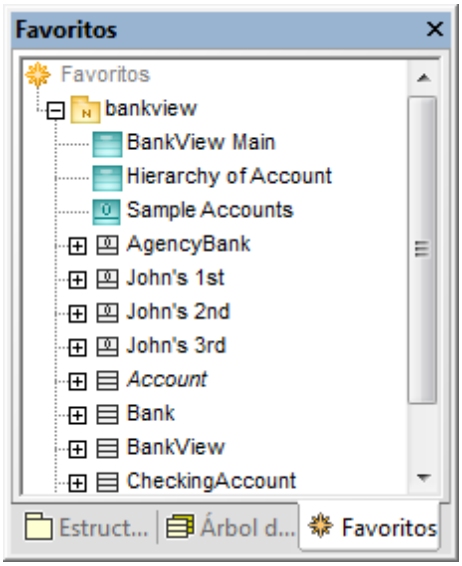
Los diagramas de este panel se puede mostrar como una lista ordenada por orden alfabético o agrupados por tipo. Para cambiar las opciones de visualización haga clic con el botón derecho en la ventana y marque o desmarque la opción **Agrupar según el tipo de diagrama**.

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.

© 2018 Altova GmbH

4.3 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*.

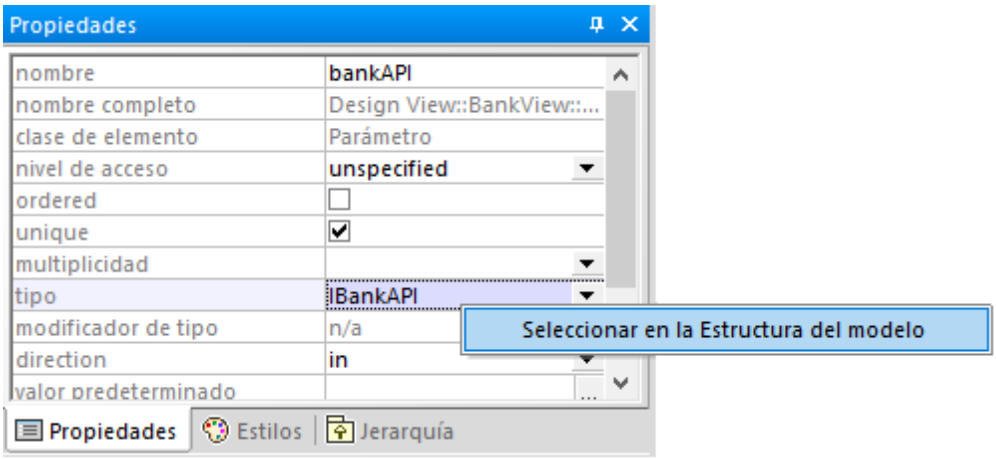
El elemento que aparece en la ventana *Favoritos* es una vista del elemento (es decir, no es ni una copia ni un clon del elemento). Las acciones que aplique en la ventana *Estructura del modelo* también se aplican en la ventana *Favoritos*, lo que incluye añadir o eliminar elementos. Para más información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

4.4 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

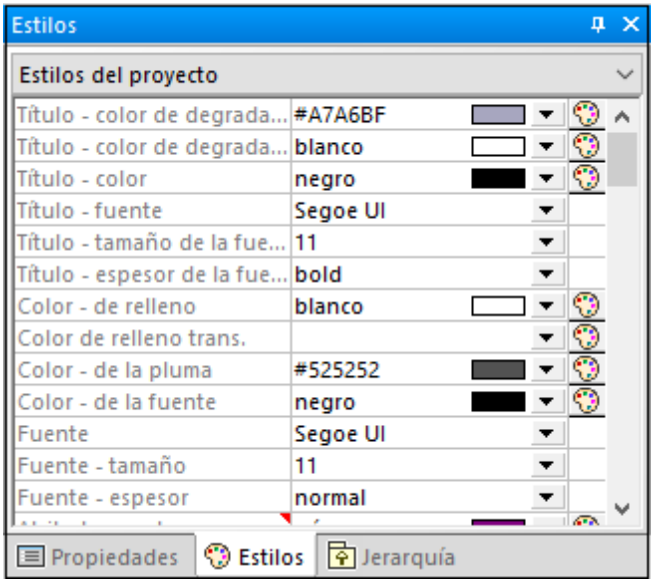
Cualquier cambio en una propiedad de un elemento de la ventana *Propiedades* se refleja inmediatamente en el diagrama. Asimismo, cualquier cambio que se efectúe en el diagrama (por ejemplo, cambiar la visibilidad de una operación de `pública` a `privada`) se verá reflejado en la propiedad correspondiente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

4.5 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

4.6 Ventana Jerarquía

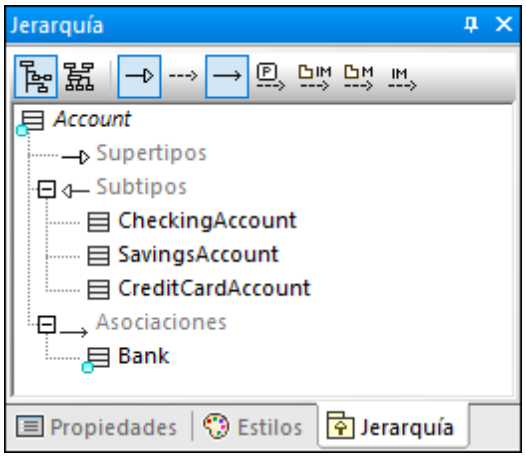
La ventana *Jerarquía* muestra todas las relaciones del elemento de modelado seleccionado en ese momento. El elemento de modelado se puede seleccionar en un diagrama, en la ventana *Estructura del modelo* o en la ventana *Favoritos*.

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.

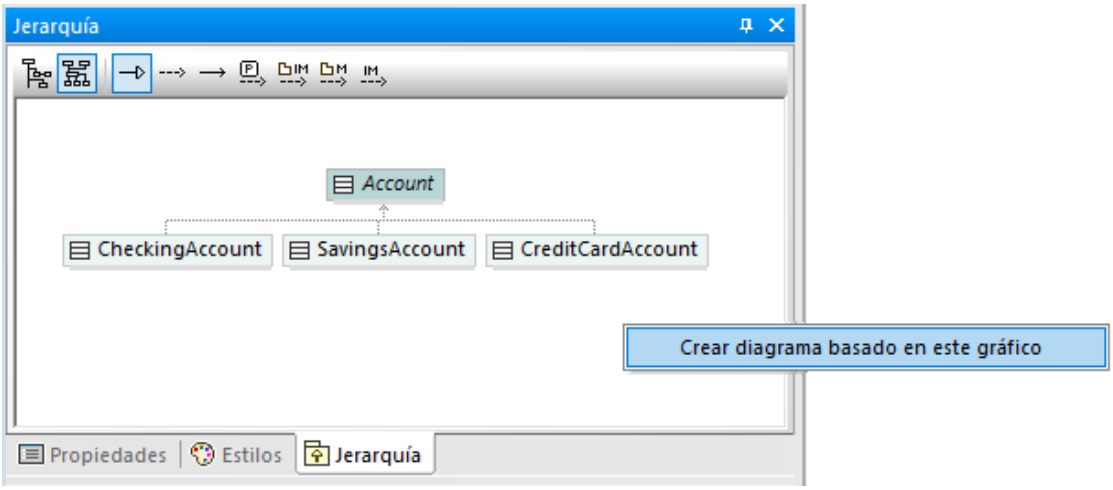
La *vista en forma de árbol* muestra en forma de árbol todas las relaciones del elemento seleccionado en ese momento. Con los botones de la parte superior de la ventana puede seleccionar qué tipos de relaciones quiere que se muestren. En la imagen siguiente solo se han seleccionado las generalizaciones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

© 2018 Altova GmbH



Panel Jerarquía (vista en forma de diagrama)

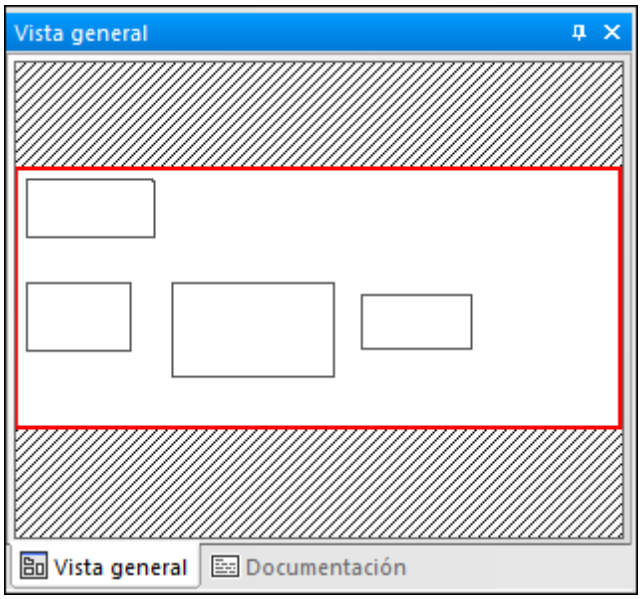
La *vista en forma de árbol* permite generar diagramas que incluyan los elementos visibles en la ventana. Para ello haga clic con el botón derecho dentro de la ventana y seleccione la opción **Crear diagrama basado en este gráfico** en el menú contextual.

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*.

4.7 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.



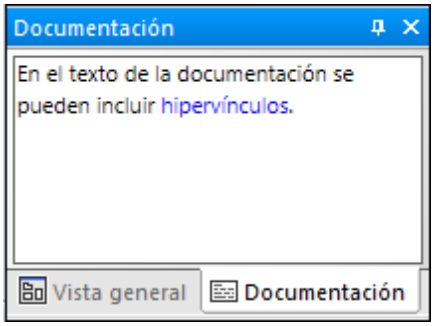
Panel Vista general

Consulte también Acercar y alejar diagramas.

© 2018 Altova GmbH

4.8 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)**.



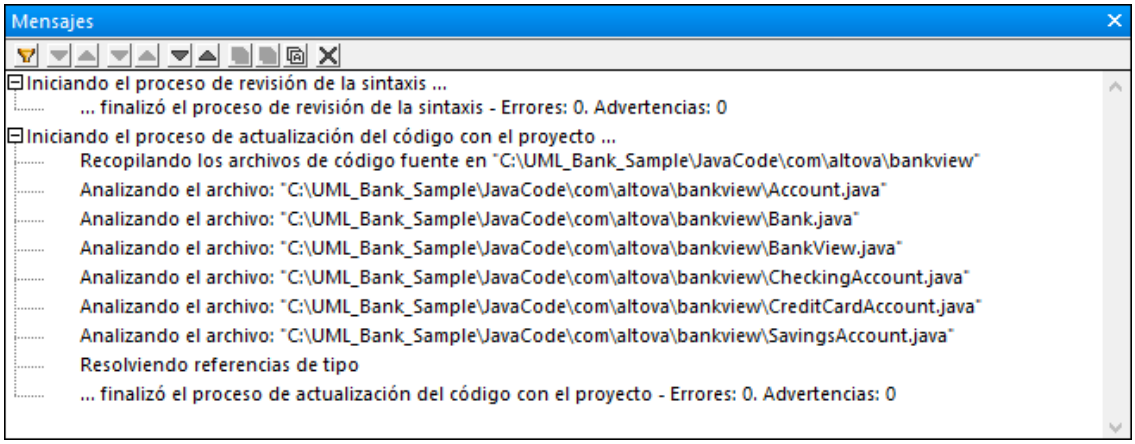
Panel documentación

Se puede pasar el corrector ortográfico al texto de la ventana *Documentación*. Haga clic con el botón derecho dentro de la ventana y seleccione la opción **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.

4.9 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, 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.

© 2018 Altova GmbH

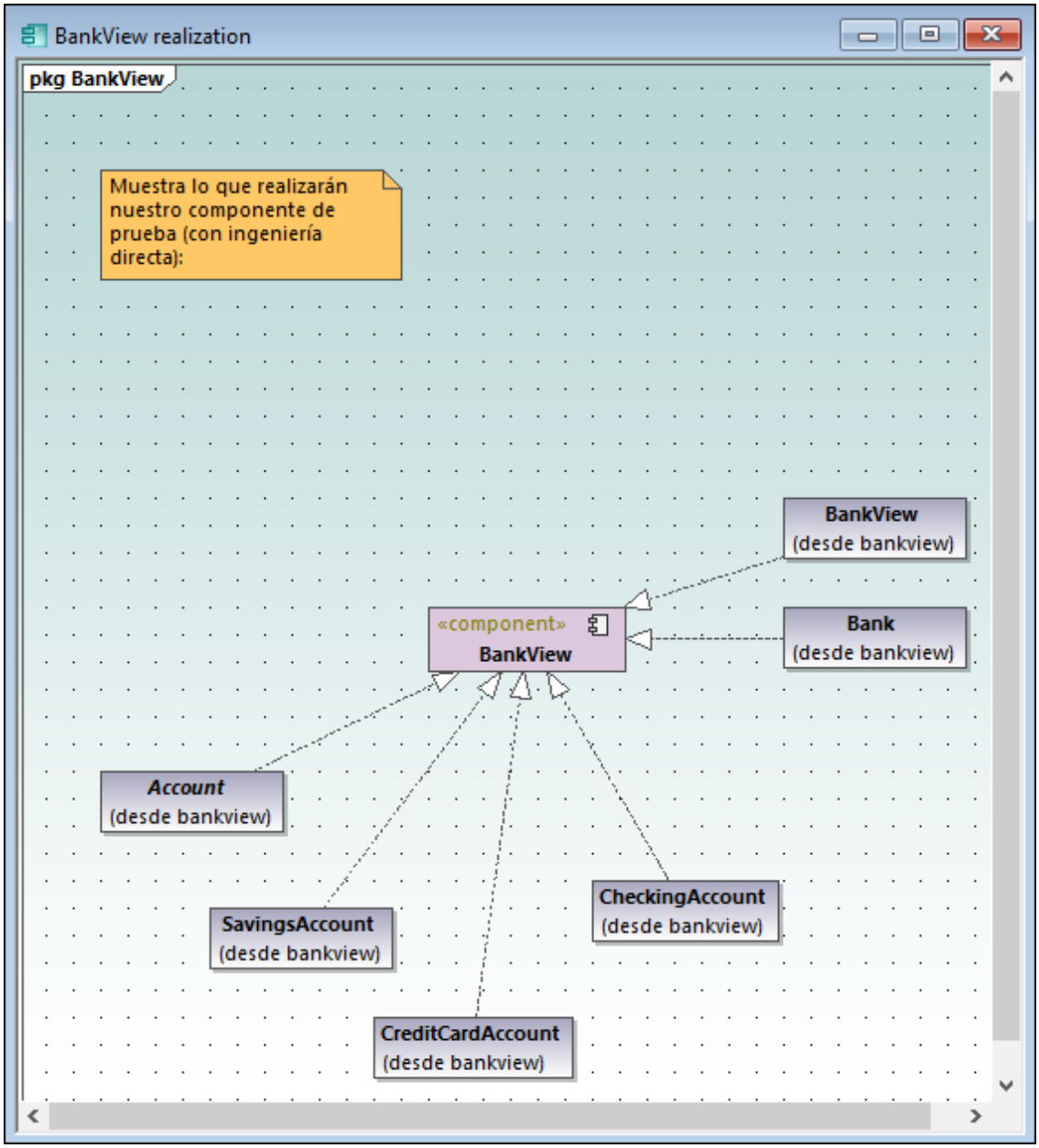
Icono	Descripción
	Ir a la advertencia anterior.
	Ir a la siguiente línea.
	Ir a la línea anterior.
	Copiar la línea seleccionada en el portapapeles.
	Copiar la línea seleccionada en el portapapeles, incluidas todas sus líneas anidadas.
	Copiar todo el contenido de la ventana Mensajes en el portapapeles.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

4.10 Ventana de diagramas

Al crear un nuevo diagrama o al abrir uno ya existente 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*).

© 2018 Altova GmbH



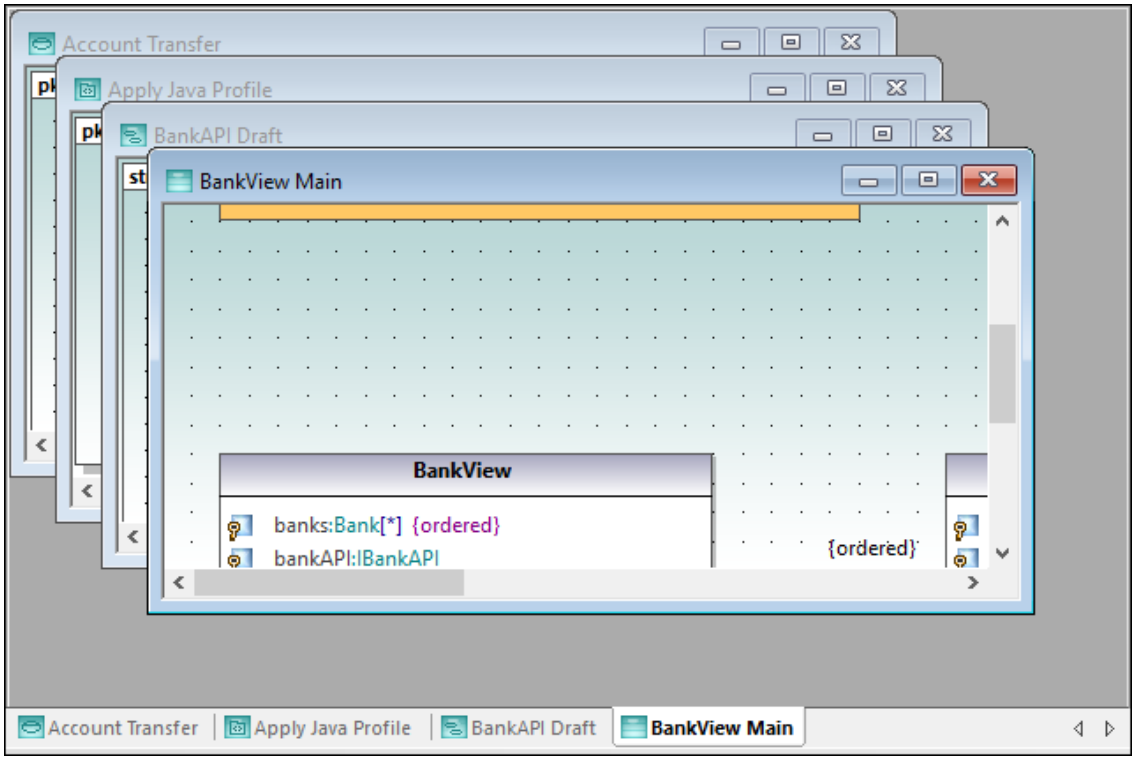
Panel Diagramas

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

4.11 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 ya existentes 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**.

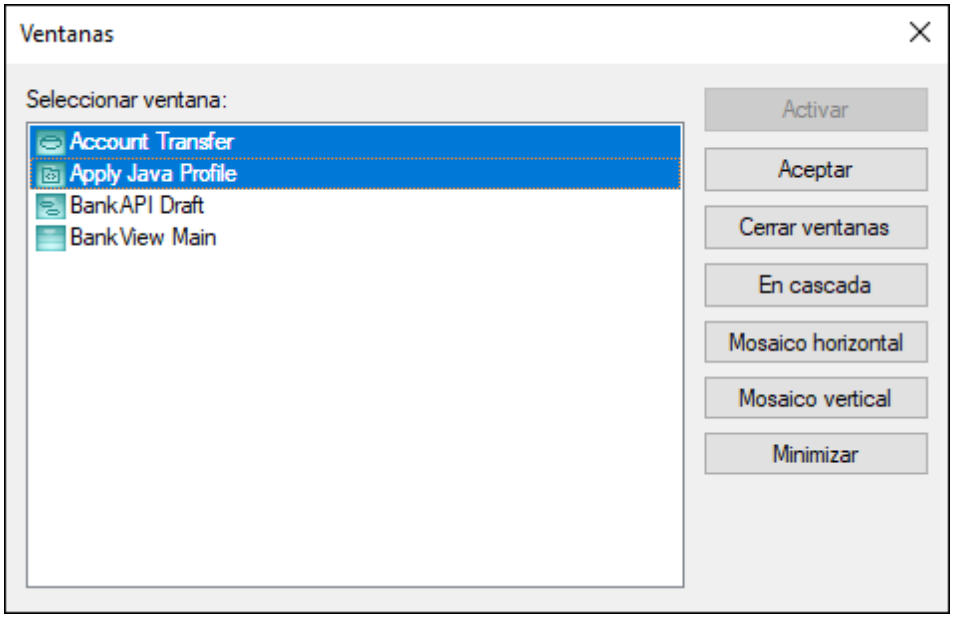


Panel Diagramas

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...**

© 2018 Altova GmbH



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**.

Altova UModel 2018

Interfaz de la línea de comandos

5 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\UModel2018`. Si trabaja con la versión de 32 bits de UModel en un sistema operativo de 64 bits, entonces la ruta de acceso del archivo `UModelBatch.exe` es `C:\Archivos de programa (x86)\Altova\UModel2018`.

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 contiene espacios, la ruta de acceso o el nombre de archivo debe ir entre comillas (p. ej. `"c:\Archivos de programa\...\MiProyecto.ump"`)

```
utilización: umodelbatch [proyecto] [opciones]

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

proyecto                ... archivo de proyecto (*.ump). Ver también
Archivo: Nuevo / Cargar / Opciones al guardar
/new[=archivo]    ... crear, guardar o guardar como proyecto nuevo
/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,*.vbproj,*.vbdpro
j,*.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
```


© 2018 Altova GmbH

```
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 | C#1.2 | C#2.0 | C#3.0 | C#4.0 | C#5.0 | C#6.0 | C#7.0 | VB7.1 |
VB8.0 | VB9.0)
/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)
/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)


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
```

En la sección de proyectos:

- el parámetro **`/new`** define la ruta de acceso y el nombre de archivo de proyecto nuevo (*.ump). También se puede usar para guardar un proyecto con otro nombre distinto (p. ej. `UModelBatch.exe MiArchivo.ump /new=MiArchivoBackup.ump`). Para más información consulte el apartado Archivo: Nuevo / Cargar / Opciones al guardar.
- el parámetro **`/set`** reemplaza las opciones de configuración predeterminadas del registro con las opciones definidas aquí.
- el parámetro **`/gui`** muestra la interfaz de UModel durante el procesamiento por lotes.

Ejemplo nº1

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

```
"C:\Archivos de programa\Altova\UModel2018\UModelBatch.exe" /new="C:\Archivos
de programa\Altova\UModel2018\UModelBatchOut\Fred.ump" /isd="X:TestCases
\UModel\Fred" /set /gui /iclg=Java5.0 /ipsd=1 /ijdc=1 /dgen=1 /dopn=1 /
dmax=5 /chk
```

/new:	Indica que el archivo de proyecto nuevo debe llamarse "Fred.ump" y se debe guardar en `C:\Archivos de programa\Altova\UModel2018\UModelBatchOut\`
/isd=	Indica que el directorio raíz en el que se debe importar es `X:\TestCases\UModel\Fred`

© 2018 Altova GmbH

/**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**:	Mostrar la interfaz gráfica de UModel durante el procesamiento por lotes
/**iclg**:	UModel importará el código como Java 5.0
/**ipsd**=1:	Procesar recursivamente todos los subdirectorios del directorio raíz suministrado con el parámetro **/isd**
/**pfd**=1:	Crear un paquete por cada directorio importado en el proyecto de UModel
/**ijdc**=1:	Crear JavaDocs donde corresponda a partir de los comentarios
/**dgen**=1:	Generar diagramas
/**dopn**=1:	Abrir los diagramas generados
/**chk**:	Revisar la sintaxis

Ejemplo nº2

Este comando importa código fuente desde `X:\TestCases\UModel` y guarda el archivo de proyecto resultante en `C:\Archivos de programa\Altova\UModel2018\UModelBatchOut\finalclass.ump`.

```
"C:\Archivos de programa\Altova\UModel2018\UModelBatch.exe" /new="C:\Archivos de programa\Altova\UModel2018\UModelBatchOut\finalclass.ump" /isd="X:\TestCases\UModel\" /iclg=Java5.0 /ipsd=1 /ijdc=1 /dgen=1 /dopn=1 /dmax=5 /dsat=1 /dsnc=1 /chk
```

/**dsat**=1	Suprimir los atributos en los diagramas generados
/**dsnc**=1	Suprimir los clasificadores anidados en los diagramas generados

Ejemplo nº3

Este comando sincroniza código usando un archivo de proyecto ya existente (`"C:\Archivos de programa\Altova\UModel2018\UModelBatchOut\Fred.ump"`).

```
"C:\Archivos de programa\Altova\UModel2018\UModelBatch.exe" "C:\Program Files\Altova\UModel2018\UModelBatchOut\Fred.ump" /m2c /ejdc=1 /ecod=1 /emrg=1 /egfn=1 /eusc=1
```

/**m2c**	Actualizar el código con el modelo
/**ejdc**:	Generar JavaDoc a partir de los comentarios del modelo del proyecto
/**ecod**=1	Convertir el código eliminado en comentarios
/**emrg**=1	Sincronizar el código combinado
/**egfn**=1	Generar en el proyecto los nombres de archivo que falten
/**eusc**=1	Revisar la sintaxis

5.1 Archivo: Nuevo / Cargar / Opciones al guardar

Modo por lotes total (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)

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.

 © 2018 Altova GmbH

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

Altova UModel 2018

Cómo modelar

6 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 y relaciones.

Elementos	Diagramas	Relaciones
Crear elementos	Crear diagramas	Crear relaciones entre elementos
Insertar elementos del modelo en un diagrama	Generar diagramas	Cambiar el estilo de las líneas y relaciones
Renombrar, mover y copiar elementos	Abrir diagramas	Ver las relaciones de los elementos
Borrar elementos	Borrar diagramas	Asociaciones
Buscar y reemplazar texto	Cambiar el estilo de los diagramas	Asociación de colecciones
Comprobar si se están usando ciertos elementos y dónde	Alinear y ajustar el tamaño de elementos de modelado	Contención
Restricción de elementos	Finalización automática en clases	
Agregar hipervínculos a elementos	Acercar y alejar 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\UModel2018\UModelExamples**.

© 2018 Altova GmbH

6.1 Elementos

6.1.1 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:

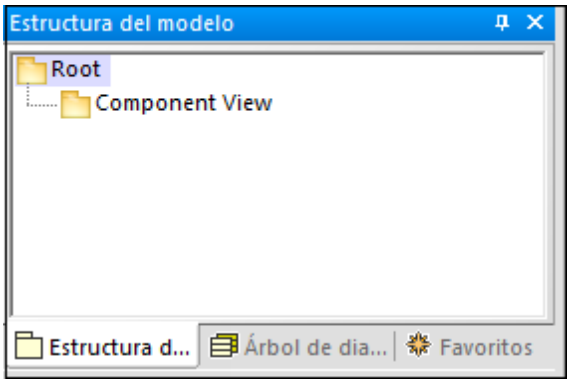
1. Cree un nuevo diagrama (*véase Crear diagramas*) o abra uno ya existente (*véase Abrir diagramas*).
2. Ahora puede:
   a. Hacer clic con el botón derecho dentro del diagrama y seleccionar **Nuevo/a | <Nombre del elemento>** en el menú contextual.
   b. Haga clic en el botón de la barra de herramientas 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

6.1.2 Insertar elementos del modelo en un diagrama

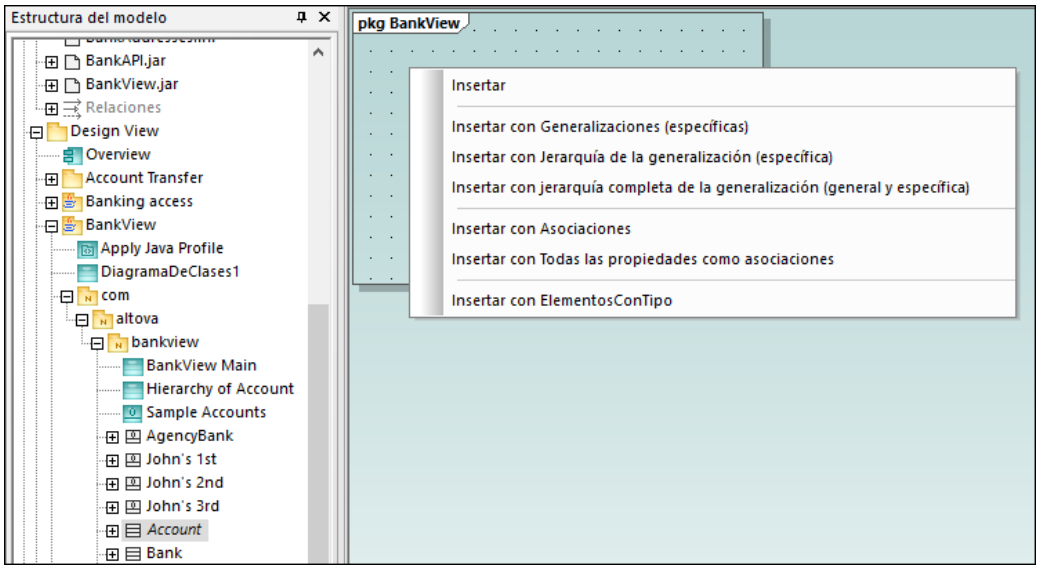
Los elementos del modelo se pueden insertar en un diagrama tanto individualmente como en grupo. Para seleccionar varios elementos de la ventana *Estructura del modelo*, mantenga pulsada la tecla **Ctrl** y vaya haciendo clic sobre los elementos que quiere seleccionar. Hay dos maneras de insertar elementos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

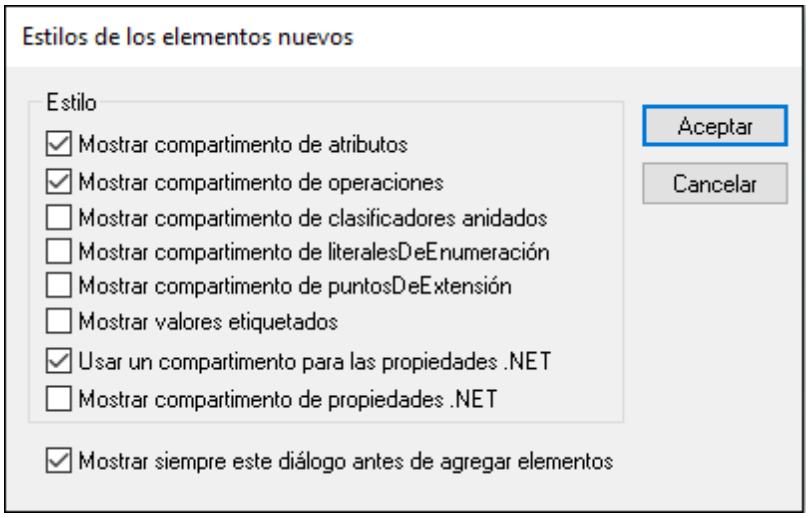
Por ejemplo, supongamos que quiere crear un nuevo diagrama de clases a partir de una clase que ya existe en el modelo. Para ilustrar este escenario, abra el proyecto de ejemplo **Bank_MultiLanguage.ump**, que encontrará en **C:\Usuarios\<usuario>\Documentos\Altova\UModel2018\UModelExamples**. Asumiendo 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.

 © 2018 Altova GmbH



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.

6.1.3 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**.

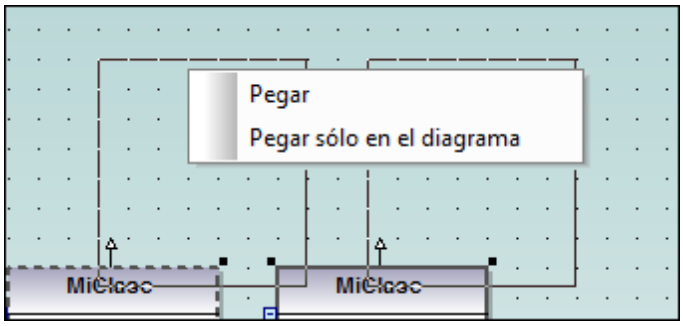
Si pega una clase en el mismo paquete, la nueva clase se crea con un número secuencial anexado al final, por ejemplo "MiClase1". Asimismo, si pega una propiedad en de la misma clase, la nueva propiedad se crea con un número secuencias anexado al final, por ejemplo "MiPropiedad1". Lo mismo ocurre para 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

 © 2018 Altova GmbH

seleccione **Pegar solo en el diagrama** en el menú contextual. El comando **Pegar solo en el diagrama** también aparece cuando arrastra un elemento ya existente 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)

6.1.4 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**.

6.1.5 Buscar y reemplazar texto

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

- Panel Diagramas
- Panel Estructura del modelo
- Panel Árbol de diagramas
- Panel Favoritos
- Panel Documentación
- Ventana Mensajes

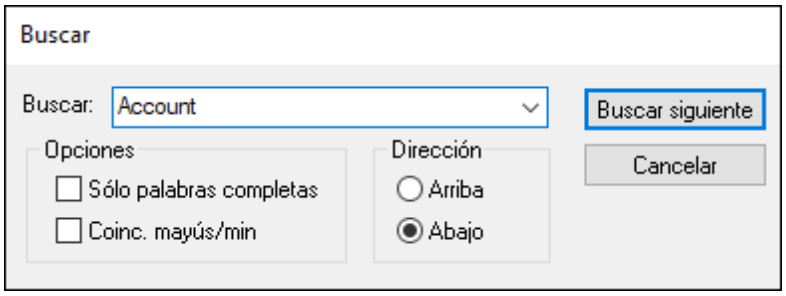
El alcance de la búsqueda lo define el lugar en que esté situado el cursor. Por tanto, si quiere buscar texto dentro de un diagrama debe hacer clic dentro de ese diagrama primero. Asimismo, si quiere buscar un elemento en el proyecto de UModel debe hacer clic dentro de la ventana *Estructura del modelo*.

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**).

 © 2018 Altova GmbH



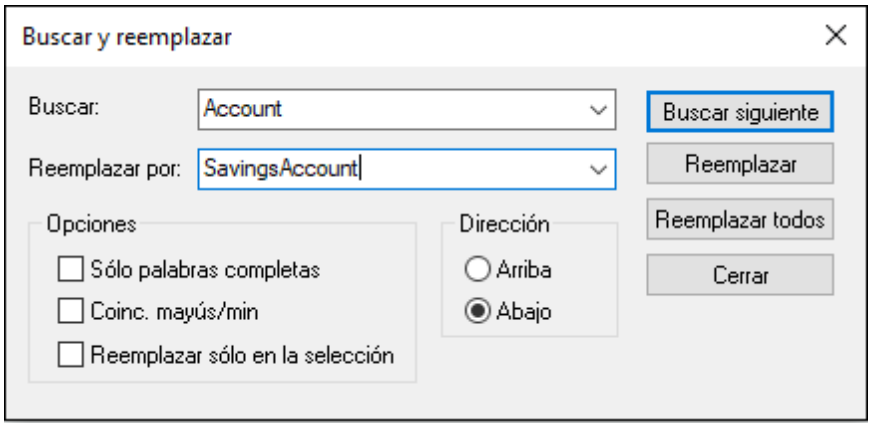
Buscar y reemplazar

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
- Estructura del modelo
- Árbol de diagramas
- Favoritos
- 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**).



6.1.6 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.

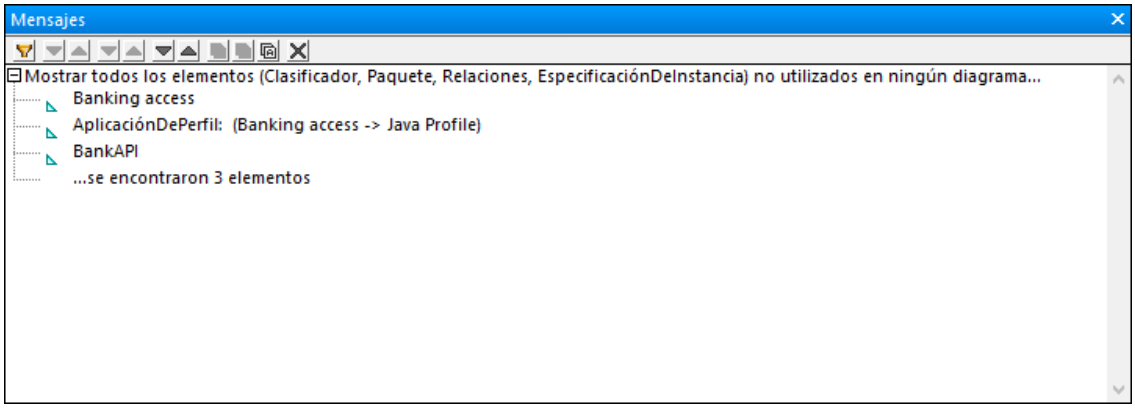
Para encontrar elementos no usados en todo el proyecto:

- 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*.

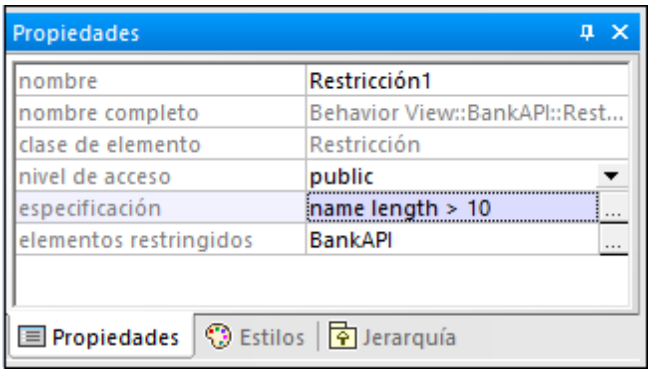
6.1.7 Restricción de elementos

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 Estructura del modelo):

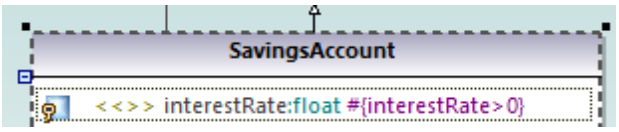
1. Haga clic con el botón derecho en el elemento que quiere restringir y seleccione **Elemento nuevo | Restricciones | Restricción**.
2. Introduzca el nombre de la restricción y pulse **Entrar**.
3. Teclee el texto de la restricción en el campo "especificación" de la ventana Propiedades (por ejemplo,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

 © 2018 Altova GmbH



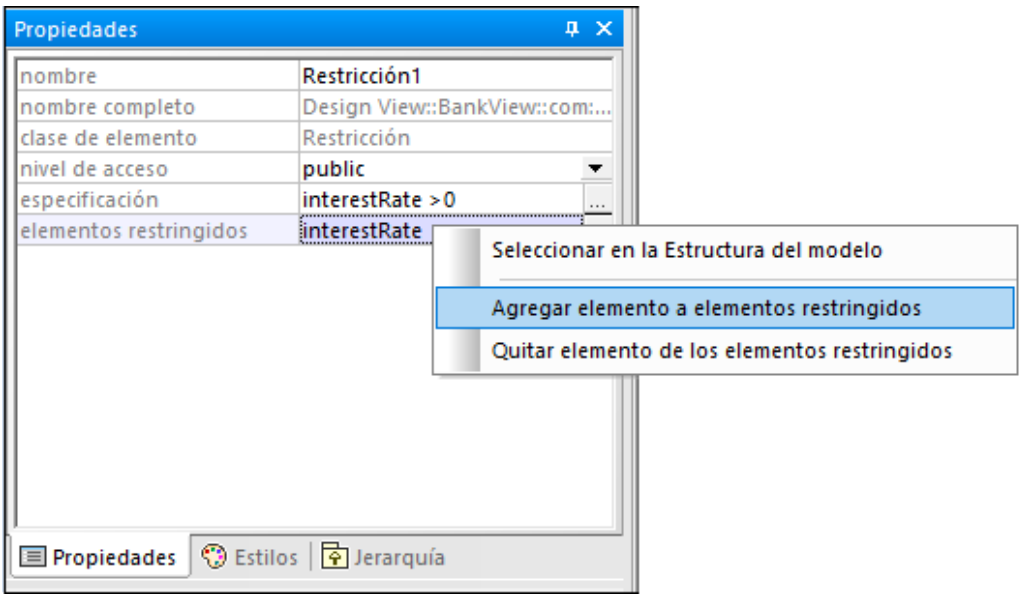
Para restringir un elemento (desde un diagrama):

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}`.

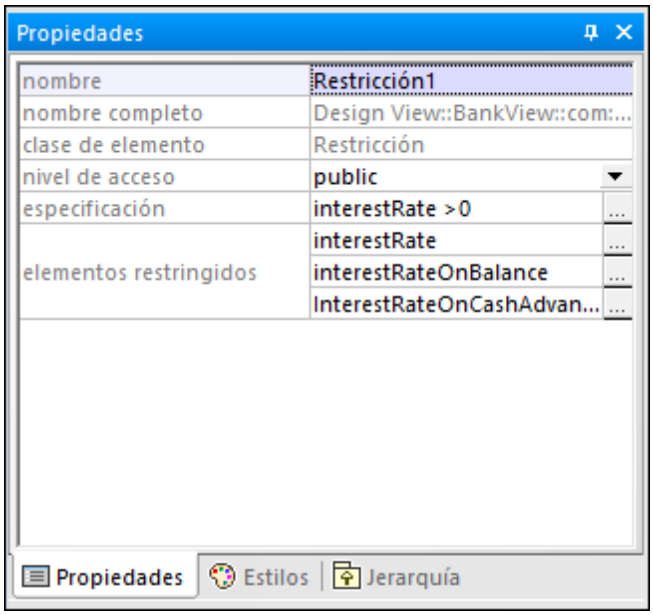


Para asignar restricciones a múltiples elementos de modelado:

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`.

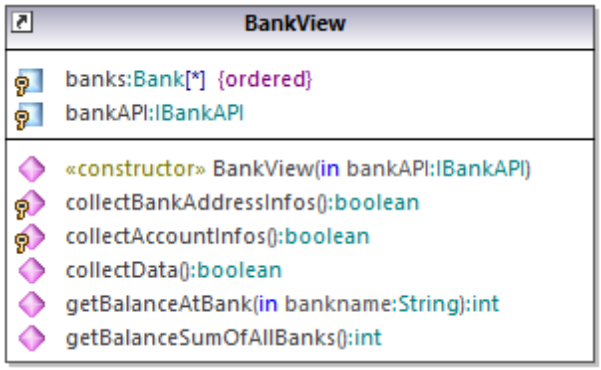
6.1.8 Agregar hipervínculos a elementos

Puede crear manualmente hipervínculos entre la mayoría de los elementos de modelado (excepto las líneas) y muchos 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.

 © 2018 Altova GmbH



Hipervínculos que contienen clases

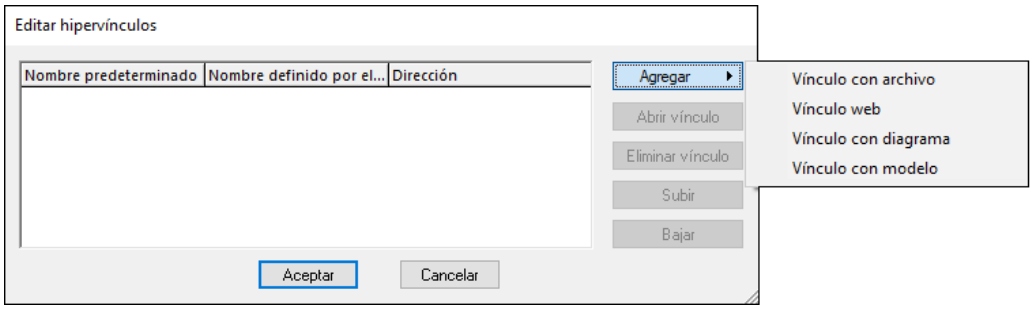
Nota: cuando navegue por la interfaz gráfica del usuario de UModel, con o sin hipervínculos, puede moverse fácilmente haciendo clic en los respectivos botones de la barra de herramientas **Atrás** o **Adelante** .

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 binarios Java, C# y VB.NET. 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:

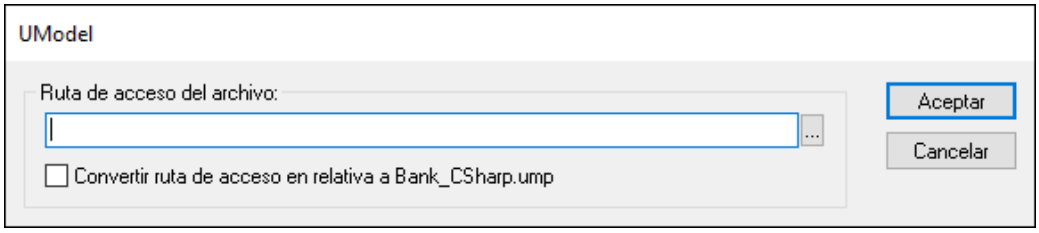
1. Haga clic con el botón derecho en un elemento de un diagrama o en la ventana *Estructura del modelo* y seleccione **Hipervínculos | Insertar o editar hipervínculos** del menú contextual.
2. Haga clic en **Agregar** y seleccione un tipo de hipervínculo (vínculo con archivo, vínculo web, vínculo con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



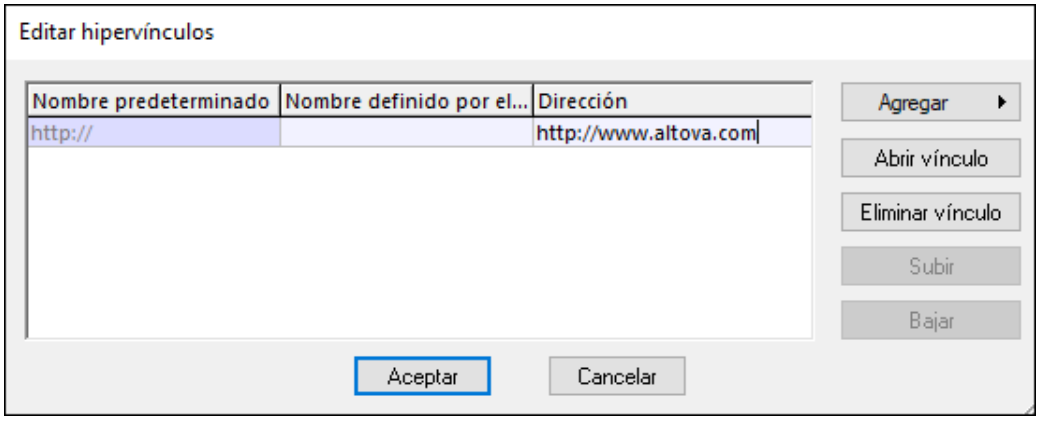
3. Escoja una de estas opciones:
   - Para crear un diagrama o hipervínculo, seleccione el elemento destino o diagrama

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.



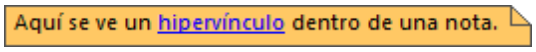
- Para crear un hipervínculo web, teclee la dirección de destino en la columna "Dirección" el cuadro de diálogo, por ejemplo:



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".

6.1.9 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*).

 © 2018 Altova GmbH

- Introduzca texto en la ventana *Documentación*.

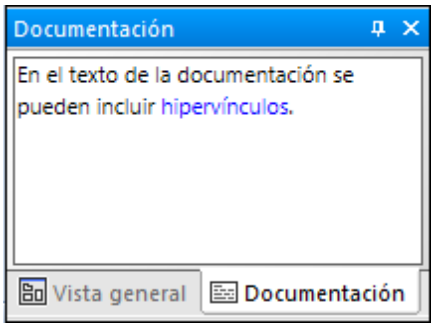
Cualquier texto de documentación se guardará junto con el proyecto.

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.

Nota: al importar esquemas XML, en la ventana *Documentación* solo se muestra la primera anotación de tipo complejo o simple.

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

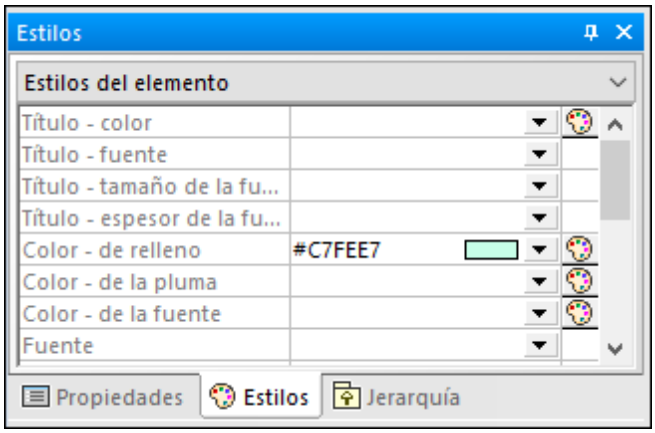
6.1.10 Cambiar el estilo de los elementos de un diagrama

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 ver un ejemplo paso a paso, consulte Asignar iconos de estereotipo personalizados.

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.



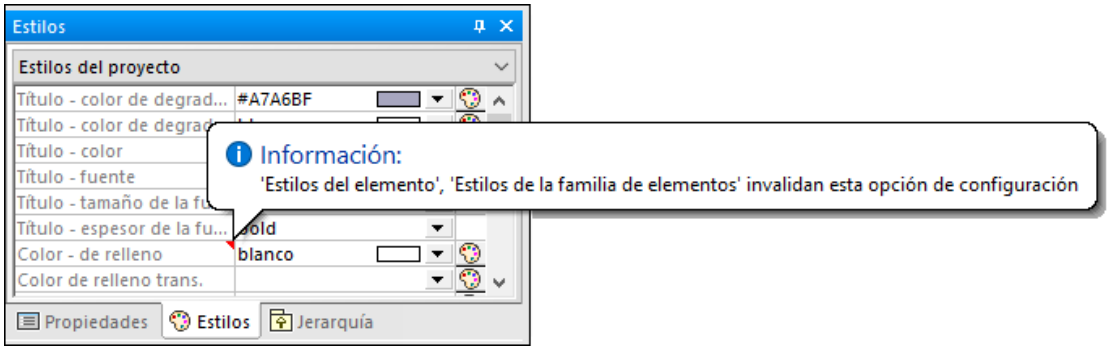
   b. Para editar las propiedades de todos los elementos del mismo tipo (por ejemplo, clases), seleccione "Estilos de la familia de elementos" de la lista.
   c. Para editar las propiedades de todos los elementos a nivel de proyecto, seleccione "Estilos del proyecto".
   d. Para editar las propiedades de 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.)

 © 2018 Altova GmbH

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

6.2 Diagramas

6.2.1 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 ya existente 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 clases
	Diagrama de comunicación
	Diagrama de componentes
	Diagrama de estructura compuesta
	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

© 2018 Altova GmbH

Icono	Descripción
	Diagrama de máquina de estados
	Diagrama de ciclo de vida
	Diagrama de casos de uso
	Diagrama de esquema XML

6.2.2 Generar diagramas

Además de crear diagramas desde cero, también puede generar automáticamente ciertos diagramas a partir de elementos de modelado ya existentes o desde código de programa. En esta sección explicamos cómo generar diagramas a partir de elementos de modelado ya existentes. 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 ya existentes 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:

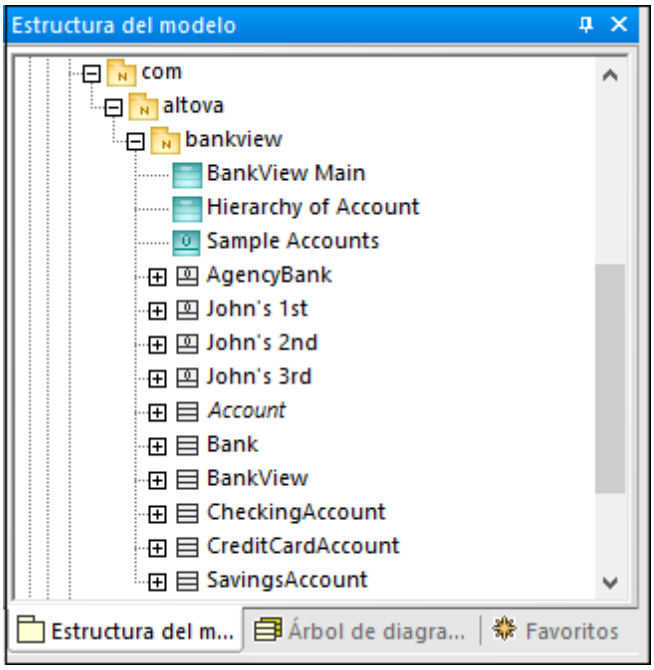
- Haga clic con el botón derecho en un paquete de la Estructura del modelo y seleccione **Mostrar en un diagrama nuevo | Contenido** en el menú contextual.

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

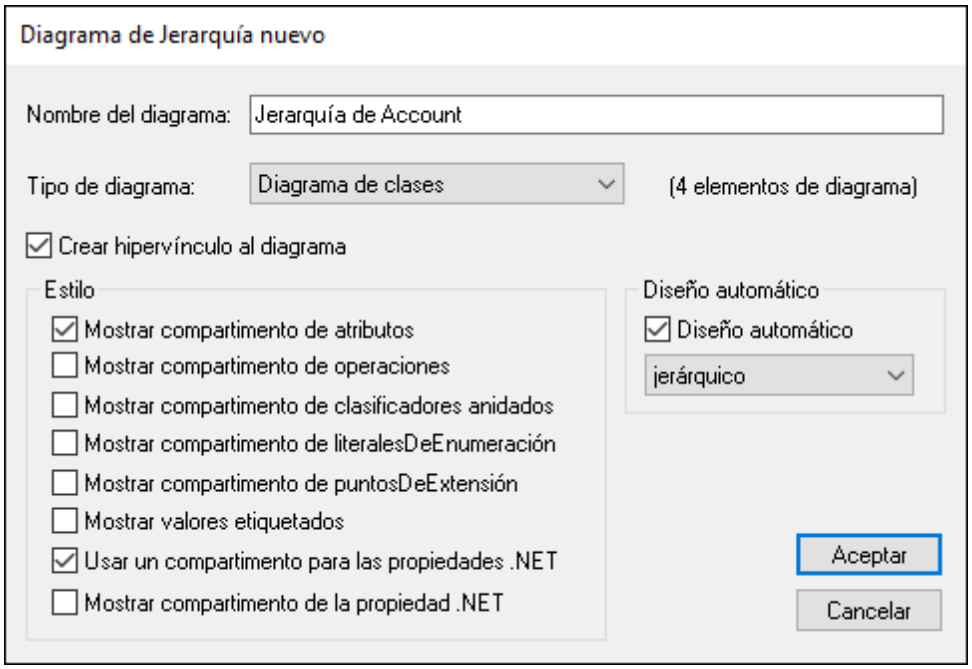
- Haga clic con el botón derecho en un paquete de la ventana *Estructura del modelo y* seleccione **Mostrar en un diagrama nuevo | Dependencias entre paquetes** en el menú contextual.

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 \UModel2018\UModelExamples\Bank_CSharp.ump**).

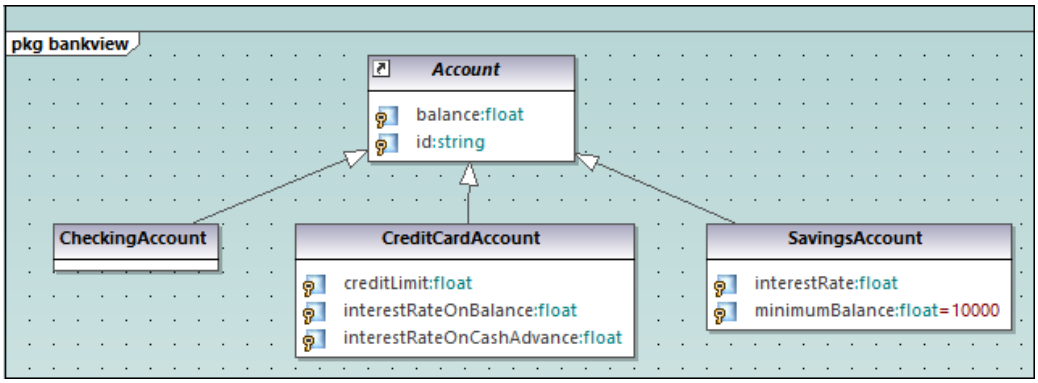


2. Seleccione **Mostrar en un diagrama nuevo | Jerarquía de la generalización** en el menú contextual. Aparecerá un cuadro de diálogo en el que, entre otras preferencias para el diagrama que va a crear, podrá ajustar el tipo de diagrama. El texto "N elementos de diagrama", que muestra el número de elementos 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.



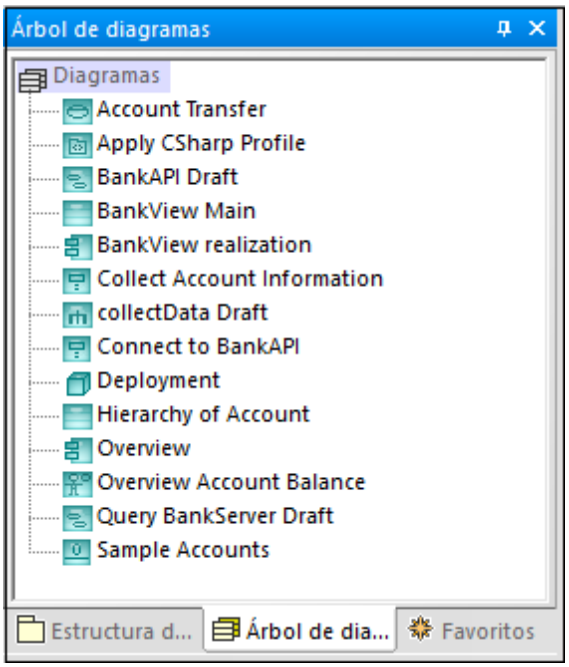
3. Haga clic en **Aceptar**. El diagrama se genera conforme a las opciones seleccionadas y se abre en la Ventana de diagramas, por ejemplo:

 © 2018 Altova GmbH



6.2.3 Abrir diagramas

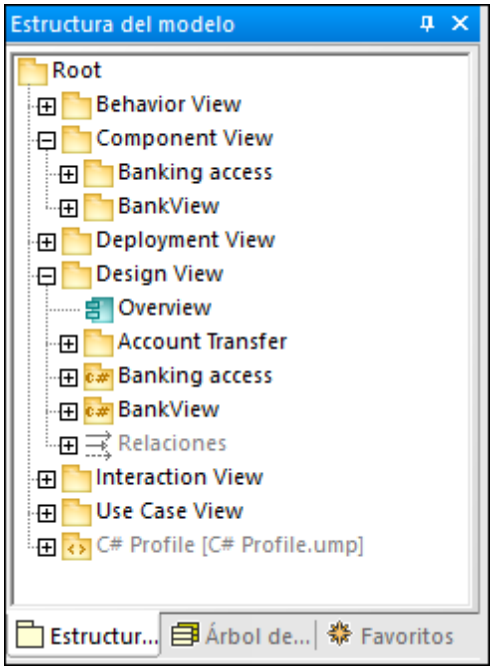
Si el proyecto de UModel contiene diagramas, estos se muestran en el Árbol de diagramas.



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 ya existente:

- 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.

6.2.4 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.

© 2018 Altova GmbH

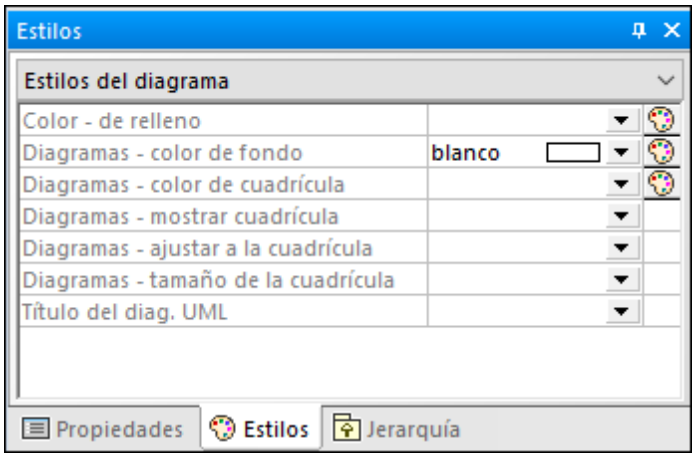
6.2.5 Cambiar el estilo de diagramas

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:

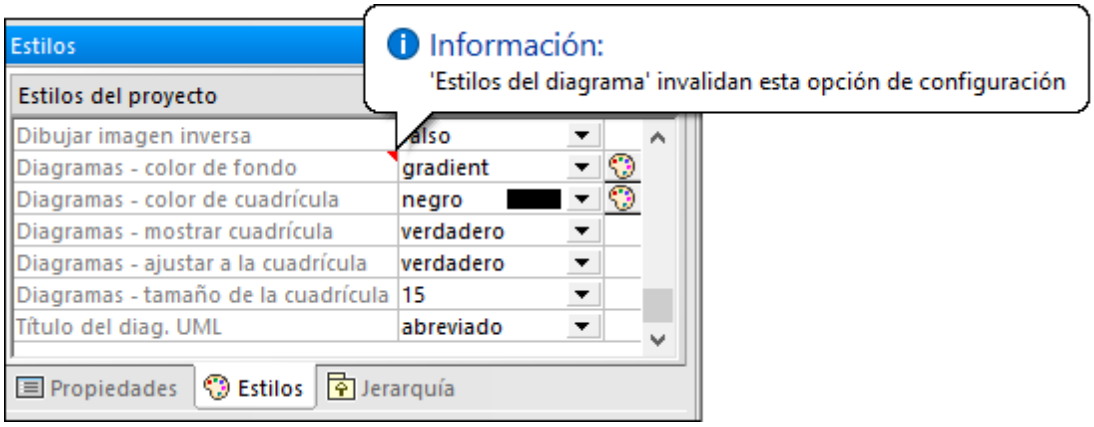
1. Abra un diagrama (*véase Abrir diagramas*).
2. En la parte superior de la ventana *Estilos* hay una lista desplegable que ofrece dos opciones:
   a. Para editar únicamente las propiedades del diagrama actual, seleccione en esa lista "Estilos del diagrama". Este es el valor predeterminado si hace clic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

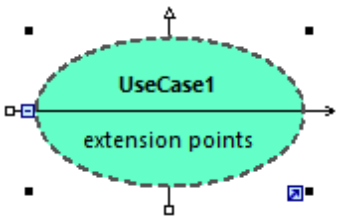
Las siguientes propiedades específicas de diagramas están disponibles como botones en la barra de herramientas. Cambiar la propiedad en la ventana *Estilos* actualizará el estado del botón de la barra de herramientas y viceversa.

	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.

6.2.6 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.

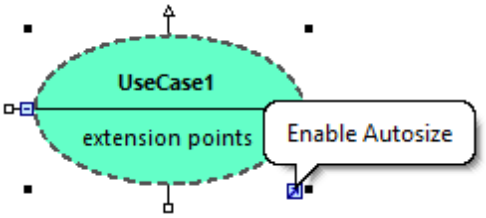


2. Arrastre cualquiera de los puntos negros en la dirección en que quiere agrandar el elemento.

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

 © 2018 Altova GmbH

- 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 **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

Icono	Comando	Notas
		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.
	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.

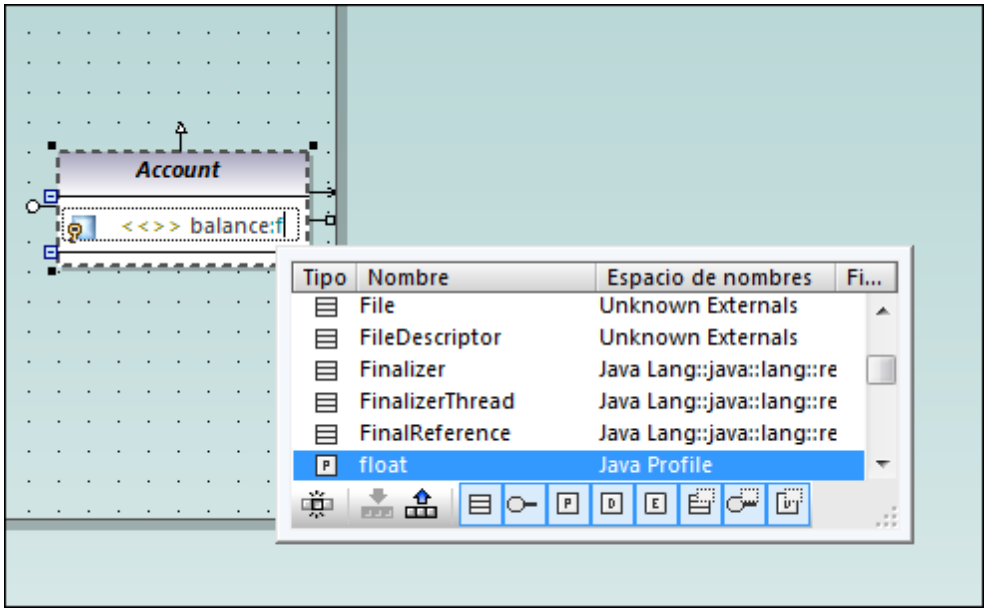
6.2.7 Finalización automática en clases

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 (:).

 © 2018 Altova GmbH

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**.

6.2.8 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**).

© 2018 Altova GmbH

6.3 Relaciones

6.3.1 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:

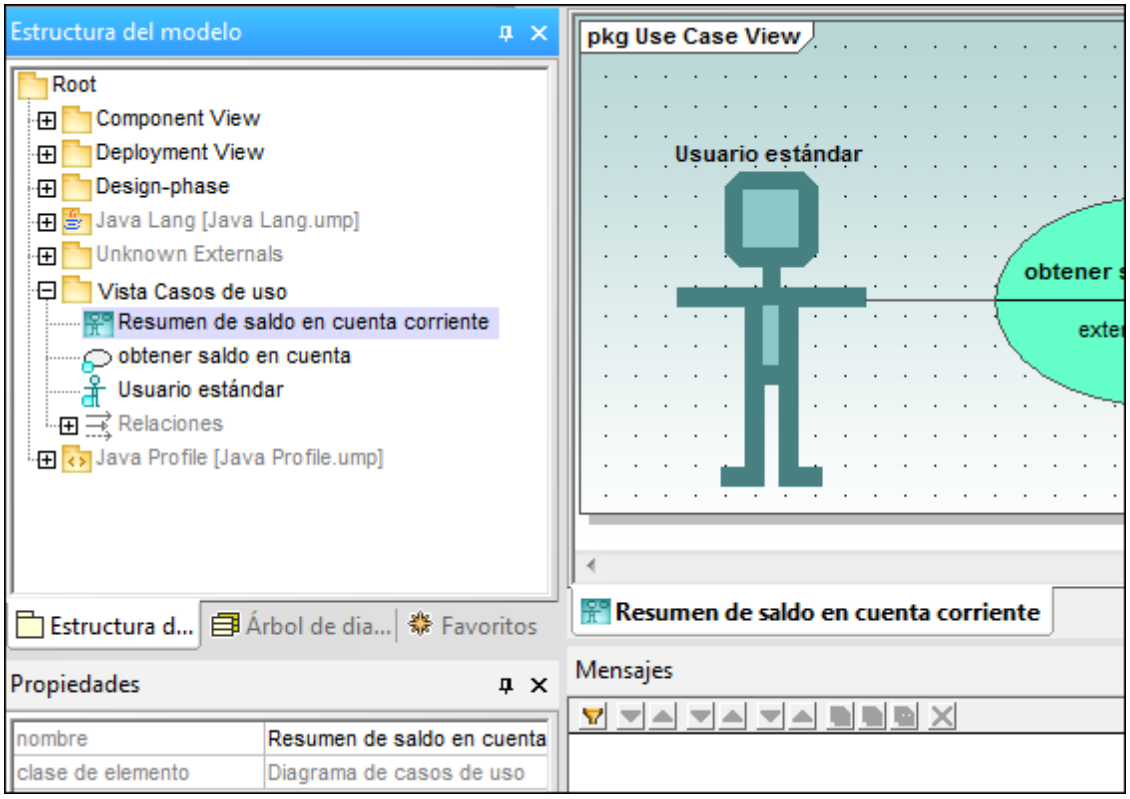
1. Usando el botón de la barra de herramientas que representa la relación que necesita (por ejemplo, asociación *).*
2. Usando los controladores que aparecen al hacer clic en cualquiera de los elementos del diagrama.

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

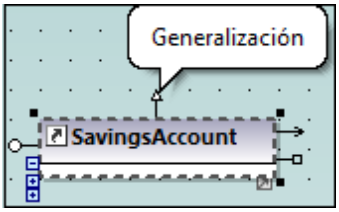
Crear relaciones usando controles

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:

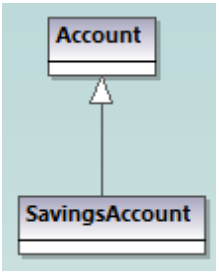
- RealizaciónDeInterfaz
- Generalización
- Asociación
- Asociación de colección

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.

 © 2018 Altova GmbH



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

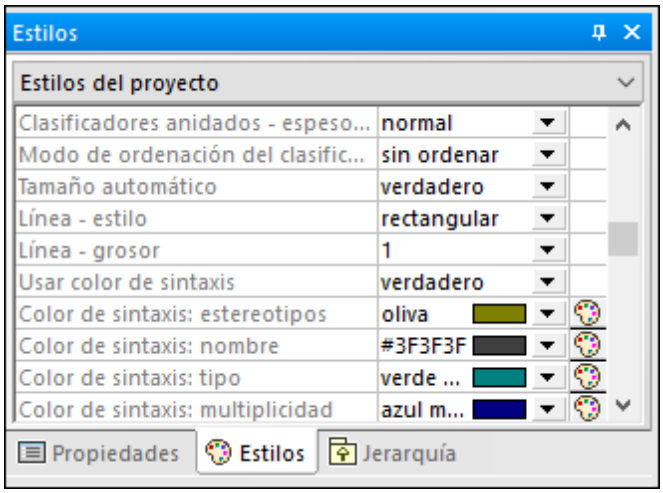
6.3.2 Cambiar el estilo de las líneas y relaciones

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 ya existentes 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.

 © 2018 Altova GmbH

Para mover la etiqueta de texto:

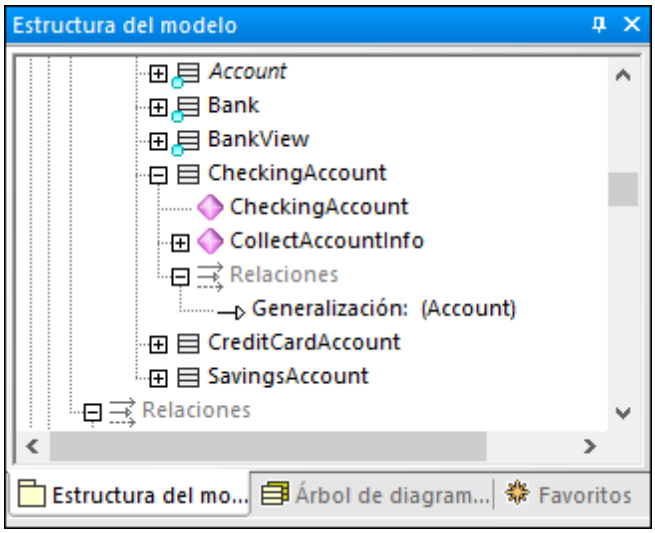
- Haga clic en la etiqueta y arrástrela a su nueva posición en el diagrama.
- Para devolver la etiqueta a su posición original haga clic con el botón derecho en la relación y seleccione **Etiquetas de texto | Ajustar la posición de las etiquetas de texto** en el menú contextual.
- Para recolocar varias 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**.

6.3.3 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.

6.3.4 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:

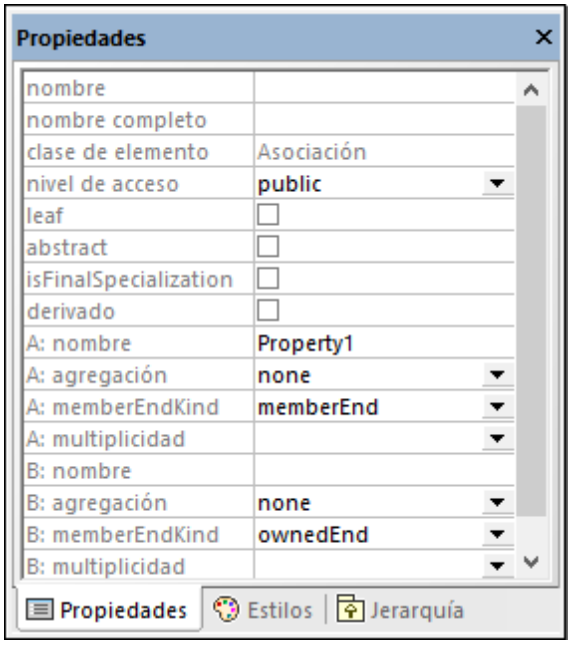
- La opción Crear asociaciones automáticamente se habilita desde **Herramientas | Opciones | Edición de diagramas**.
- Se determina el tipo de atributo (en la imagen anterior, `Propiedad1` es de tipo `Motor`)
- La clase del "tipo" referenciado también está presente en el diagrama actual (en la imagen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

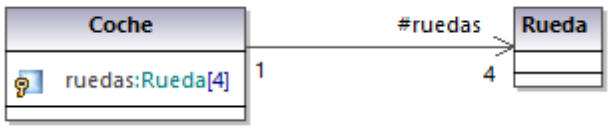
Al hacer clic en una asociación del diagrama, sus propiedades se pueden modificar, si es necesario, desde la ventana Propiedades.

© 2018 Altova GmbH



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
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

Propiedad	Finalidad
	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. 

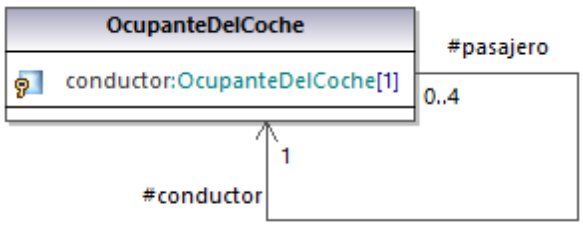
Existen los mismos atributos disponibles para el extremo B de la relación.

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.

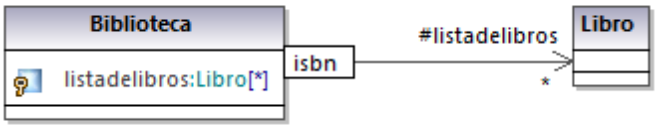


 © 2018 Altova GmbH

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:

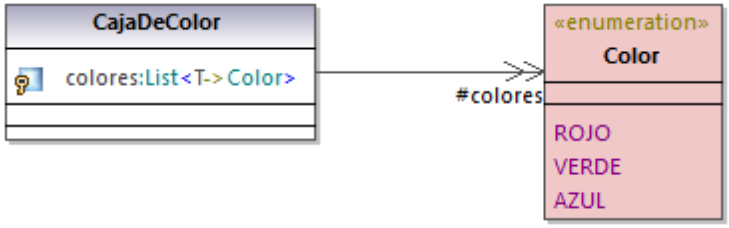
1. Cree una asociación entre dos clases.
2. Haga clic con el botón derecho en la asociación y seleccione **Nuevo/a | Calificador**.



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).

6.3.5 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.



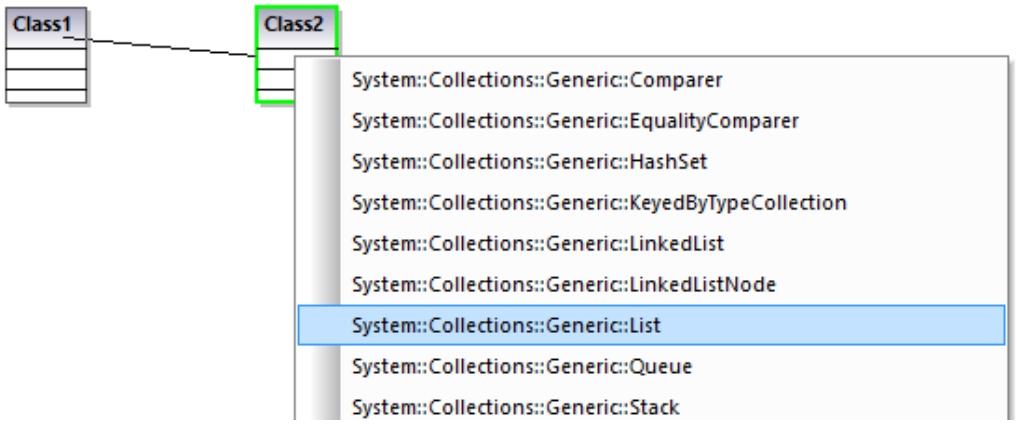
Para que pueda crear asociaciones de colecciones, el proyecto de UModel primero tiene que contener las plantillas de colección para el lenguaje de proyecto que quiera usar (como Java, C#, o VB.NET). De lo contrario aparecerá el texto "No se definieron colecciones para este lenguaje" cuando intente crear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

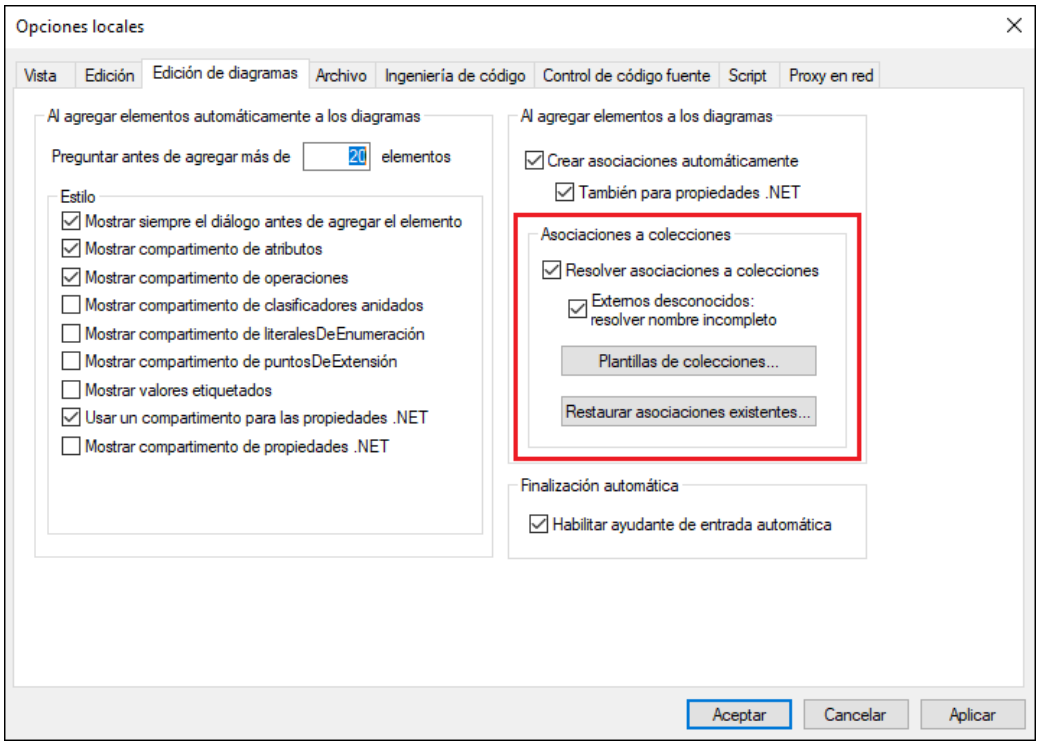


Asociación de colecciones e ingeniería de código

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:

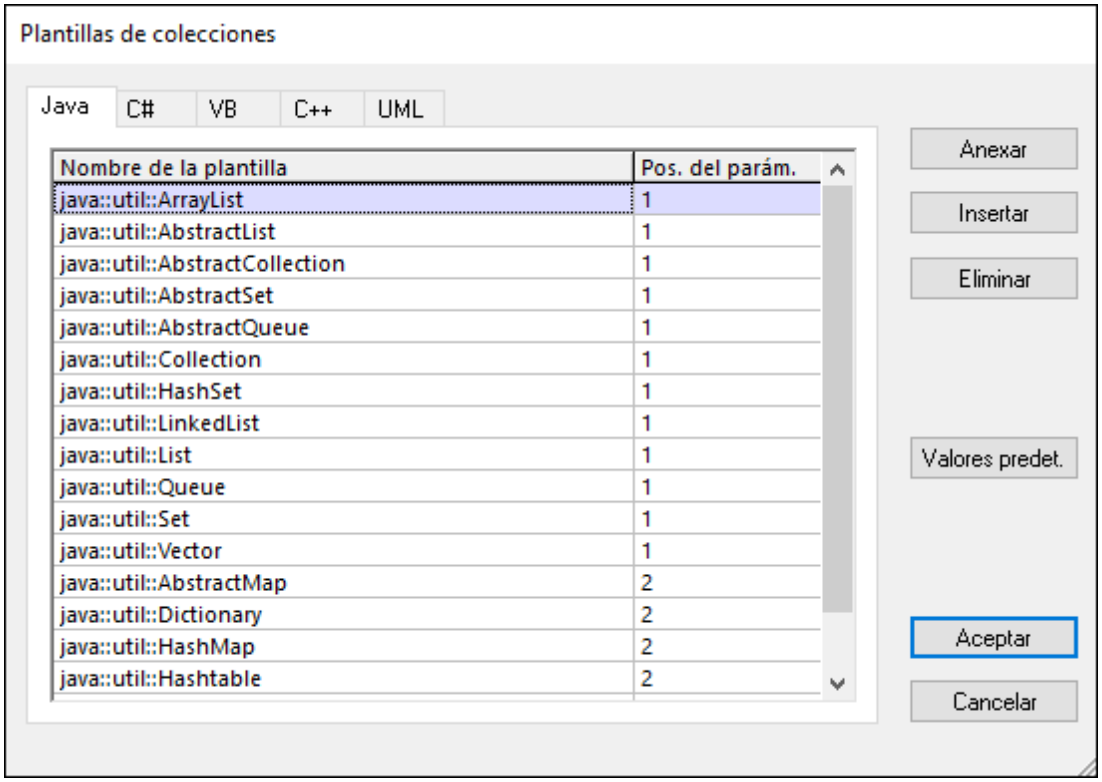
1. En el menú **Herramientas** haga clic en **Opciones**.
2. Haga clic en la pestaña **Edición de diagramas**.
3. Marque o desmarque, según lo que necesite, la casilla **Resolver asociaciones a colecciones**.

 © 2018 Altova GmbH



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

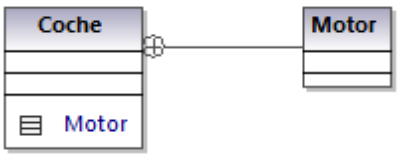
Para restaurar las plantillas de colecciones a sus valores originales haga clic en **Valores predet**.

6.3.6 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.

 © 2018 Altova GmbH

Altova UModel 2018

Proyectos e ingeniería de código

7 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 en 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# o 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 binarios Java, C# y VB.NET*).

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.

© 2018 Altova GmbH

7.1 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*.

7.1.1 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.

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. Además al proyecto se 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, 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. Cuando 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 externamente en el archivo de proyecto .ump o en los archivos que incluye y emite un aviso solicitando la recarga del proyecto. Esta 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 pretty-print cuando se guarda (esto se habilita en **Herramientas | Opciones | Archivo**).

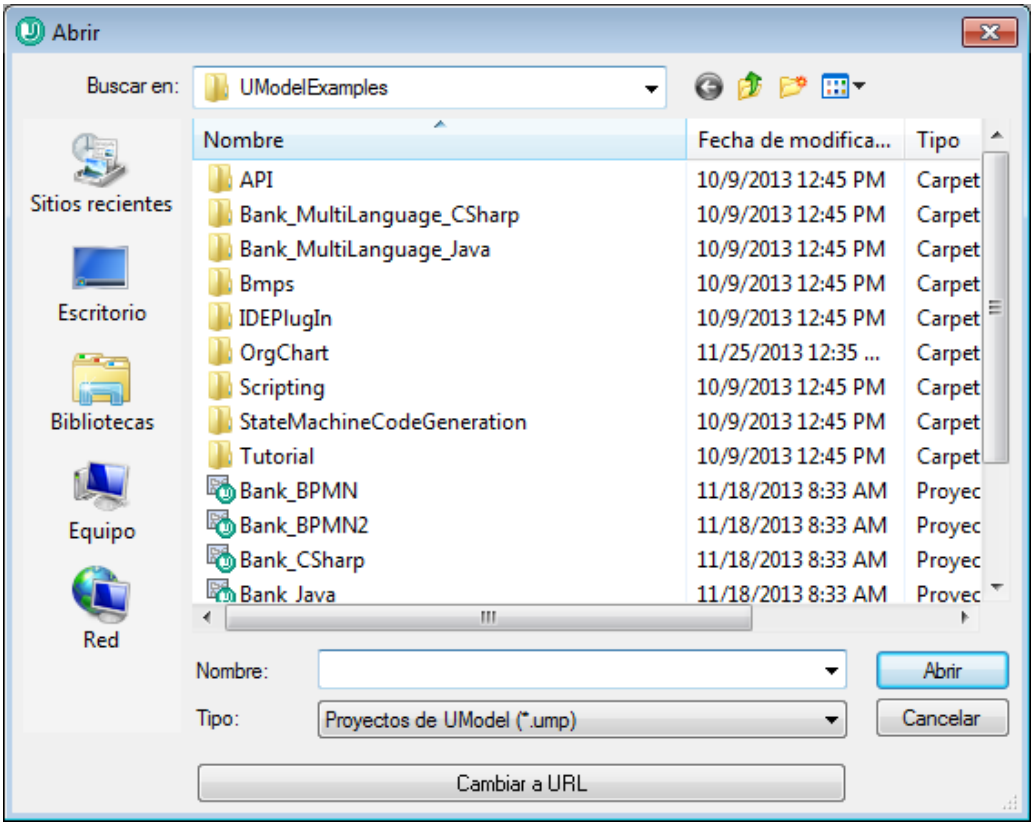
© 2018 Altova GmbH

7.1.2 Abrir proyectos desde una URL

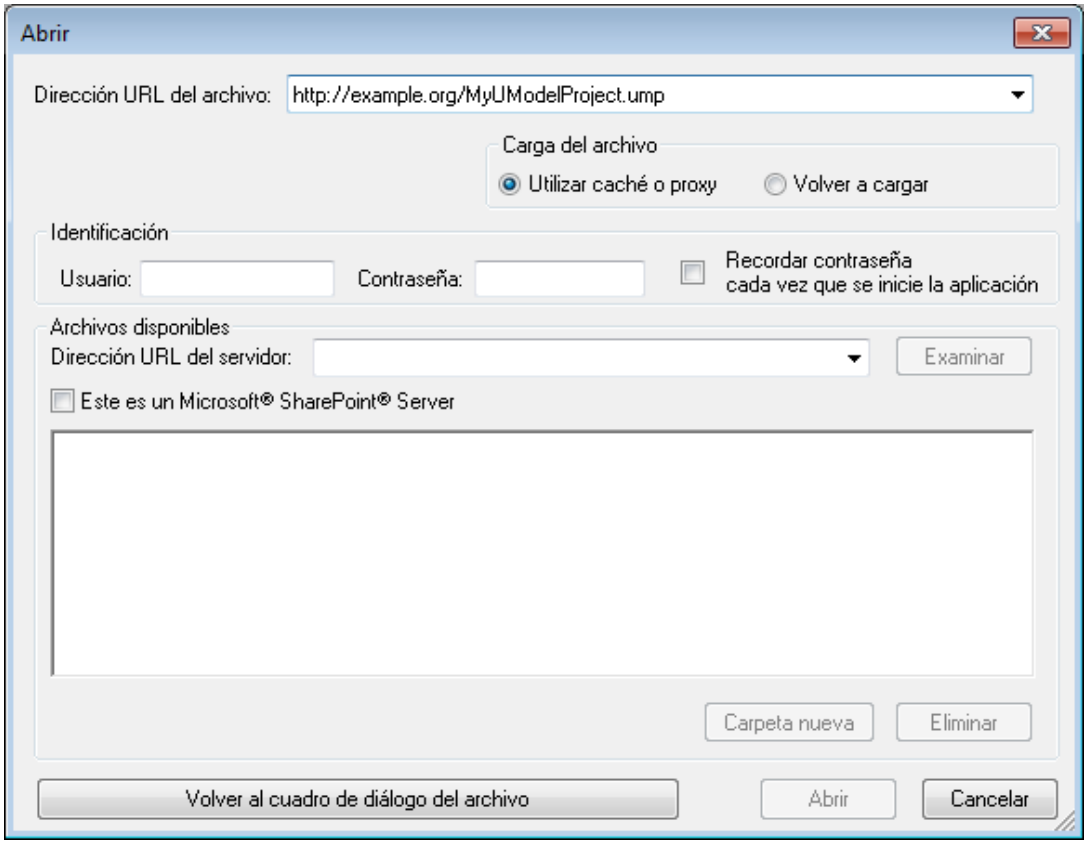
Además de abrir archivos de proyecto locales, también puede abrir archivos desde direcciones URL. Los protocolos compatibles son HTTP, HTTPS y FTP. No obstante, recuerde que los archivos que se cargaran desde direcciones URL no se pueden volver a guardar en su ubicación original (en otras palabras, el 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*.

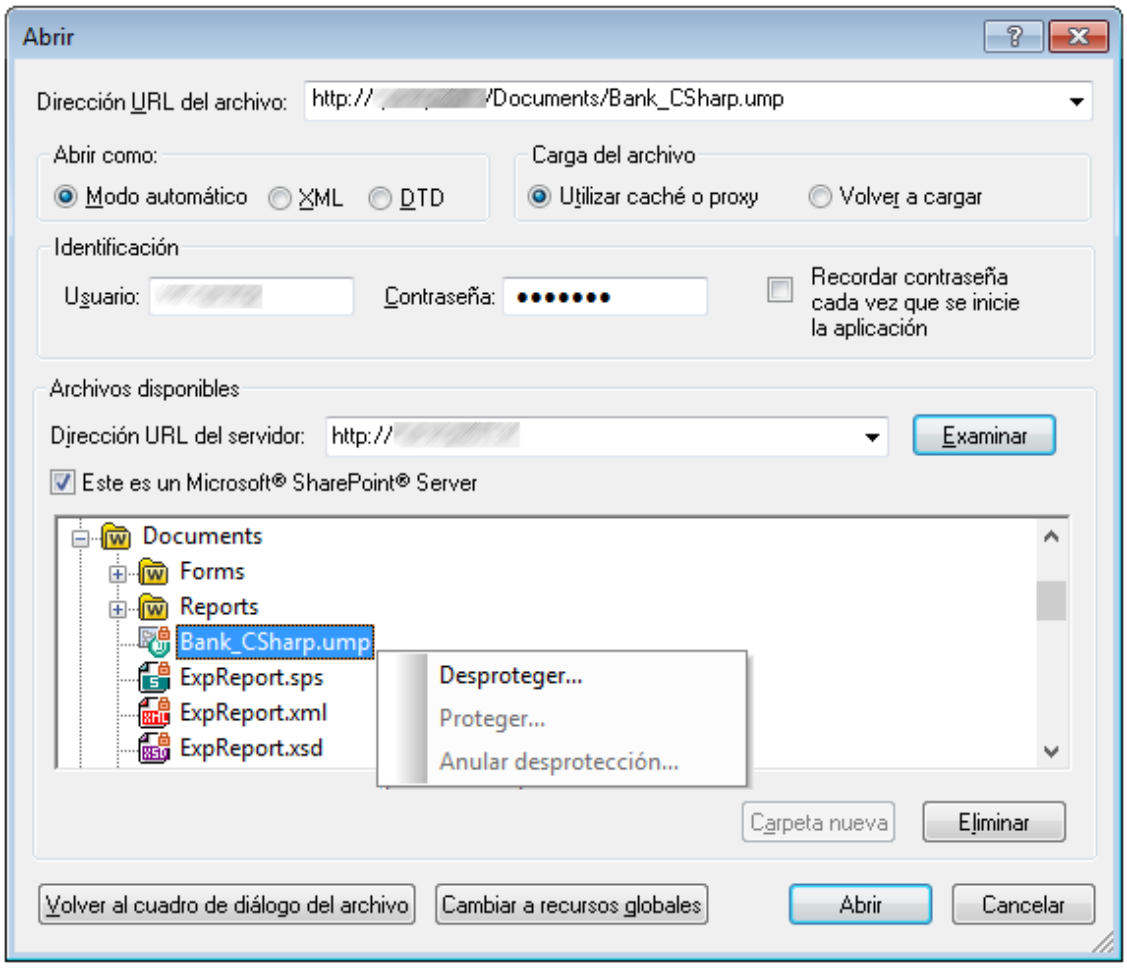
Si es improbable que el archivo elegido cambie en el futuro, seleccione la opción *Utilizar caché o proxy* para almacenar los datos en caché y acelerar la carga del archivo. De lo contrario, si desea que el archivo se vuelva a cargar cada vez que se abra UModel, entonces seleccione *Volver a cargar*.

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.

 © 2018 Altova GmbH



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).

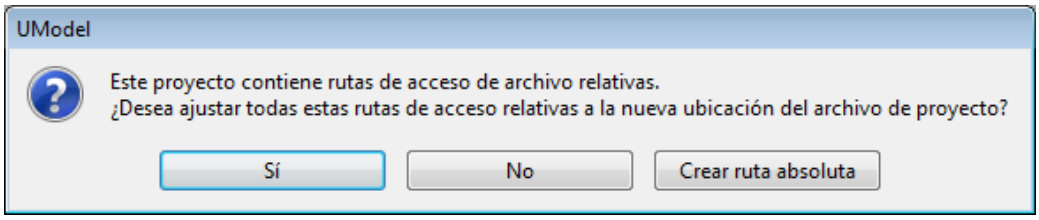
Es importante tener en cuenta que:

- 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.

7.1.3 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:

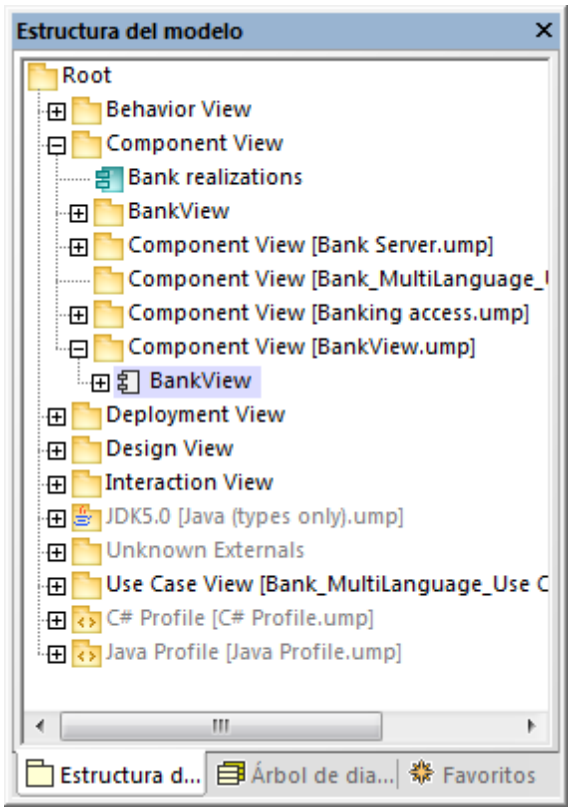
- Seleccionando el comando de menú **Archivo | Guardar como...** y haciendo clic en **Sí** cuando la aplicación pregunte si las rutas de acceso de los archivos se deben ajustar a la nueva ubicación del proyecto.



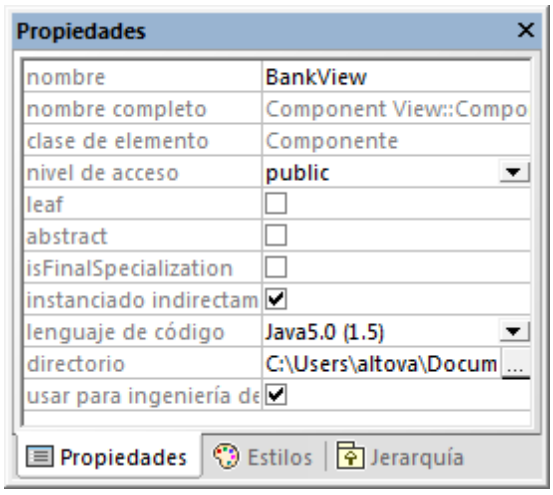
- Copiando el proyecto de UModel (*.ump) a una nueva ubicación y ajustando más tarde las rutas de generación de código de todos los componentes que participen en la generación de código.

El ejemplo de proyecto **C:\Usuarios\<usuario>\Documentos\Altova\UModel2018\UModelExamplesBank_Multilanguage.ump** permite experimentar con este último método.

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



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

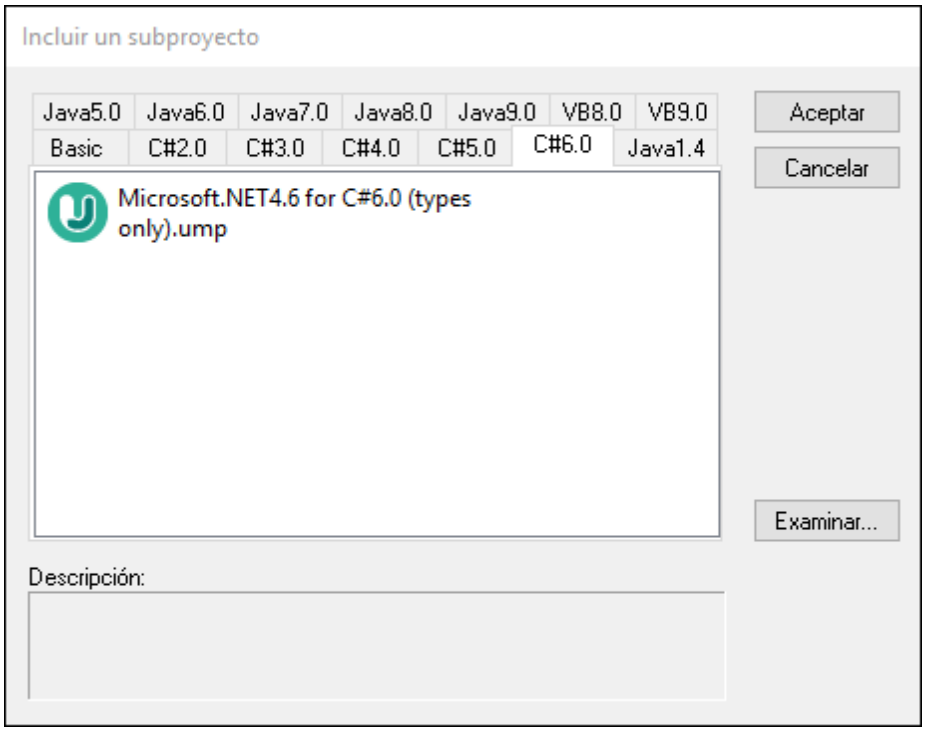


3. Vuelva a sincronizar el modelo y el código.

7.1.4 Incluir otros proyectos de UModel

Cuando necesite generar código a partir de un modelo o importar código fuente en 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, UModel viene con varios subproyectos (perfiles de lenguaje necesarios para las funciones de ingeniería de código).



Cuadro de diálogo "Incluir un subproyecto"

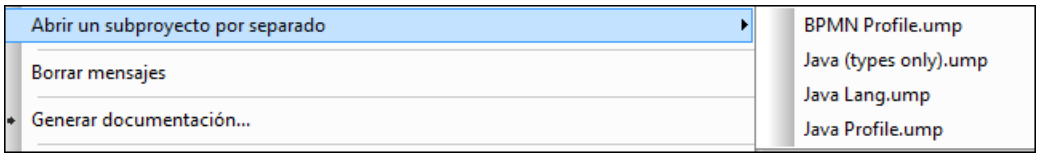
Las pestañas y los proyectos de UModel (archivos `.ump`) que aparecen en el cuadro de diálogo "Incluir un subproyecto" se pueden configurar. UModel obtiene esta información de esta ruta de acceso relativa a la carpeta "`Archivos de programa`" del sistema: `\Altova\UModel2018\UModelInclude`. Recuerde que los archivos de proyecto que aparecen en la pestaña *Basic* existen en la carpeta `UModelInclude` directamente, mientras que los proyectos de las pestañas *Java*, *VB* y *C#* existen en las correspondientes subcarpetas de la carpeta `UModelInclude`.

UModel viene con varios perfiles de proyecto (archivos `.ump`) que pueden ser de gran utilidad para sus proyectos. Por ejemplo, en las pestañas *Java* encontrará paquetes, interfaces y clases del lenguaje Java. Asimismo, en las pestañas *C#* y *VB* encontrará archivos .ump para estos lenguajes.

Para ver todos los proyectos que se importaron al proyecto:

- Seleccione el comando de menú **Proyecto | Abrir un subproyecto por separado**. El menú desplegable enumera todos los subproyectos que forman parte del proyecto.

 © 2018 Altova GmbH



Para crear una pestaña nueva en el cuadro de diálogo "Incluir un subproyecto":

- Navegue hasta la carpeta `\Altova\UModel2018\UModelInclude` (relativa a su carpeta `Archivos de programa`) y cree la carpeta correspondiente a la pestaña que desea crear en el cuadro de diálogo (p. ej. `\UModelInclude\micarpeta`). El nombre que le asigne a la carpeta será el nombre que toma la pestaña en el cuadro de diálogo "Incluir un subproyecto".
- Copie los archivos `.ump` pertinentes en la carpeta que acaba de crear para que estén disponibles en el cuadro de diálogo "Incluir un subproyecto".

Para crear una descripción para cada archivo de proyecto de UModel:

- Cree un archivo de texto cuyo nombre debe ser el mismo que el del archivo `.ump` y guárdelo en la misma carpeta. Por ejemplo, el archivo `MiModelo.ump` necesita un archivo de texto llamado `MiModelo.txt`. **Nota:** la codificación del archivo debe ser **UTF-8**.

Para eliminar un subproyecto incluido en el proyecto:

1. Haga clic en el subproyecto incluido en la ventana *Estructura del modelo* y pulse la tecla **Supr**.
2. Aparece un aviso preguntando si desea continuar con el proceso de eliminación.
3. Haga clic en **Aceptar** para eliminar el archivo incluido del proyecto.

Para eliminar o quitar un proyecto del cuadro de diálogo "Incluir un subproyecto":

- Elimine o quite el archivo `.ump` que desea eliminar de la carpeta correspondiente en el sistema de archivos.

7.1.5 Compartir paquetes y diagramas

En UModel puede compartir paquetes y diagramas UML con otros proyectos. A su vez, los paquetes se pueden incluir en otros proyectos de UModel mediante referencia o como copia.

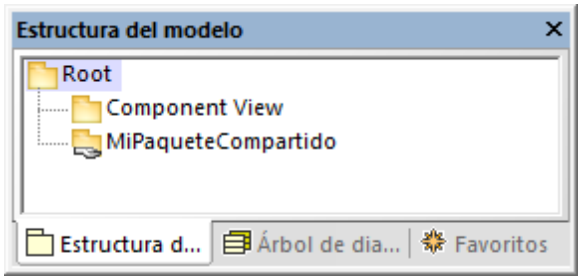
Además los subproyectos se pueden separar del proyecto principal en cualquier momento e incluirse otra vez en el proyecto principal como subproyectos editables o de solo lectura. Asimismo, los paquetes se comparten y se guardan como archivos de subproyecto. Los subproyectos también se pueden añadir al 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.

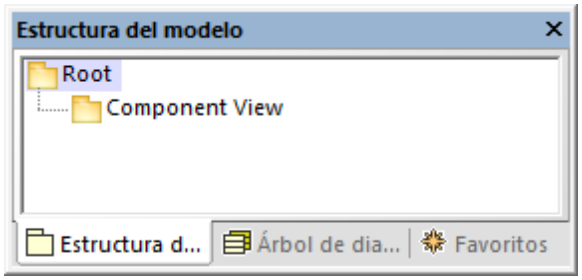
Para compartir un paquete con otros proyectos:

1. En la *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.



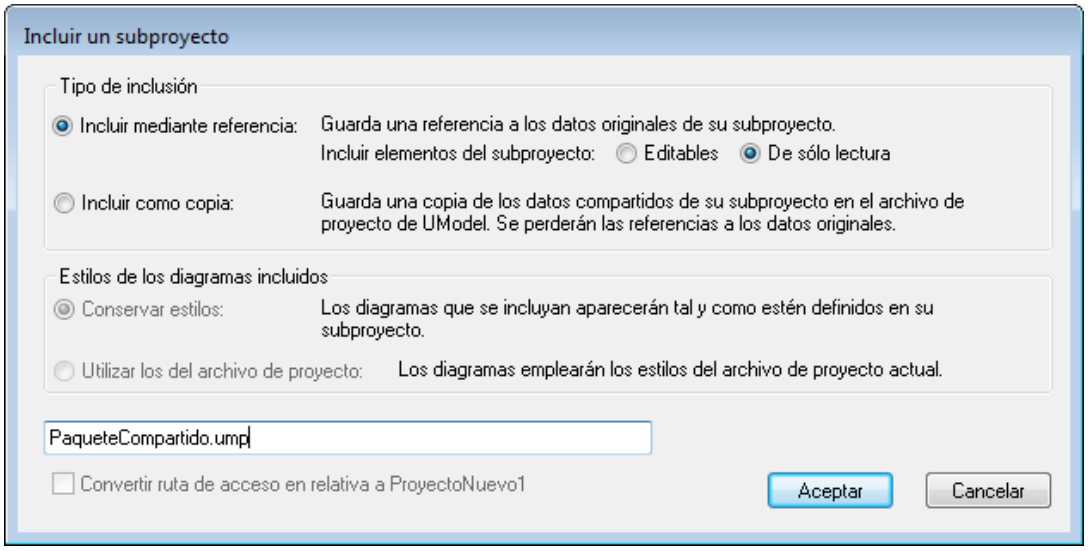
Para incluir/importar una carpeta compartida en un proyecto:

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



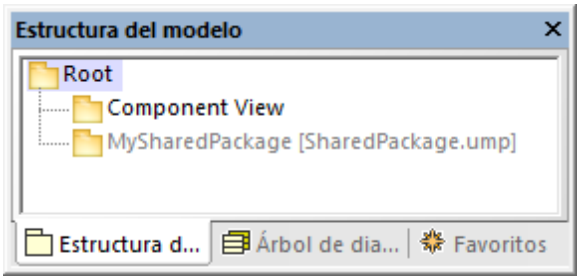
2. Seleccione la opción de menú **Proyecto | Incluir un subproyecto...**.
3. Haga clic en el botón **Examinar**, seleccione el proyecto que incluye el paquete compartido y haga clic en **Abrir**. Aparece el cuadro de diálogo "Incluir un subproyecto", donde puede elegir si el paquete/proyecto se incluye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

© 2018 Altova GmbH



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**).

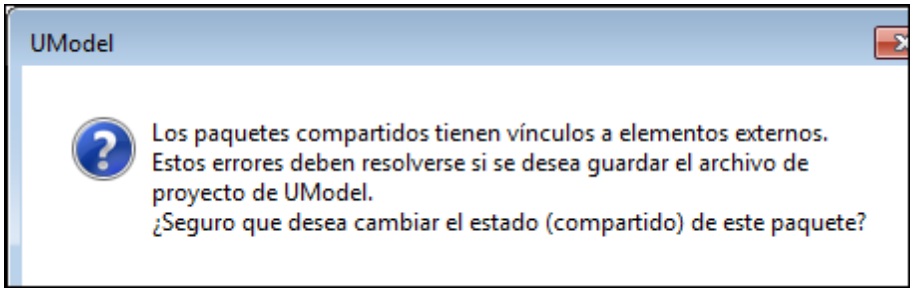


Notas:

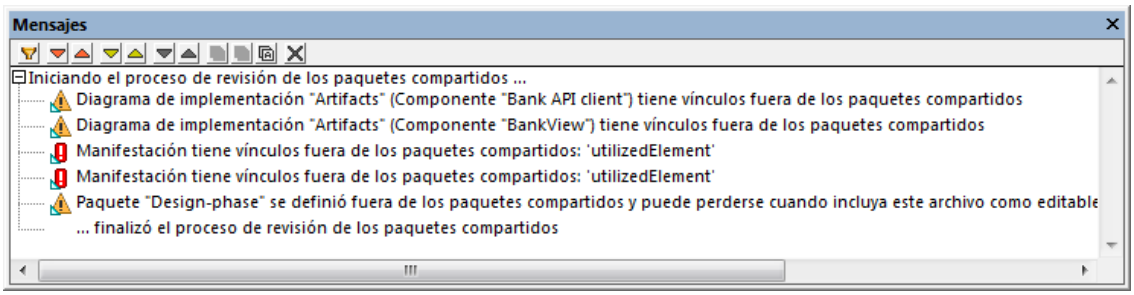
- Cuando incluya un proyecto de código fuente que contiene subproyectos, todos los subproyectos del proyecto de código fuente se incluirán también en el proyecto de destino.
- Las carpetas compartidas que se incluyeron mediante referencia se pueden cambiar a *Incluir como copia* en cualquier momento. 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\UModel2018\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*.

7.1.6 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 **false** a la propiedad `usar color de sintaxis`).
- Deshabilite el color de fondo `degradado` para los diagramas. En su lugar utilice 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.*

 © 2018 Altova GmbH

7.2 Generar código de programa

Tras diseñar el modelo de la aplicación en UModel (p. ej. varios diagramas de clases), tendrá la posibilidad de generar un proyecto prototipo que incluya todas las interfaces, clases, operaciones, etc. en el lenguaje que prefiera. UModel permite generar código de programa C#, VB.NET o Java a partir de un modelo, basándose en los elementos UML encontrados 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 (*ver instrucciones más abajo*). La raíz de espacio de nombres puede ser un espacio de nombres 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.
- En el proyecto debe haber un componente que realiza todas las clases o interfaces a partir de las que se debe generar el código.
- El componente debe tener asignada una ubicación física (es decir, un directorio). El código se generará en este directorio.
- En el componente debe estar habilitada la propiedad *usar para 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 .

Además, es recomendable incluir uno de los subproyectos integrados de UModel en el lenguaje (o versión del lenguaje) que tiene pensado usar (*véase Incluir otros proyectos de UModel*). Por ejemplo, si el destino de su aplicación es una versión concreta de Java, C# o VB.NET e incluye el subproyecto correspondiente, podrá usar los tipos de datos pertinentes mientras diseña sus clases UML, interfaces UML, etc.

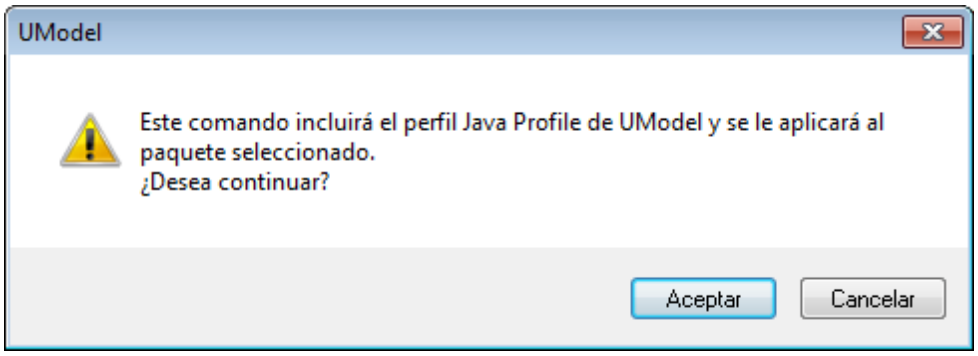
El apartado Ejemplo: generar código Java desde UModel explica cómo se crea un proyecto desde cero y cómo se genera código.

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

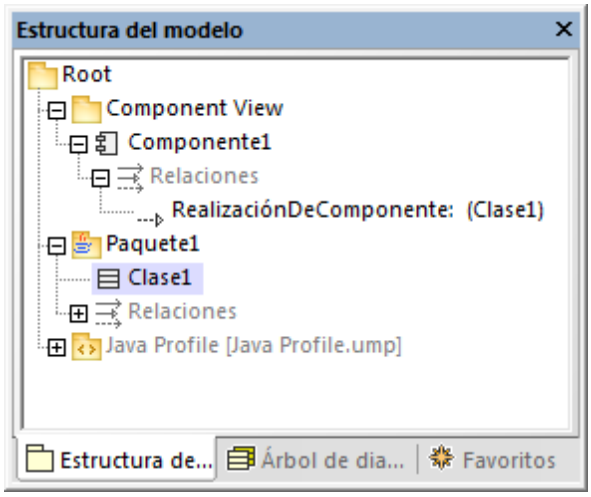
Ingeniería de código | Establecer como raíz de espacio de nombres de <...> en el menú contextual.

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:



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*.



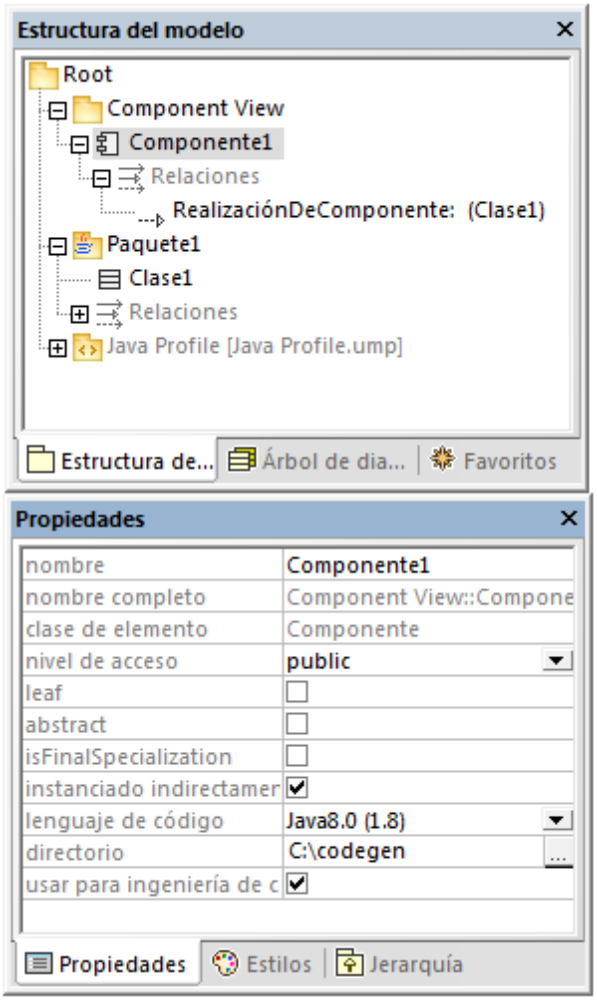
Esto también se puede hacer de otra manera: creando un diagrama de componentes y dibujando la relación `RealizaciónDeComponente` entre el componente y las clases o interfaces. Consulte el apartado Diagramas de componentes para obtener más información.

 © 2018 Altova GmbH

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**:



7.2.1 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 pueden 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 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 una clase o interfaz como mínimo.	Error
Clase	...se definió el nombre del archivo de código. **Nota:** esta comprobación no es relevante cuando se trata de 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 devuelto 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 cuando se trata de clases anidadas.	Advertencia
	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave)	Error

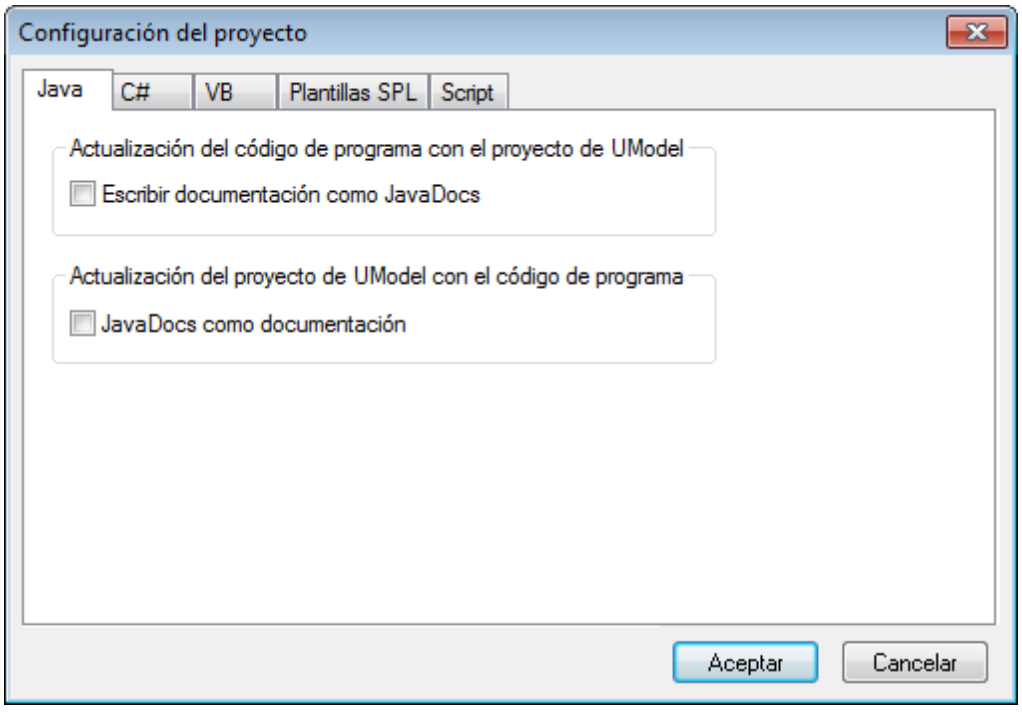
© 2018 Altova GmbH

Nivel	Se comprueba si...	Gravedad del error si no se cumple la condición
	...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 devolución.	Error
Parámetro de operación de clase	...el nombre es válido (no incluye caracteres no válidos y no es una palabra clave)	Error
	...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 de un espacio de nombres del lenguaje.	Error
	...se definió el tipo para las propiedades.	Error
	...se definió el tipo para los parámetros de operación.	Error
	...se definió el tipo devuelto 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)	Error

Nivel	Se comprueba si...	Gravedad del error si no se cumple la condición
	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*.	
Enumeración	...existe una relación `RealizaciónDeComponente` con un componente.	Advertencia

7.2.2 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

 © 2018 Altova GmbH

Pestaña	Opciones
	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.
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*).

Además de las opciones de configuración aquí descritas, hay unas cuantas más 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).

7.2.3 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 las mismas instrucciones.

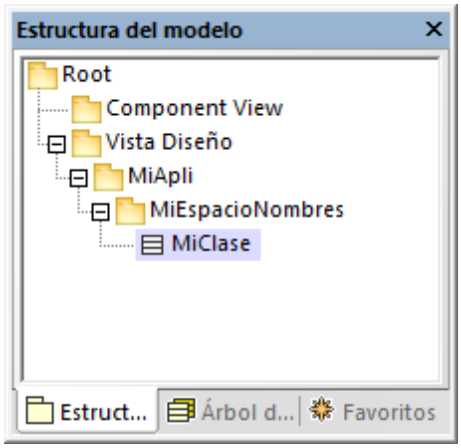
Crear un proyecto de UModel nuevo

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 "Vista Diseño".
2. Haga clic con el botón derecho en "Vista Diseño" y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "MiApli"
3. Haga clic con el botón derecho en "MiApli" y seleccione **Elemento nuevo | Paquete** en el menú contextual. Cambie el nombre del nuevo paquete por "MiEspacioNombres".
4. Haga clic con el botón derecho en "MiEspacioNombres" 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:

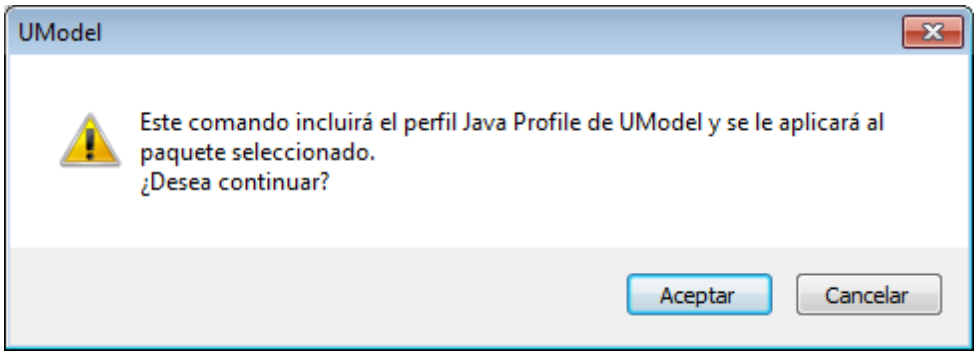
- Debe tener definido un paquete raíz de espacio de nombres Java, C# o VB.NET.
- Debe existir un componente que sea realizado por todas las clases o interfaces para las que se debe generar código.
- El componente debe tener asignada una ubicación física (un directorio). El código se generará en dicha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

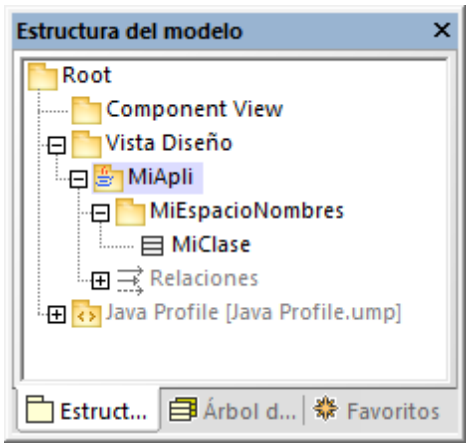
Si se valida el proyecto en este momento, la ventana Mensajes emite el error de validación "`No se encontró la raíz de espacio de nombres. Utilice el menú contextual (submenú "Ingeniería de código") en la estructura del modelo para definir un paquete como raíz de espacio de nombres`." Para resolver este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

1. Haga clic con el botón derecho en el paquete "MiApli" 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**.

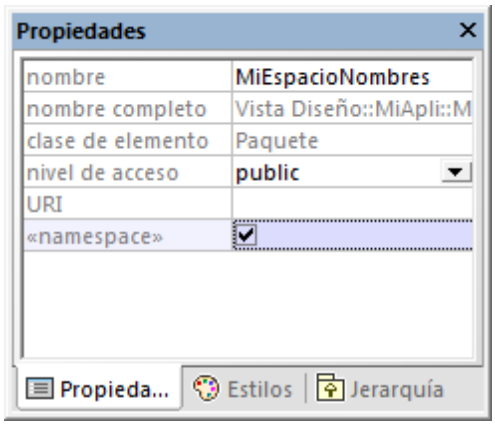


Observe que ahora el paquete tiene un icono distinto ( ) que nos indica que este paquete es una raíz de espacio de nombres de Java. Además, se añadió el perfil Java Profile al proyecto.



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

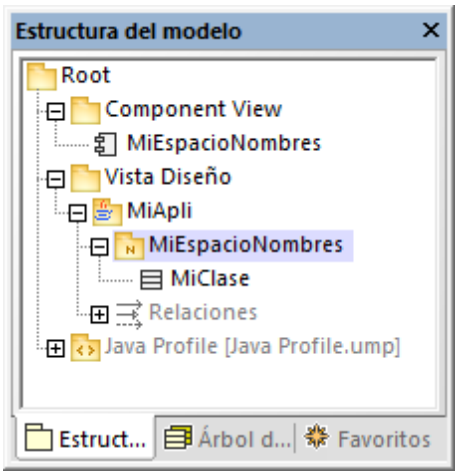
1. Seleccione el paquete "MiEspacioNombres" en la ventana *Estructura del modelo*.
2. En la ventana *Propiedades* habilite la propiedad `<<namespace>>`.



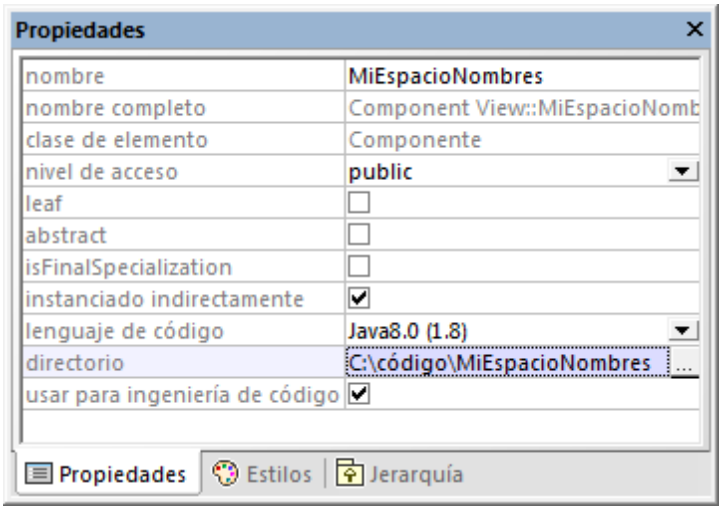
Observe que ahora el paquete tiene un icono distinto ( ) que nos indica que este paquete es un espacio de nombres.

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 "MiEspacioNombres".



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



Nota: el directorio de generación de código también puede ser relativo al proyecto .ump. Por ejemplo, si se usa la ruta de acceso **código\MiEspacioNombres**, el directorio **código**

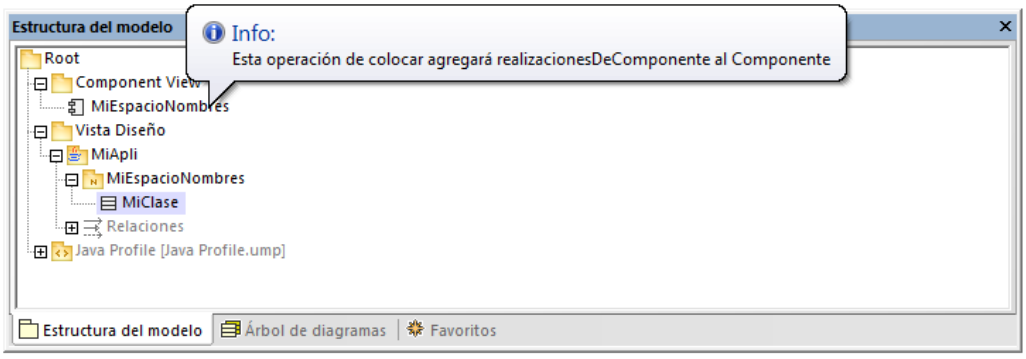
 © 2018 Altova GmbH

se creará en el mismo directorio donde reside 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 "MiEspacioNombres" tiene la ruta de acceso "C:\código\MiEspacioNombres", 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*.

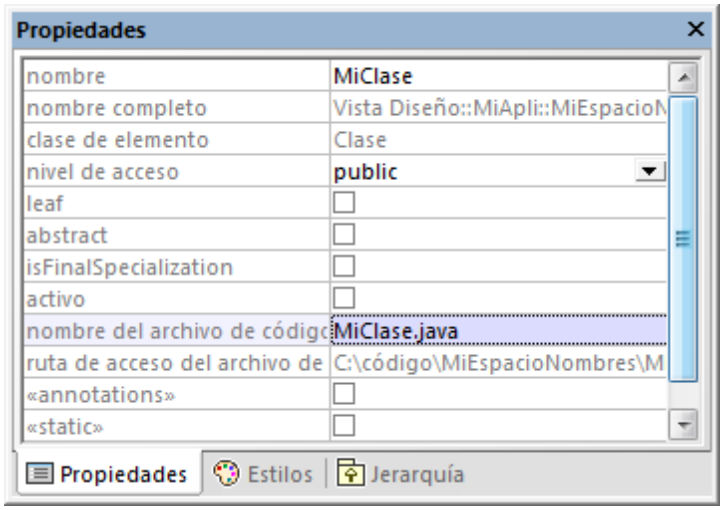
Ahora veamos cómo se puede crear la relación de realización de componente. En la ventana *Estructura del modelo* haga clic en la clase "MiClase" y arrástrela hasta el componente "MiEspacioNombres".



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:

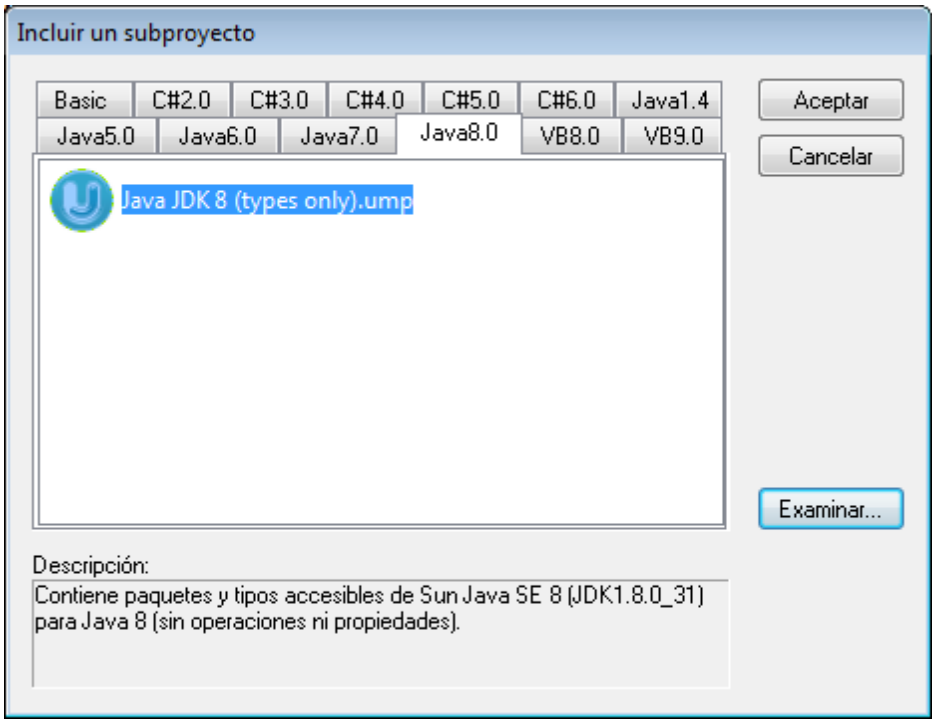
1. Seleccione la clase "MiClase" en la ventana *Estructura del modelo*.
2. En la ventana *Propiedades* cambie el valor de la propiedad `nombre del archivo de código` por el nombre correspondiente (p. ej. `MiClase.java`).



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# y VB.NET):

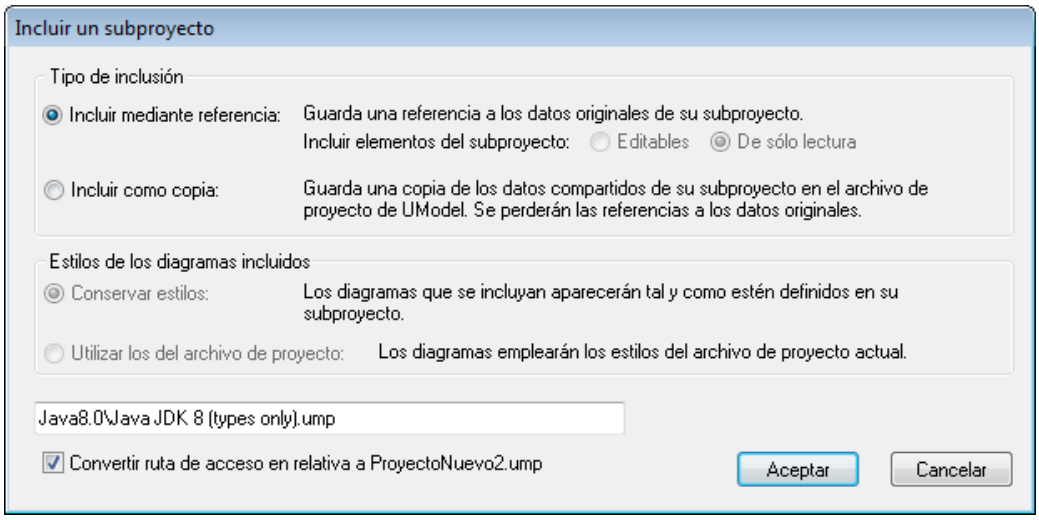
1. En el menú **Proyecto** haga clic en **Incluir un subproyecto**.
2. Haga clic en la pestaña **Java 8.0**, seleccione el proyecto **Java JDK 8 (types only)** y haga clic en **Aceptar.**



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

 © 2018 Altova GmbH

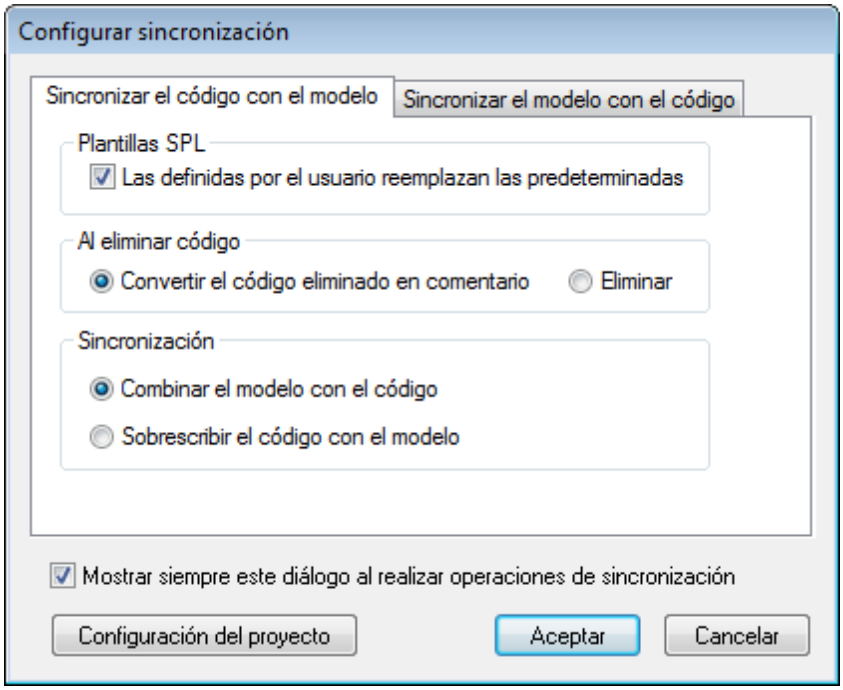
opción *Incluir mediante referencia*.



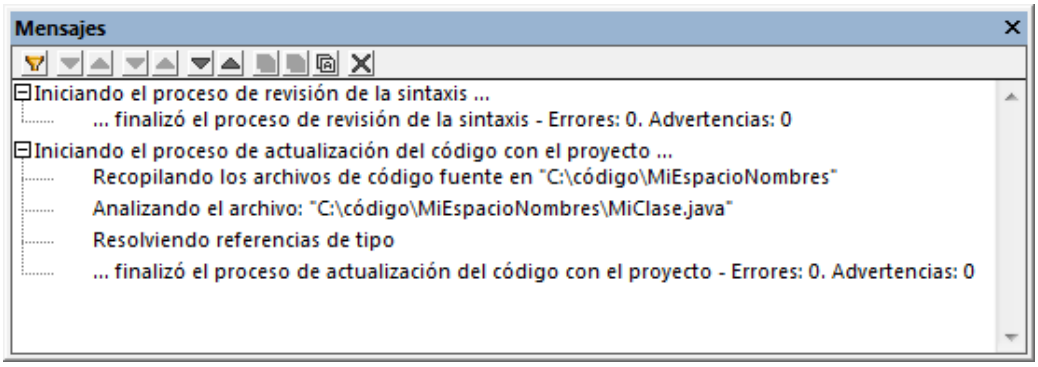
Generar código

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 (p. ej. `public float sum(float num1, float num2){}`). El archivo **MiClase.java** tendrá este aspecto:

```
package MiEspacioNombres;
public class MiClase
{
  public float sum(float num1, float num2){}
}
```

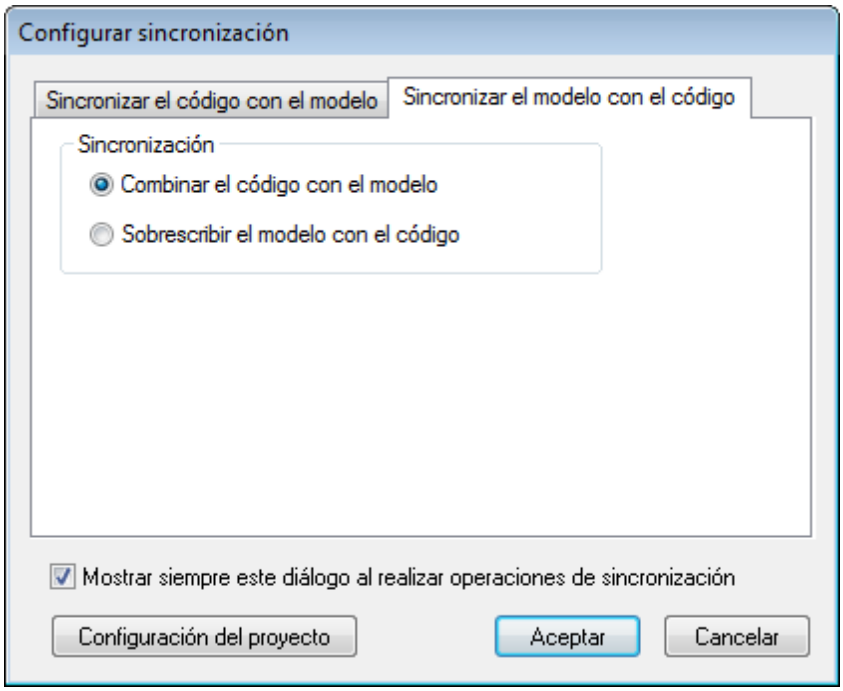
MiClase.java

Combinar cambios realizados en el código con el modelo original

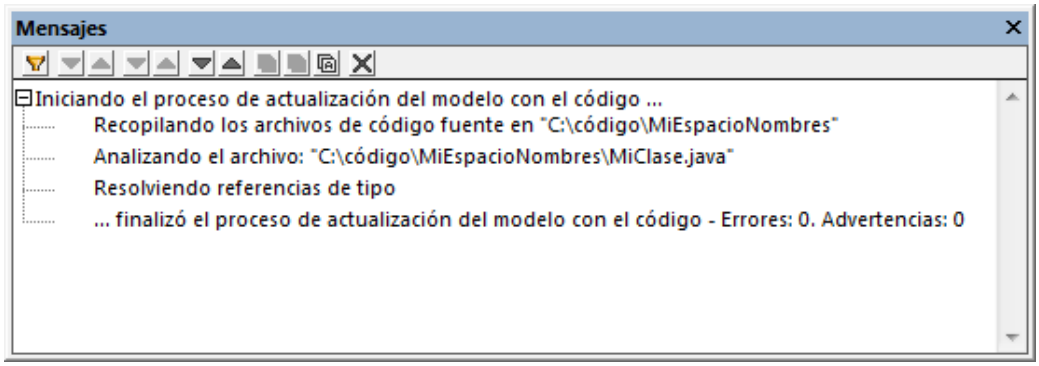
Ahora podemos combinar los cambios realizados en el código con el código original:

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

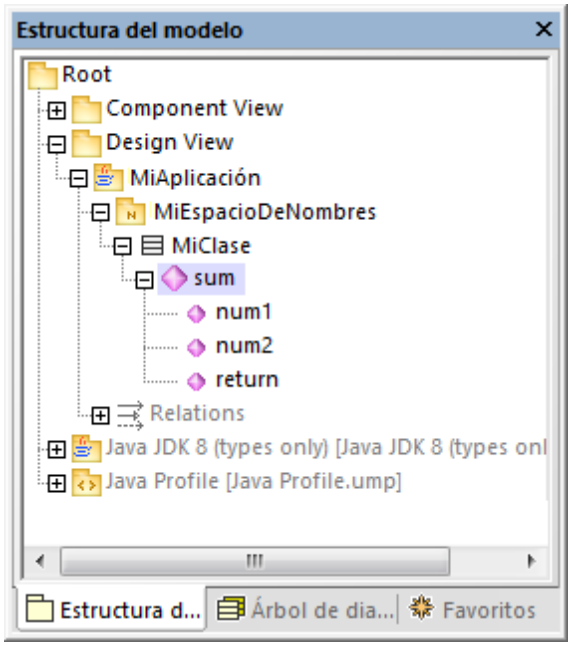
 © 2018 Altova GmbH



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*.



7.2.4 Plantillas SPL

A la hora de generar código 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, la edición de las plantillas SPL solamente tiene sentido cuando se trata de un lenguaje compatible con UModel. Si quiere crear plantillas SPL completamente nuevas para otros lenguajes, podría generar código nuevo pero no podría actualizar el que ya existe (porque la sintaxis del lenguaje será desconocida para UModel).

Las plantillas SPL están en el directorio **UModelSPL** relativo al directorio de instalación del programa.

> No modifique las plantillas SPL predeterminadas porque afectan directamente a la generación de código en UModel. Si necesita personalizar la generación de código, debe crear plantillas personales (*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. El código ya existente no accede a las plantillas SPL ni las utiliza.

En el apartado SPL: el lenguaje de programación Spy 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

 © 2018 Altova GmbH

están guardadas es: **...\UModel2018\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 **...\UModel2018\UModelSPL\Java\Default** y péguelo en **...\UModel2018 \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*.

7.3 Importar código fuente

Puede importar código de programa Java, C# y VB.NET en UModel mediante el proceso conocido como *ingeniería inversa.* Estos son los tipos de proyecto que se pueden importar en 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)

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 en 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.

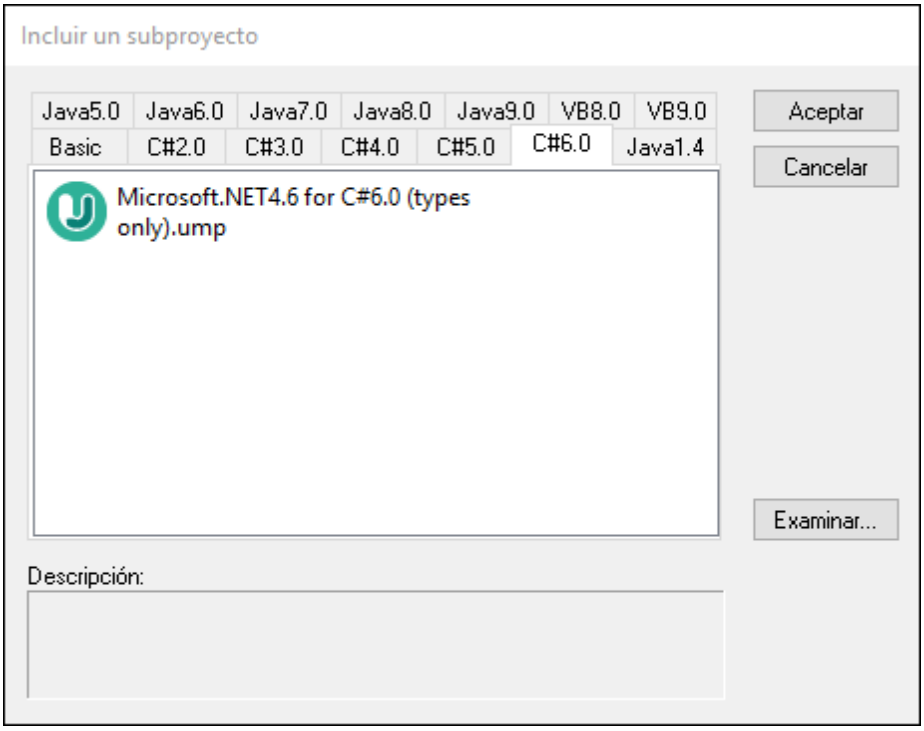
Requisitos

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**.

© 2018 Altova GmbH



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.

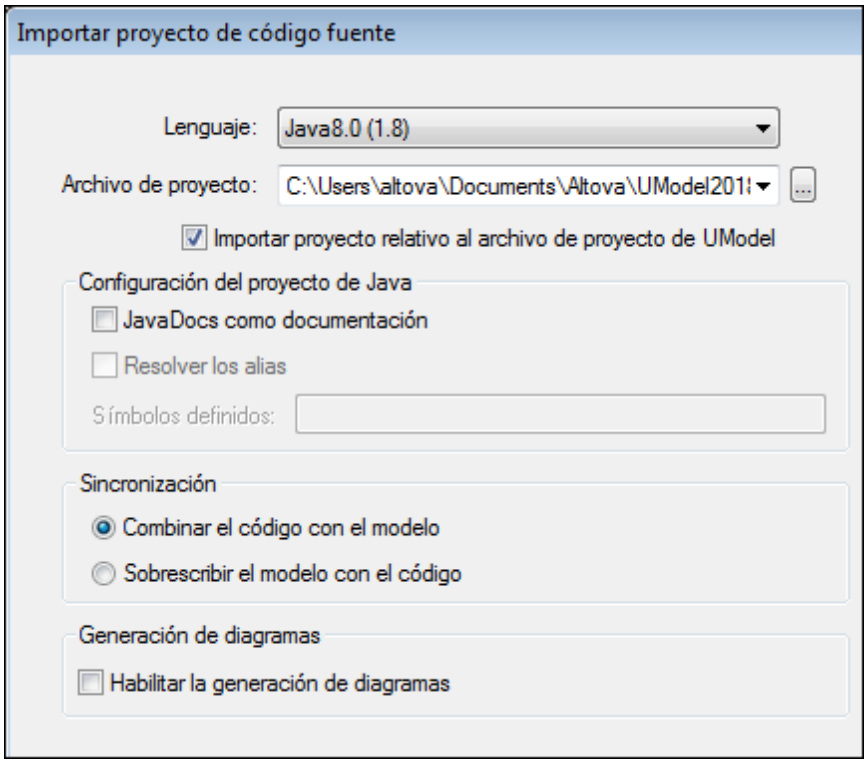
Importar código fuente desde un proyecto

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#.

7.3.1 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 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 un componente UML automáticamente en el proyecto de UModel (se puede ver en la ventana *Estructura del modelo* y es secundario de `Component View`). Este componente realiza

Opción	Descripción
	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 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 generarán.

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 marcar esta casilla. De lo contrario, puede que durante la importación UModel no detecte automáticamente las asociaciones y dependencias en las que participen clases y espacios de nombres con alias (y, por tanto, 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`

Opción	Descripción
	`desconocidos` del proyecto de UModel si su uso no está del todo claro. 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.

© 2018 Altova GmbH

7.3.2 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\UModel2018\UModelExamples\Tutorial\Anagram_CSharp.zip**. Antes de importarla no es necesario compilar la solución con Visual Studio, pero 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 importemos el código, elegiremos la opción de generar diagramas de clases y de paquetes automáticamente.

Paso nº1: crear un proyecto nuevo

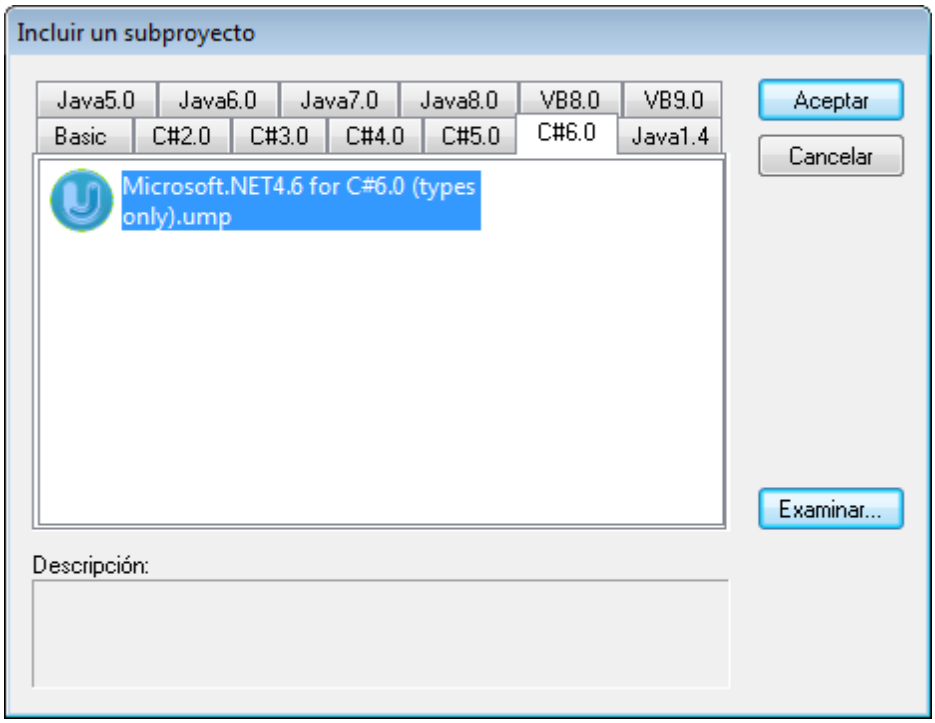
En UModel puede importar código fuente a los proyectos:

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

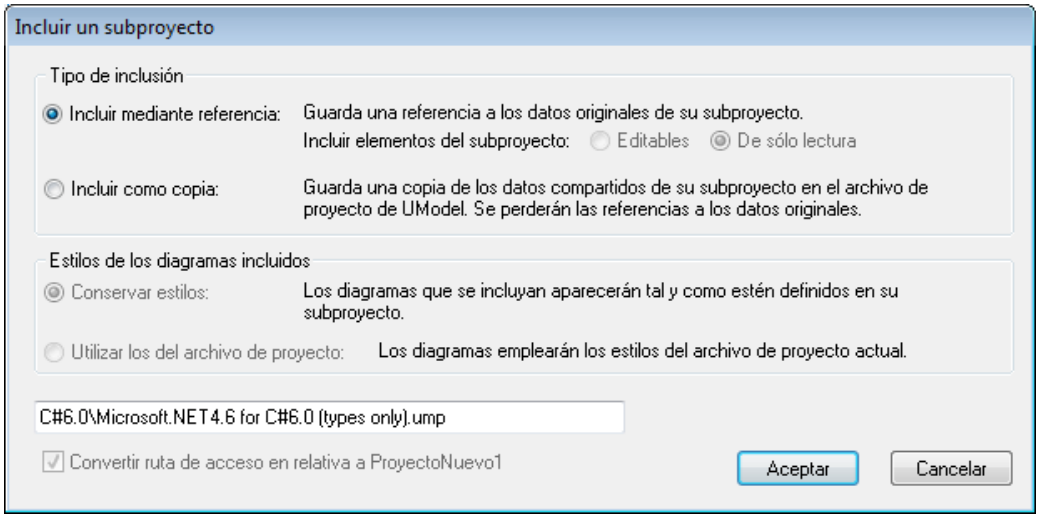
Paso nº2: incluir los tipos del lenguaje C#

El proyecto de origen se escribió en C# con Visual Studio 2015 así que incluiremos un proyecto de UModel integrado que contiene 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#6.0*.

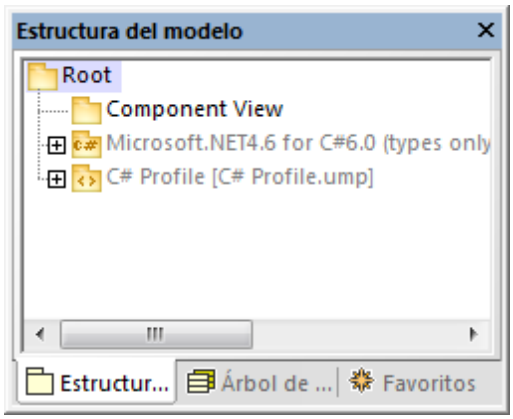


3. Seleccione el proyecto **Microsoft .NET 4.6 for C# 6.0 (types only).ump** y haga clic en **Aceptar**.
4. Cuando deba elegir el tipo de inclusión (mediante referencia o como copia), deje la configuración predeterminada como está.



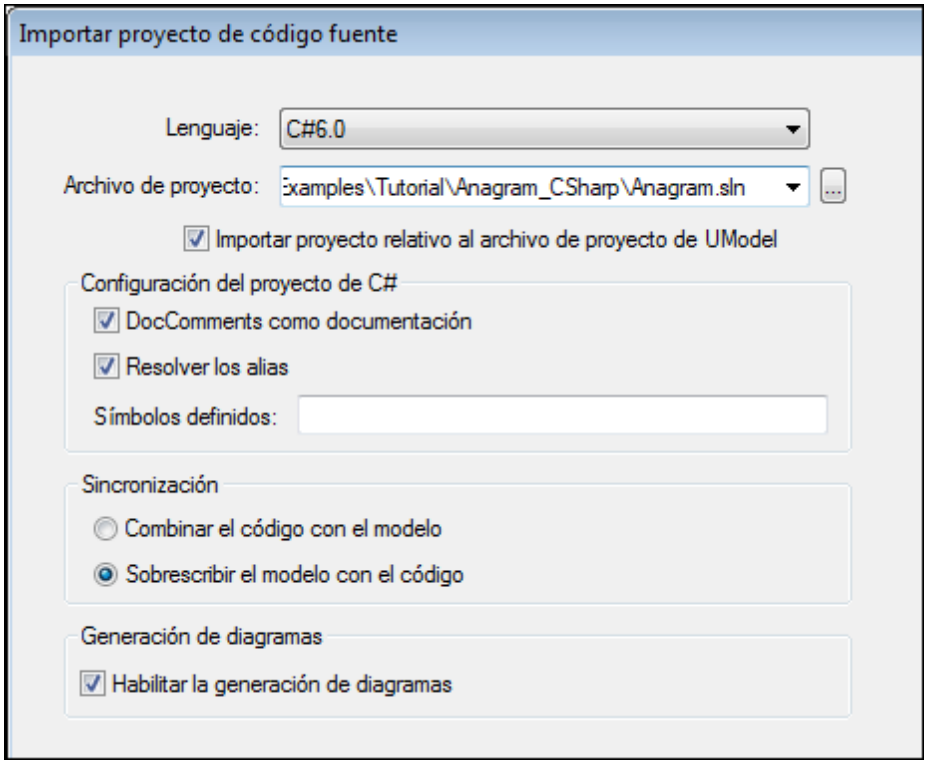
Cuando termina la operación, tanto los tipos del lenguaje C# como el perfil del lenguaje C# se incluyen en el proyecto y pueden verse en la *Estructura del modelo*.

 © 2018 Altova GmbH



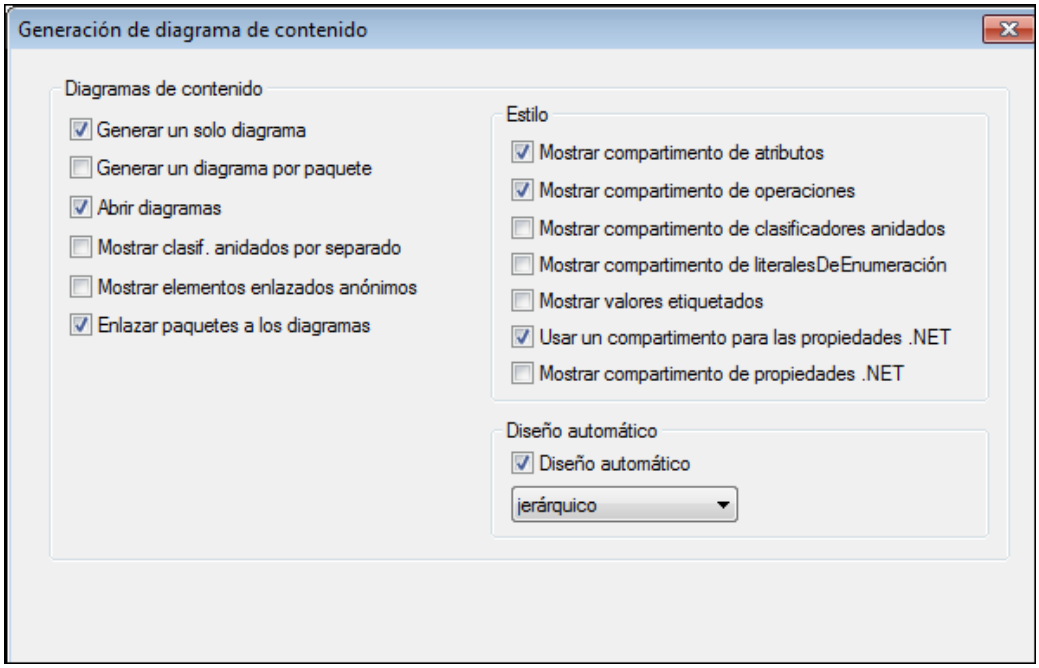
Paso nº3: importar la solución C#

1. En el menú **Proyecto** haga clic en Importar proyecto de código fuente.

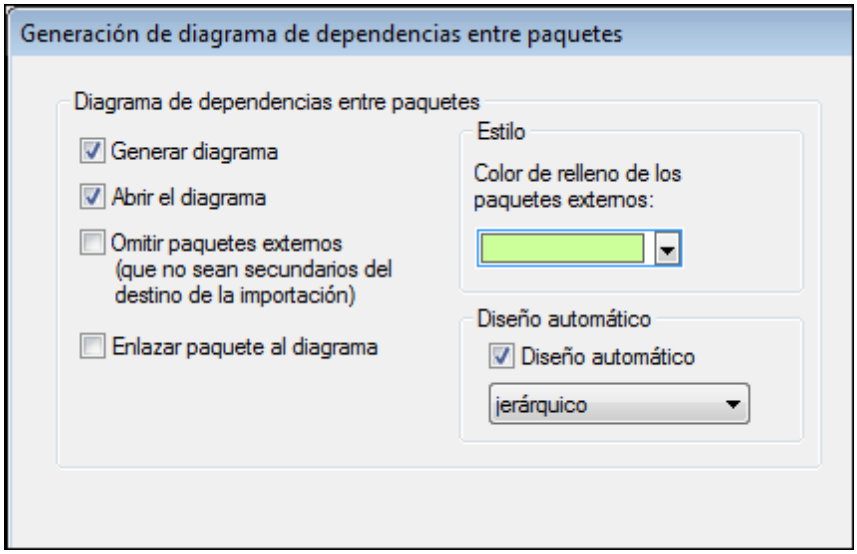


2. Seleccione el lenguaje **C# 6.0**.
3. Haga clic en el botón **Examinar** del campo *Archivo de proyecto* y navegue hasta el archivo de solución .sln.
4. Marque la casilla *DocComments como documentación* (esto importará los comentarios de código de operaciones o propiedades al modelo).
5. Como estamos 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.

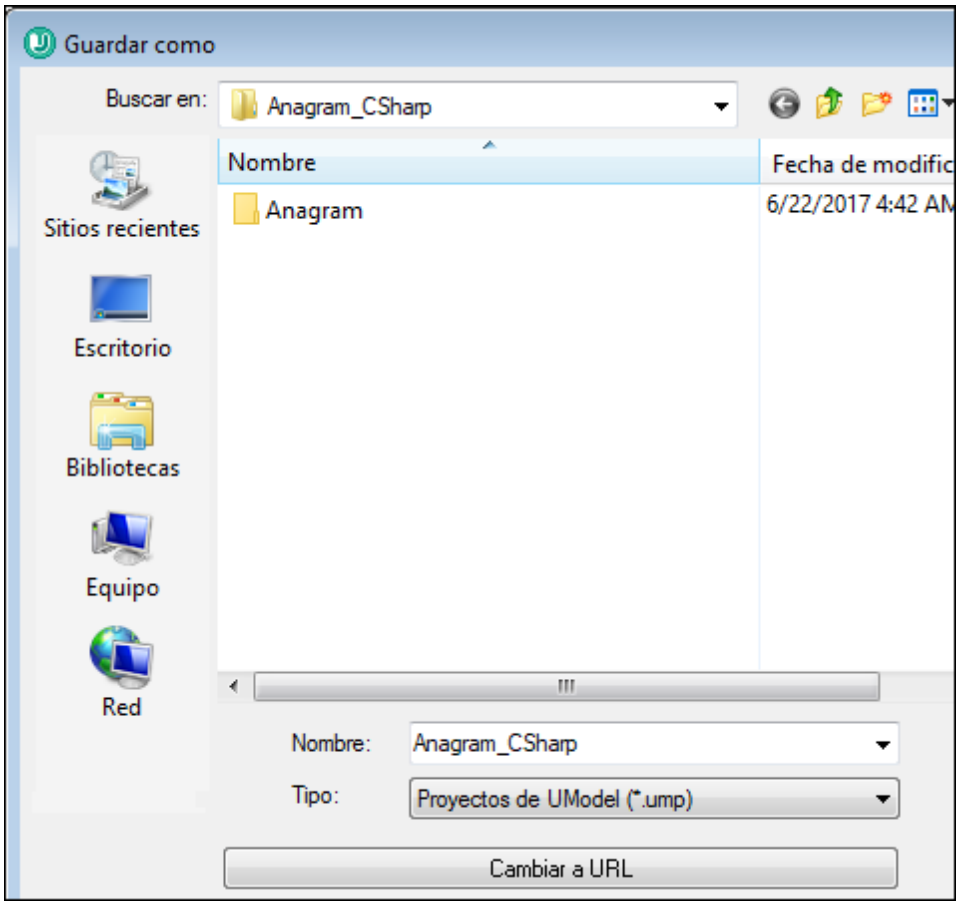


8. Seleccione las opciones de generación de diagramas que aparecen en la imagen siguiente y haga clic en **Finalizar**. Estas opciones afectan a los diagramas de paquetes que se generan automáticamente cuando se importa código.

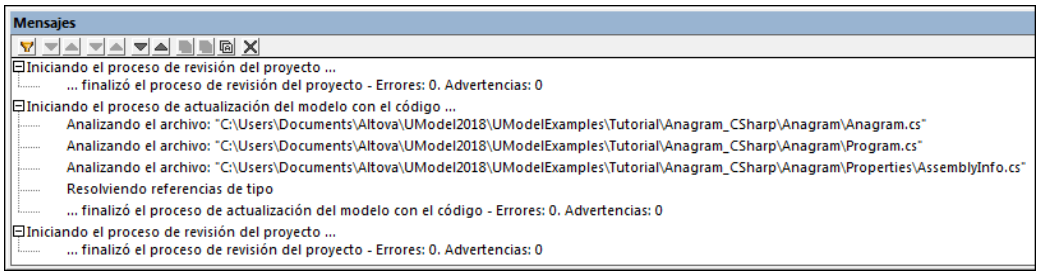


9. Introduzca un nombre y seleccione una carpeta de destino para el nuevo proyecto de UModel y haga clic en **Guardar** (este cuadro de diálogo muestra por defecto la carpeta donde está la solución que se está importando).

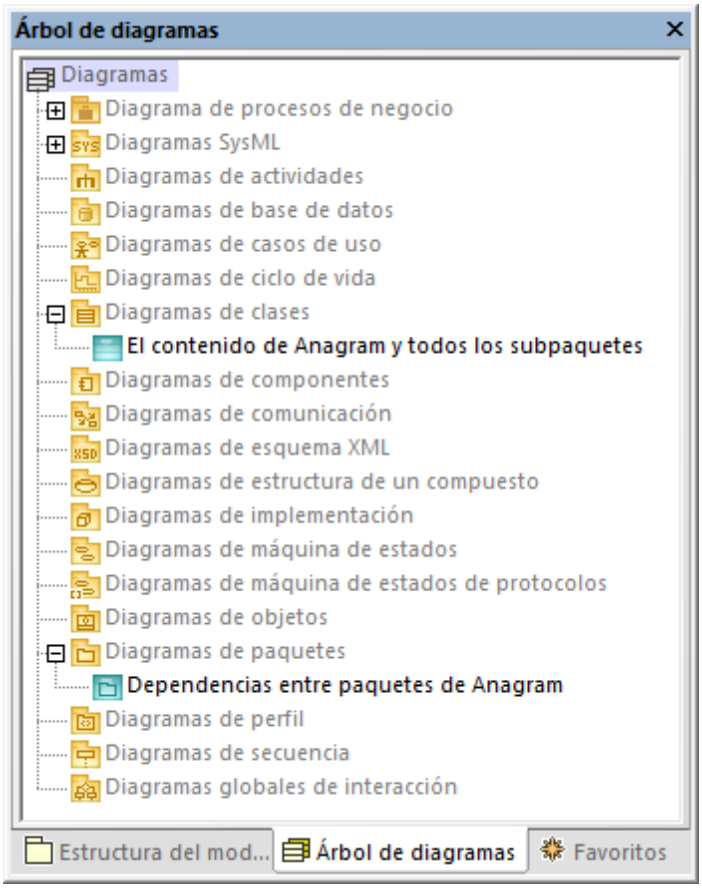
 © 2018 Altova GmbH



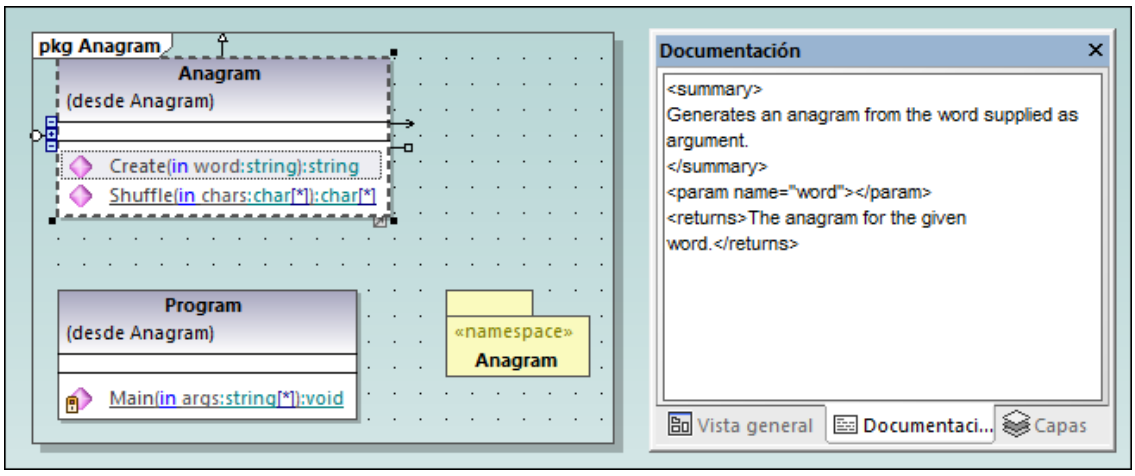
El progreso de la operación de ingeniería inversa aparece en la ventana Mensajes.



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.



7.4 Importar binarios Java, C# y VB.NET

UModel permite la importación de binarios C#, Java y VB, lo cual es de gran utilidad cuando se trabaja con binarios de terceros o cuando el código fuente original ya no está disponible. Debe tener en cuenta que:

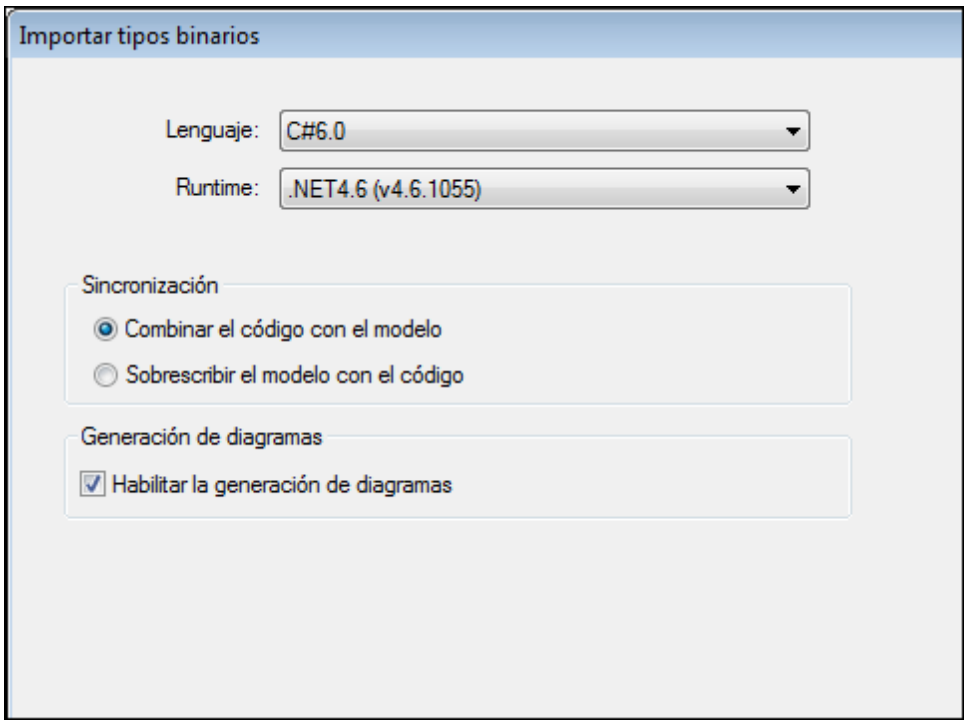
- Si tiene pensado importar archivos binarios Java, es necesario tener instalado Sun Java Runtime Environment (JRE) o el kit de desarrollo JDK en las versiones 1.4, 1.5, 1.6, 1.7 o 1.8. La importación de tipos es compatible con todos los archivos de clases que apunten a estos entornos (es decir, que cumplan la especificación de Java Virtual Machine).

- Si tiene pensado importar archivos binarios C# o VB.NET, es necesario tener instalado .NET Framework. La importación de tipos es compatible con todos los ensamblados que apunten a .NET Framework 1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.6 y .NET Compact Framework v1.0, v2.0, v3.5 (para PocketPC, Smartphone, WindowsCE).

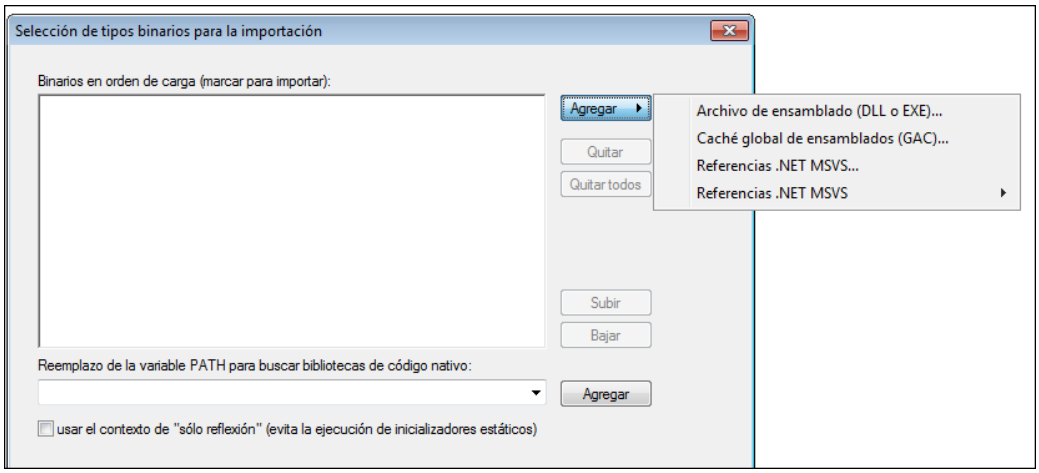
- No es posible importar binarios confusos.

Para importar archivos binarios:

1. En el menú **Proyecto** haga clic en el comando **Importar tipos binarios**.
2. Elija el lenguaje y la plataforma de origen y haga clic en **Siguiente**.



3. Haga clic en **Agregar** y seleccione el binario de origen (p. ej. un archivo .exe o .dll) y después siga los pasos del asistente.



Más abajo explicamos cómo importar binarios Java, pero las instrucciones son similares para C# y VB.NET.

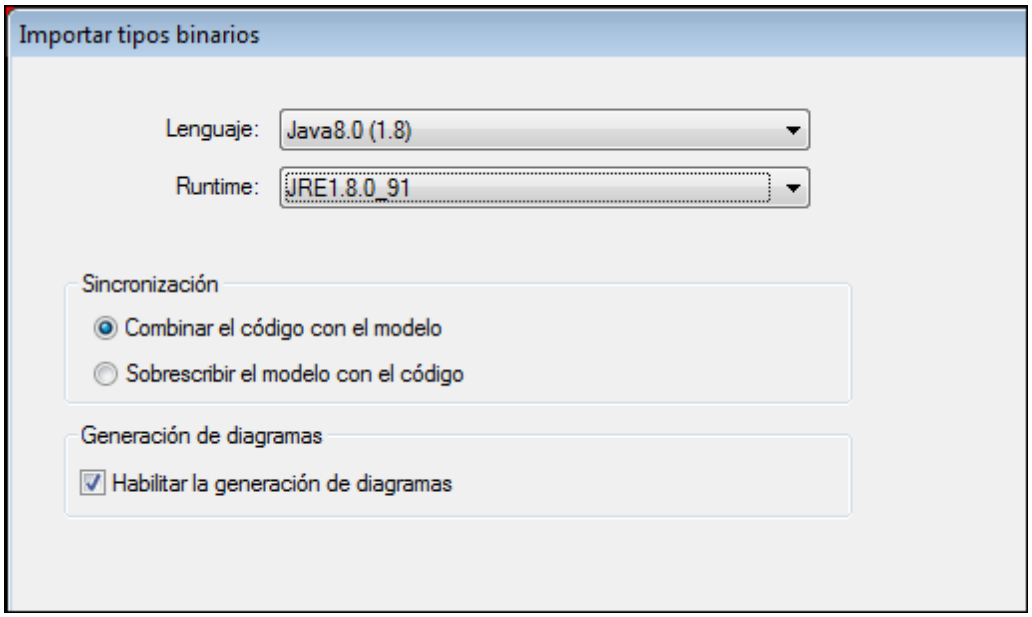
Notas:

- El cuadro de texto *Reemplazo de la variable PATH...* solamente es relevante para Java. Si quiere, puede pegar aquí todas las rutas de clase Java que deban consultarse junto a las que se deben leer de la variable del entorno `CLASSPATH` (o haga clic en **Agregar** y busque en las carpetas correspondientes).
- La casilla *usar el contexto de "sólo reflexión"...* solamente es relevante cuando se importa un binario C# o VB.NET. Esta casilla es muy práctica a la hora de importar una biblioteca (.dll, etc.) que tenga dependencias que no se pueden resolver o cargar. Además, si se marca esta casilla, no se ejecutará el código estático inicializador porque esto podría causar errores durante la importación.

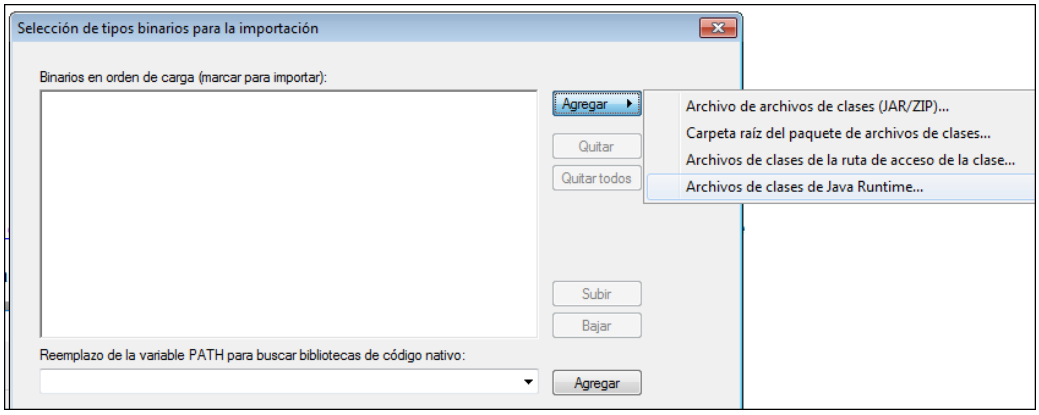
Ejemplo: importar binarios Java

1. Seleccione el comando de menú **Proyecto | Importar tipos binarios**.

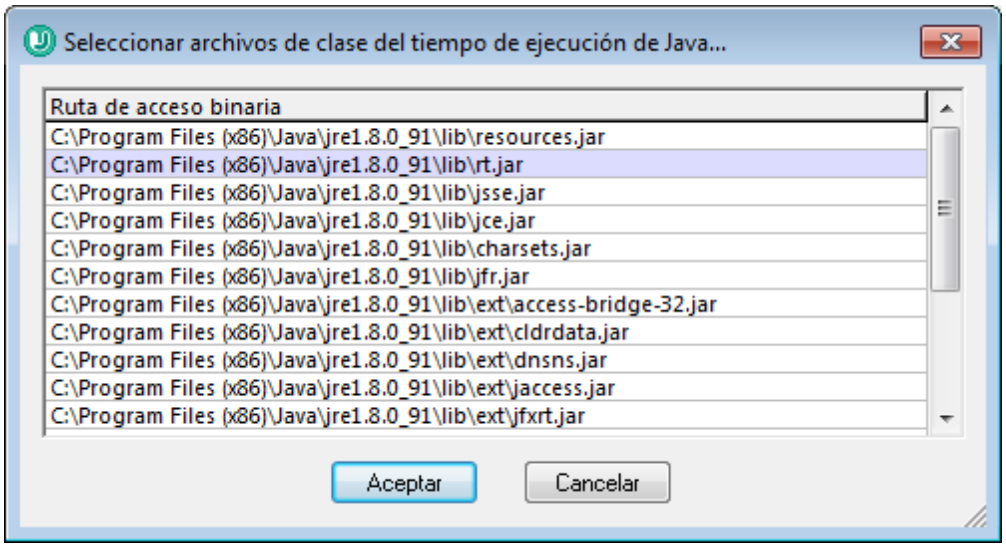
 © 2018 Altova GmbH



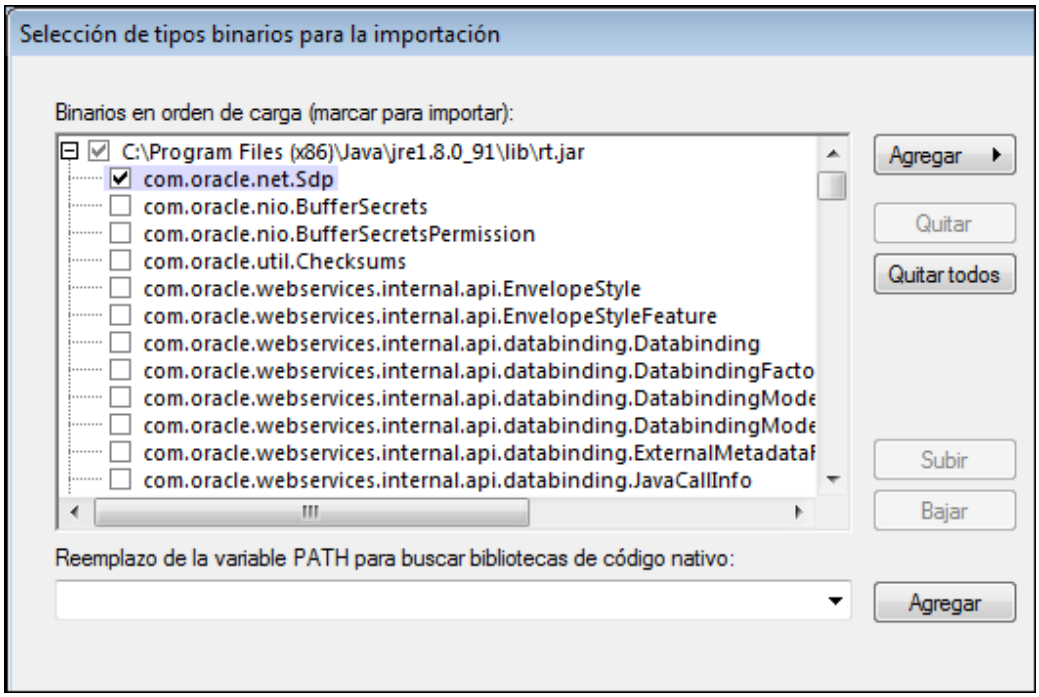
2. Seleccione el lenguaje y la edición de Runtime.
3. Para que UModel pueda generar diagramas de clases y diagramas de paquetes a partir del código fuente, marque la casilla *Habilitar la generación de diagramas* y haga clic en **Siguiente**.
4. Ahora aparece el cuadro de diálogo "Selección de tipos binarios para la importación".



5. Haga clic en **Agregar** y seleccione el archivo de clases en el menú contextual (en este ejemplo añadimos **Archivos de clases de Java Runtime**).
6. Seleccione un binario (en este ejemplo utilizamos `rt.jar`).

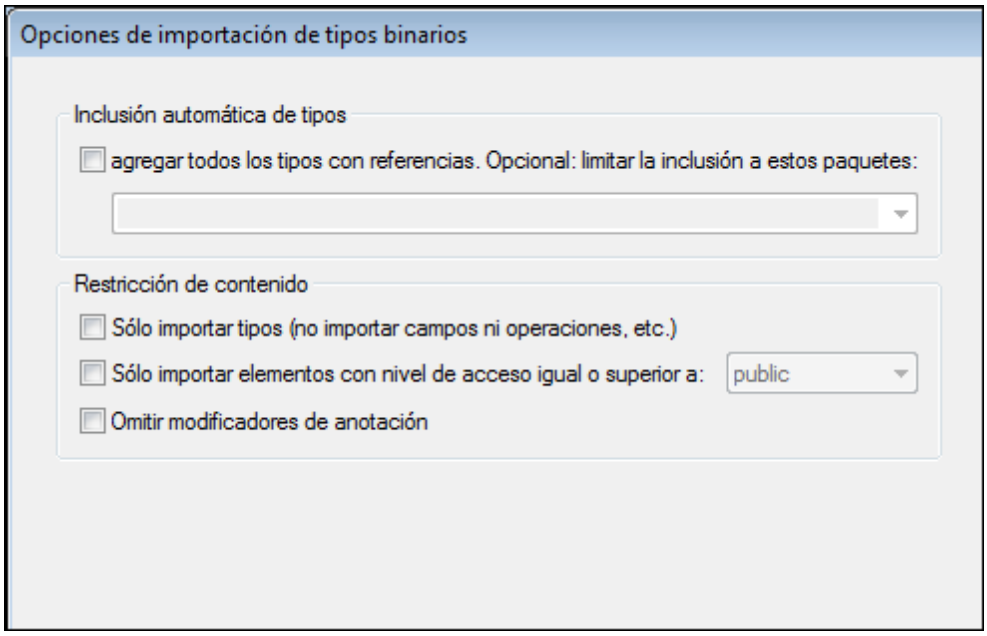


7. Ahora haga clic en el botón + para expandir la lista de binarios y marque las casillas de los binarios que desea importar.

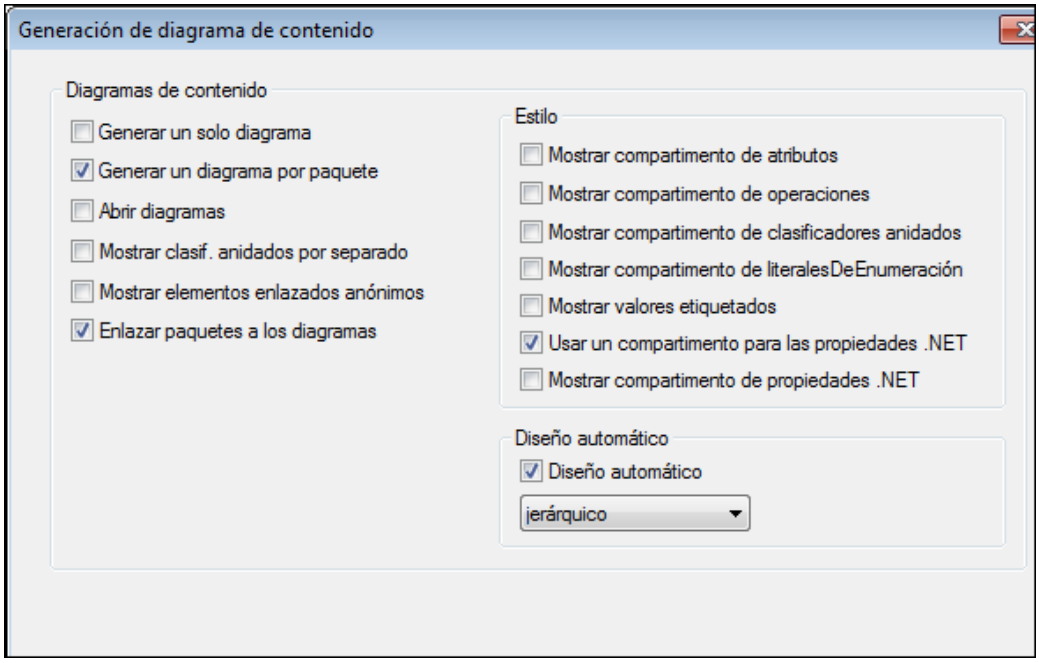


8. Haga clic en **Siguiente**. Esto abre el cuadro de diálogo "Opciones de importación de tipos binarios".

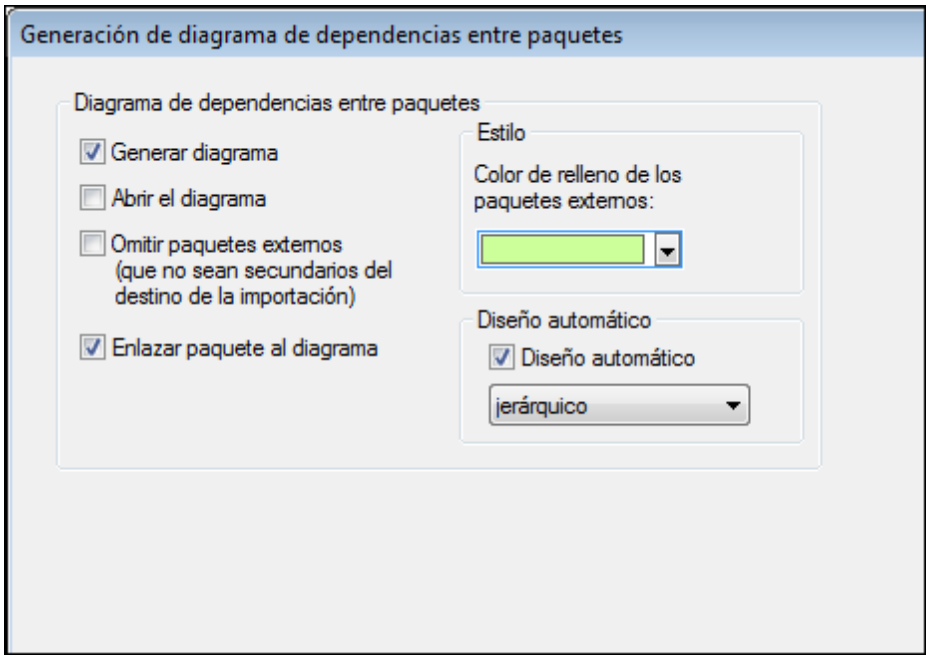
 © 2018 Altova GmbH



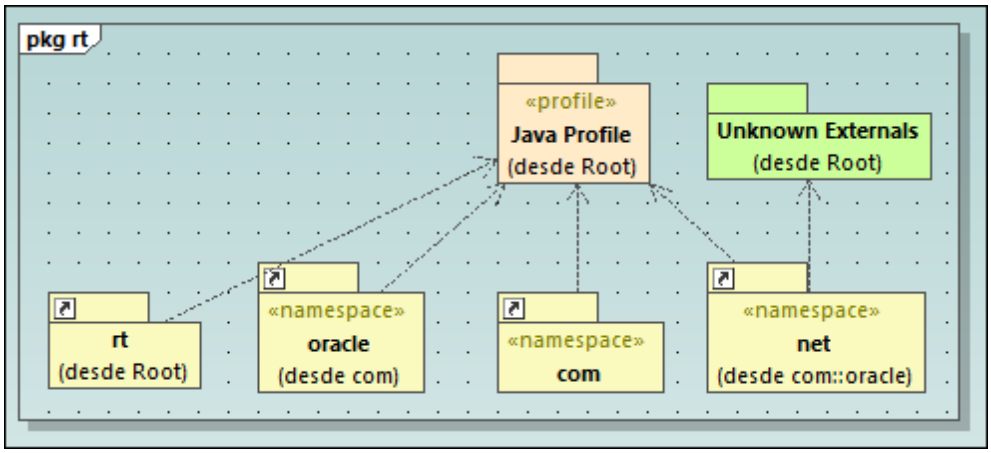
9. Haga clic en **Siguiente**.
10. Si es necesario, defina el destino de la importación o marque la casilla *Importar en un paquete nuevo* y después haga clic en **Siguiente**.



11. Seleccione las propiedades de generación del diagrama de contenido y haga clic en **Siguiente** para continuar. Puede generar un diagrama por cada paquete o un solo diagrama.



12. Seleccione las opciones de dependencias entre paquetes que desea incluir y haga clic en **Finalizar** para terminar el proceso de importación. En la imagen siguiente puede ver el diagrama que contiene las dependencias entre paquetes de los binarios Java.



Nota: al hacer clic en el icono en forma de flecha de una carpeta aparece automáticamente el diagrama al que se hace referencia (véase también Agregar hipervínculos a elementos).

 © 2018 Altova GmbH

7.5 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 hacer a nivel de proyecto. La opción para actualizar clases y paquetes por separado no está disponible.

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 en este caso un proyecto, un paquete, un componente, una clase, etc., dependiendo de qué tipo de elemento se seleccionara 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 pulsado **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 pulsado **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.

A fin de evitar resultados no deseados a la hora de sincronizar modelo y código, debe tener en cuenta todas 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.

7.5.1 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

 © 2018 Altova GmbH

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 **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 se le puede asignar un solo componente. Es decir:

- un clasificador que no tiene ninguna RealizaciónDeComponente o
- un clasificador que está 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:

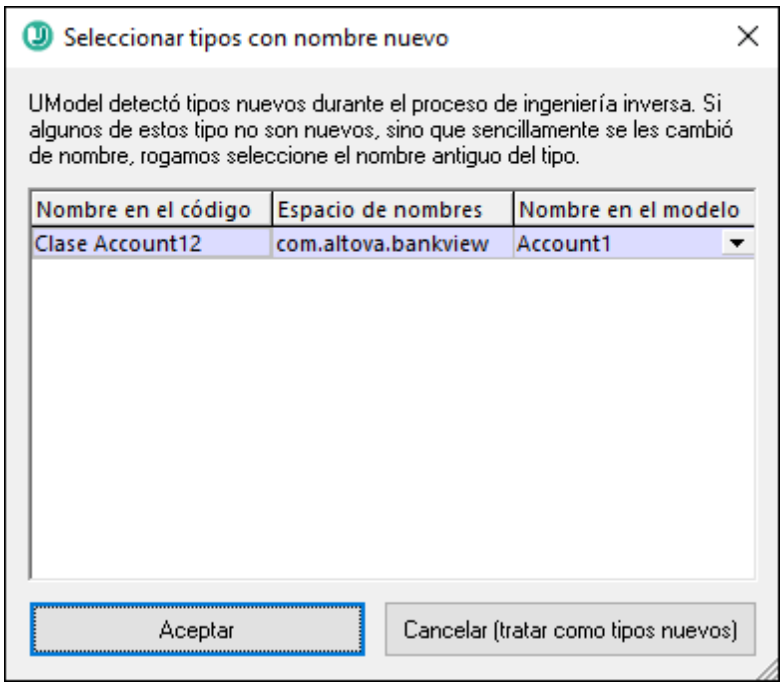
- Es necesario activar la opción *Generar las realizacionesDeComponente que falten* (de la pestaña *Ingeniería de código* del cuadro de diálogo "Opciones locales").
- En cuanto UModel encuentra UN componente viable durante los pasos descritos más arriba, el componente se utiliza y se omiten los pasos siguientes.

Errores/advertencias:

- Si no se encuentra ningún componente viable, se genera una advertencia.
- Si se encuentran varios componentes viables, se genera un error.

7.5.2 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`

 © 2018 Altova GmbH

`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.

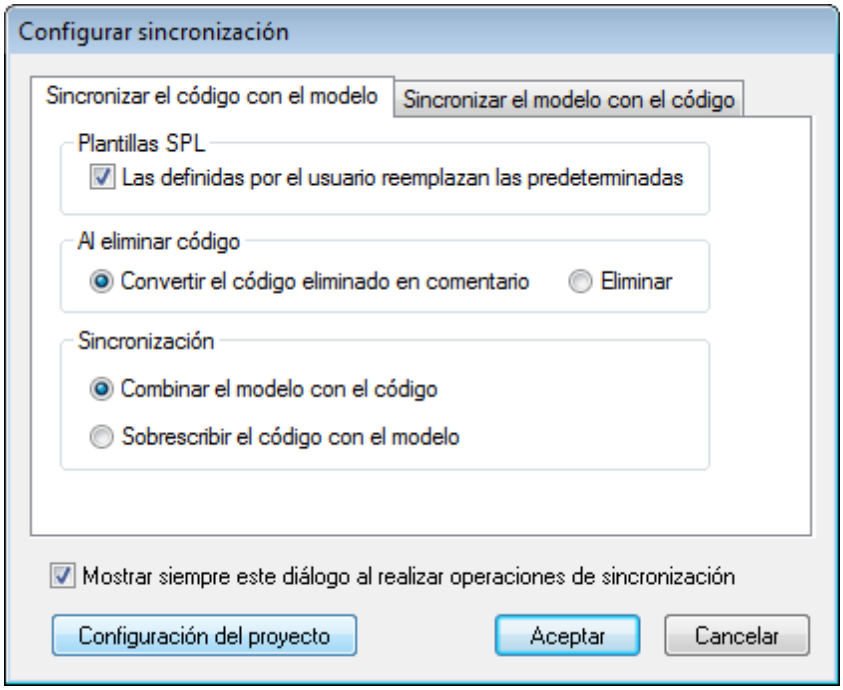
7.5.3 Configurar la sincronización del código

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 ejecutan los comandos **Proyecto | Combinar el código de programa con el proyecto de UModel** y **Proyecto | Sobrescribir el código de programa con el proyecto de UModel**).
- Cuando se importa código fuente en el modelo (es decir, cuando se ejecutan los comandos **Proyecto | Combinar el proyecto de UModel con el código de programa** y **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, durante la sincronización el código, eliminado se elimina por completo o se convierte en comentario.
Sincronización	Esta opción se aplica tanto si se genera como si se importa código de programa. Elija si los cambios deben combinarse o sobrescribirse.

 © 2018 Altova GmbH

Opción	Descripción
	Suponiendo que el código fuera generado a partir de un modelo y que se realizaran 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).

7.6 Correspondencias con elementos de UModel

Esta sección muestra las correspondencias entre elementos de UModel y elementos (construcciones) de los demás lenguajes de programación (C#, 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.

7.6.1 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	
		private		private	
		sealed	leaf		

© 2018 Altova GmbH

C#				UModel			
	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)			
Constant	name			name			Property <<const >>
	modifiers	internal		visibility	package		
		protected internal			protected		

C#				UModel			
					<<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)			
		implemented interfaces		implements			
		type		direction	return	Parameter	
		Parameter	name	name			

© 2018 Altova GmbH

C#					UModel				
		er	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		
			modifiers	ref	direction	inout			
				out		out			
				params	varArgList				

C#					UModel				
			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>>				
		Get Accessor	modifiers	internal	visibility	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			

© 2018 Altova GmbH

<table>
<tr><th colspan="5">C#</th><th colspan="5">UModel</th></tr>
<tr><td rowspan="19"></td><td rowspan="6"></td><td rowspan="6"></td><td rowspan="6"></td><td>private</td><td rowspan="6"></td><td>private</td><td rowspan="6"></td><td rowspan="6"></td><td rowspan="19"></td></tr>
<tr><td rowspan="5">Set Accessor</td><td rowspan="5">modifiers</td><td>internal</td><td rowspan="5">visibility</td><td>internal</td><td rowspan="5"><<SetAccessor>></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>Operator</td><td colspan="2">name</td><td colspan="3">name</td><td rowspan="13">Operation
<<operator>></td></tr>
<tr><td rowspan="3">modifiers</td><td colspan="2">public</td><td colspan="2">visibility</td><td>public</td></tr>
<tr><td colspan="2">static</td><td colspan="3">static</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">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="3">name</td></tr>
<tr><td>modifier</td><td>params</td><td colspan="3">varArgList</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">type pointer</td><td colspan="3">type modifier</td></tr>
<tr><td colspan="2">nullable</td><td colspan="3"><<nullable>></td></tr>
<tr><td>Indexer</td><td colspan="2">name (="this")</td><td colspan="3">name (="this")</td><td rowspan="12">Operation
<<indexer>></td></tr>
<tr><td rowspan="11">modifiers</td><td>internal</td><td rowspan="11">visibility</td><td>package</td></tr>
<tr><td>protected internal</td><td>protected
<<internal>></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>static</td><td colspan="2">static</td></tr>
<tr><td>abstract</td><td colspan="2">abstract</td></tr>
<tr><td>sealed</td><td colspan="2">leaf</td></tr>
<tr><td>override</td><td colspan="2"><<override>></td></tr>
<tr><td>virtual</td><td colspan="2"><<virtual>></td></tr>
</table>

<table>
<tr><th colspan="5">C#</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="20"></td><td rowspan="19"></td><td rowspan="2"></td><td colspan="2">new</td><td colspan="3"><<new >></td><td rowspan="19"></td><td rowspan="19"></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" rowspan="2">type</td><td>direction</td><td>return</td><td rowspan="7">Parameter</td></tr>
<tr><td rowspan="6">Parameter</td><td colspan="2">name</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>
<tr><td>private</td><td>private</td></tr>
<tr><td>Event</td><td colspan="2">name</td><td colspan="2">name</td><td rowspan="11">Operation <<event>></td></tr>
<tr><td rowspan="10">modifiers</td><td rowspan="2">internal</td><td rowspan="5">visibility</td><td>package</td></tr>
<tr><td>protected internal</td><td>protected <<internal>></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 colspan="2">static</td><td colspan="2">static</td></tr>
<tr><td colspan="2">abstract</td><td colspan="2">abstract</td></tr>
<tr><td colspan="2">sealed</td><td colspan="2">leaf</td></tr>
</table>

© 2018 Altova GmbH

C#				UModel				
			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>>		
		new	<<new >>		
	filename		code file name		

C#				UModel		
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	
		new		<<new >>		
	type			type		

© 2018 Altova GmbH

C#				UModel			
		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			
			sealed	leaf			
			override	<<override>>			

C#					UModel				
			partial		<<partial>>				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		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			

C#					UModel			
			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>>			
			virtual		<<virtual>>			
			new		<<new>>			
			unsafe		<<unsafe>>			

C#					UModel				
		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>>				
	Indexer	name (="this")			name (="this")			Operation <<indexer>>	
		modifiers	internal		visibility	package			
			protected internal			protected			

© 2018 Altova GmbH

<table>
<tr><th colspan="4">C#</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="15"></td><td rowspan="15"></td><td rowspan="11"></td><td></td><td rowspan="5"></td><td><<internal>></td><td rowspan="15"></td><td rowspan="15"></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>static</td><td colspan="2">static</td></tr>
<tr><td>abstract</td><td colspan="2">abstract</td></tr>
<tr><td>sealed</td><td colspan="2">leaf</td></tr>
<tr><td>override</td><td colspan="2"><<override>></td></tr>
<tr><td>virtual</td><td colspan="2"><<virtual>></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="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="7">Parameter</td></tr>
<tr><td rowspan="6">Parameter</td><td colspan="2">name</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>
<tr><td>private</td><td>private</td></tr>
<tr><td>Event</td><td colspan="3">name</td><td colspan="2">name</td><td>Operatio</td></tr>
</table>

C#				UModel			
		modifiers	internal	visibility	package	n <<event >>	
			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	Paramet er	
		type dimensions		multiplicity			
		nullable		<<nullable>>			
		Add Accessor		<<AddRemoveAccessor>>			
		Remove Accessor					
	Type Paramet er	name		name		Template Paramet er	
		constraint		constraining classifier			
		predefin ed constrai nt	struct	<<ValueTypeConstraint>>			
			class	<<ReferenceTypeConstraint>>			
			new ()	<<ConstructorConstraint>>			
		attribute sections		<<attributes>>			

C# Interface

C#		UModel	
Interface	name	name	Interface

© 2018 Altova GmbH

C#					UModel				
	modifiers	internal			visibility	package			
		protected internal				protected <<internal>>			
		public				public			
		protected				protected			
		private				private			
		unsafe			<<unsafe>>				
		partial			<<partial>>				
		new			<<new >>				
	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			
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Comment(->Documentation)				
		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				

C#					UModel			
			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		
			new		<<new >>			
			unsafe		<<unsafe>>			
		attribute sections			<<attributes>>			
		doc comments			Comment(->Documentation)			

© 2018 Altova GmbH

C#				UModel				
		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>>		
			modifiers: protected internal		protected internal			
			modifiers: protected		protected			
			modifiers: private		private			
		Set Accessor	modifiers: internal	visibility	internal	<<SetAccessor>>		
			modifiers: protected internal		protected internal			
			modifiers: protected		protected			
			modifiers: 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>>				
		Remove Accessor						
	Type Parameter	name		name			Template Parameter	
		constraint		constraining classifier				
		predefin	struct	<<ValueTypeConstraint>>				

C#				UModel		
		ed constraint	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				
		nullable			<<nullable>>				
	Type Parameter	name			name			Template Parameter	
		constraint			constraining classifier				

© 2018 Altova GmbH

C#					UModel				
		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>>		
		doc comments	Comment(->Documentation)		

C# Parameterized Type

C#	UModel
Parameterized Type	Anonymous Bound Element

7.6.2 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)		
	attribute sections		<<Attributes>>		
	doc comments		Comment(->Documentation)		
	Field	name	name	Property	

© 2018 Altova GmbH

VB.NET				UModel			
		modifiers	Friend	visibility	package		
			Protected Friend		protected <<Friend>>		
			Public		public		
			Protected		protected		
			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)			
	Constant	name		name		Property <<Const>>	
		modifiers	Friend	visibility	package		
			Protected Friend		protected <<Friend>>		
			Public		public		
			Protected		protected		
			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		
			Protected Friend		protected <<Friend>>		

VB.NET					UModel			
			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 constraint	Structure	<<ValueTypeConstraint>>			
				Class	<<ReferenceTypeConstraint>>			
				New	<<ConstructorConstraint>>			

© 2018 Altova GmbH

VB.NET				UModel			
			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		
			modifiers ByVal		in		
			modifiers ParamArray	varArgList			
			modifiers 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			
			MustOverride	abstract			
			NotOverriddable	leaf			
			Overrides	<<Overrides>>			

VB.NET					UModel				
			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)				
		type			direction	return	Parameter		
		Paramet	name		name				

© 2018 Altova GmbH

VB.NET					UModel				
			modifier	ByVal	direction	in			
			type		type				
			type dimensions		multiplicity				
			nullable		<<Nullable>>				
	Event	name			name			Operation <<Event>>	
		modifiers	Friend		visibility	package			
			Protected Friend			protected <<Friend>>			
			Public			public			
			Protected			protected			
			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 constrai	Structure		<<ValueTypeConstraint>>				

VB.NET				UModel			
		nt	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)			
	Constant	name		name			Property <<Const>>
		modifiers	Friend	visibility	package		

© 2018 Altova GmbH

VB.NET					UModel			
			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		
				ByVal		in		
				ParamArray	varArgList			
				Optional	default			
			type		type			

VB.NET					UModel			
			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 <<Property>>
		modifiers	Friend		visibility	package		
			Public			public		
			Private			private		
			Shared		static			

© 2018 Altova GmbH

VB.NET					UModel				
			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)				
		type			direction	return	Parameter		
		Parameter	name		name				
			modifier	ByVal	direction	in			
			type		type				

VB.NET				UModel			
			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>>			
			New	<<ConstructorConstraint>>			
		attribute sections		<<Attributes>>			
Interface	name			name			Interface
	modifier	Friend		visibility	package		

© 2018 Altova GmbH

VB.NET					UModel			
	s	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		
				ByVal		in		
				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	<<ConstructorConst			

VB.NET				UModel			
				raint>>			
			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>>			
	Type Parameter	name		name			Template Parameter
		constraint		constraining classifier			
		predefined constraint	Structure	<<ValueTypeConstraint>>			
			Class	<<ReferenceTypeConstraint>>			
			New	<<ConstructorConstraint>>			

© 2018 Altova GmbH

VB.NET			UModel			
		attribute sections	<<Attributes>>			
Delegate	name		name			Class <<Delegate>>
	modifiers	Friend	visibility	package		
		Protected Friend		protected <<Friend>>		
		Public		public		
		Protected		protected		
		Private		private		
		Shadows	<<Shadows>>			
	filename		code file name			
	associated projectfile/directory		ComponentRealization			
	attribute sections		<<Attributes>>			
	doc comments		Comment(->Documentation)			
	type		direction	return	Parameter	Operation
	Parameter	name	name			
		modifiers ByRef	direction	inout		
		ByVal		in		
		type	type			
		type dimensions	multiplicity			
		nullable	<<Nullable>>			
	Type Parameter	name	name			Template Parameter
		constraint	constraining classifier			
		predefined constraint struct	<<ValueTypeConstraint>>			
		class	<<ReferenceTypeConstraint>>			
		new()	<<ConstructorConstraint>>			
		attribute sections	<<Attributes>>			
Enum	name		name			Enumeration
	modifiers	Friend	visibility	package		
		Protected Friend		protected <<Friend>>		
		Public		public		
		Protected		protected		
		Private		private		

VB.NET			UModel		
		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		

7.6.3 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
	modifiers	package	visibility	package	
		public		public	
		protected		protected	
		private		private	
		abstract	abstract		
		strictfp	<<strictfp>>		

© 2018 Altova GmbH

Java			UModel		
	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	
		static	static		
		abstract	abstract		
		final	<<final>>		
		native	<<native>>		
		strictfp	<<strictfp>>		
		synchronized	<<synchronized>>		
	throws clause		raised exceptions		

Java					UModel				
		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				
	Type Parameter	name			name			Template Parameter	
		bound			constraining classifier				
Interface	name				name				Interface
	modifiers	package			visibility	package			
		public				public			
		protected				protected			
		private				private			

© 2018 Altova GmbH

Java				UModel			
		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			
			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			Enumera

Java			UModel		
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
	modifiers	package	visibility	package	
		public		public	
		protected		protected	
		private		private	
		static	static		
		abstract	abstract		
		final	<<final>>		
		native	<<native>>		

© 2018 Altova GmbH

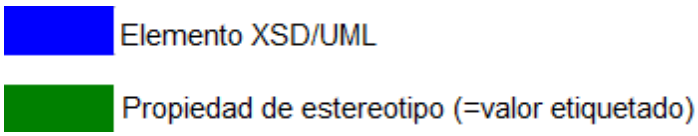
Java					UModel			
			strictfp		<<strictfp>>			
			synchronized		<<synchronized>>			
		throws clause			raised exceptions			
		java docs			Comment(->Documentation)			
		type			direction	return	Parameter	
		Parameter	name		name			
			modifier	final	<<final>>			
			...		varArgList			
			type		type			
			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			
Parameterized Type					Anonymous Bound Element			
Annotation					<<annotations> modifiers			

7.6.4 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:



<table>
<tr><th colspan="6">XSD</th><th colspan="3">UModel</th></tr>
<tr><td colspan="6">file path</td><td colspan="2">projectfile</td><td>Component</td></tr>
<tr><td rowspan="14">schema</td><td colspan="5">target namespace</td><td colspan="2">name</td><td>Package
<<namespace>></td></tr>
<tr><td colspan="5">attributeFormDefault</td><td colspan="2">attributeFormDefault</td><td rowspan="13">Class
<<schema>></td></tr>
<tr><td colspan="5">blockDefault</td><td colspan="2">blockDefault</td></tr>
<tr><td colspan="5">elementFormDefault</td><td colspan="2">elementFormDefault</td></tr>
<tr><td colspan="5">finalDefault</td><td colspan="2">finalDefault</td></tr>
<tr><td colspan="5">version</td><td colspan="2">version</td></tr>
<tr><td colspan="5">xml:lang</td><td colspan="2">xml:lang</td></tr>
<tr><td colspan="5">xmlns</td><td colspan="2">xmlns</td></tr>
<tr><td rowspan="3">annotation</td><td colspan="4">source</td><td colspan="2">source</td></tr>
<tr><td colspan="3">appinfo</td><td></td><td>Comment
<<appinfo>></td></tr>
<tr><td>documentation</td><td>xml:lang</td><td></td><td>xml:lang</td><td>Comment
<<documentation>></td></tr>
<tr><td rowspan="2">attributeGroup</td><td colspan="3">name</td><td colspan="2">name</td><td rowspan="2">Class
<<attributeGroup>></td></tr>
<tr><td>annotation</td><td>appinfo</td><td></td><td></td><td>Comment
<<appinfo>></td></tr>
</table>

XSD					UModel			
			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>>		
		simpleType			name (= name of Class + "_anonymousType[n]")	DataType <<simpleType>>		
	element	name			name		Class <<element>>	
		abstract			abstract			
		block			block			

XSD					UModel				
		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 <<all>>	
			annotation	appinfo			Comment <<appinfo>>		
				documentation			Comment <<docum		

© 2018 Altova GmbH

XSD					UModel				
						entation>>			
			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>>			
				process Contents	process Contents				
			choice			Property			
						Class <<choice>>			
			sequence			Property			
						Class <<sequence>>			
		sequence			name (= "_sequence")		Property		
					name (= "mg"_ + "sequence")		Class <<sequence>>		
			annotation	appinfo		Comment <<appinf			

XSD					UModel				
						o>>			
				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				
		final			final				
		mixed			mixed				

© 2018 Altova GmbH

XSD					UModel			
		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			
			type					
		attributeGroup	ref		type	Property <<attribu		

XSD					UModel			
						teGroup >>		
		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			
		maxExclusive	value		value		<<maxExclusive>>	
			fixed		fixed			
		maxInclu	value		value		<<maxIn	

© 2018 Altova GmbH

XSD					UModel			
		sive	fixed		fixed	clusive>>		
		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>>		
		minExclusive	value		value	<<minExclusive>>		
			fixed		fixed			

XSD					UModel				
		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>>		
			processContents		processContents				
		simpleType					DataType		

© 2018 Altova GmbH

XSD					UModel				
							<<simple Type>>		
		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>>	

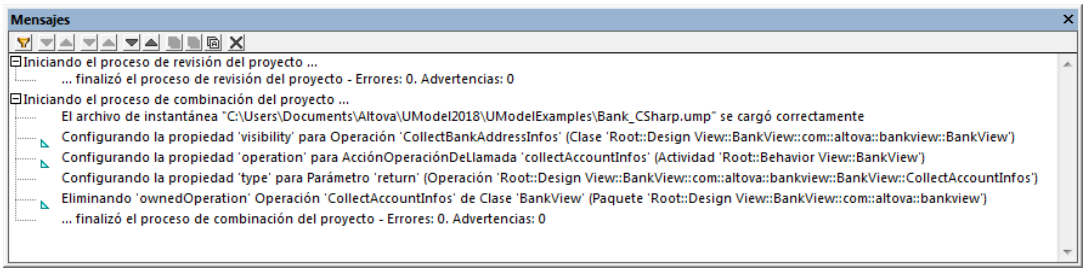
7.7 Combinar proyectos de UModel

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 proyecto principal.
   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*.

¿Qué efectos tiene la fusión de proyectos a 2 bandas?

- **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 de un atributo, así que se usa el valor predeterminado del segundo modelo.
- **Diferencias entre diagramas**: UModel comprueba si hay diferencias entre los diagramas de los dos modelos.

  - Si hay diferencias, los diagramas nuevos/diferentes 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. Después puede eliminar los diagramas que quiera.
- El proceso de combinación entero se puede deshacer 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 proyecto principal combinado conserva el mismo **nombre de archivo**.

 © 2018 Altova GmbH

7.7.1 Fusión de proyectos a 3 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 3 bandas*.

La fusión de proyectos a 3 bandas sirve para combinar proyectos de UModel de nivel superior, es decir, proyectos principales que pueden contener subproyectos. La fusión a 3 bandas no funciona con archivos independientes si tienen referencias sin resolver a otros archivos.

Cuando se combinan proyectos principales, sus subproyectos editables también se combinan automáticamente. Es decir, no hace falta combinar los subproyectos por separado. Para ver un ejemplo consulte el apartado *Ejemplo de fusión manual a 3 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 (el que se eligió como destino de la combinación) conserva el mismo **nombre de archivo**.

¿Qué efectos tiene la fusión de proyectos a 3 bandas?

Es importante tener en cuenta que cuando decimos "proyecto original", nos referimos al primer archivo de proyecto que se eligió como destino de la combinación 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 (resultado de la fusión).
  - si el proyecto original tiene elementos que no existen en el segundo proyecto, estos elementos se conservan en el proyecto de destino (resultado de la fusión).
- **Elementos de modelado eliminados:**
  - si en el segundo proyecto se eliminaron elementos que todavía existen en el proyecto original, estos elementos se eliminan en el proyecto de destino (resultado de la fusión).
  - si en el proyecto original se eliminaron elementos que todavía existen en el segundo proyecto, estos elementos se conservan en el proyecto de destino (resultado de la fusión).
- **Diferencias entre los mismos elementos de modelado:**
  - si una propiedad (p. ej. el nivel de acceso de una clase) se modificó en el proyecto original o en el segundo proyecto, el modelo de destino incluirá el valor más reciente.
  - si una propiedad (p. ej. el nivel de acceso de una clase) se modificó tanto en el archivo original como en el segundo archivo, el modelo de destino incluirá el valor del segundo archivo (y aparece una advertencia en la ventana Mensajes).
- **Elementos con una posición distinta:**
  - si un elemento cambió de posición en el proyecto original o en el segundo proyecto, en el modelo de destino también se cambia la posición del elemento.
  - si un elemento cambió de posición tanto en el proyecto original como en el segundo proyecto, aparece un aviso y el usuario puede seleccionar el elemento primario donde

se debe insertar el elemento en el modelo de destino.

- **Diferencias entre diagramas**:
  UModel comprueba si hay diferencias entre los diagramas de los dos modelos.
  - si hay diferencias, el diagrama nuevo o diferente 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. Después puede eliminar los diagramas que quiera.

Sistemas de control de versiones para fusiones a 3 bandas

Cuando trabaje con un sistema de control de versiones y proteja/desproteja archivos de proyecto, UModel genera automáticamente archivos de instantánea denominados *archivo antecesor común,* que se utilizan para el proceso de fusión a 3 bandas y que permiten una combinación mucho más precisa que la fusión a 2 bandas.

La fusión a 3 bandas **automática** de UModel no funciona con todos los sistemas de control de versiones, pero en casos así puede usar la fusión a 3 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 3 bandas automática de UModel.
- Los sistemas de control de versiones que piden que el usuario elija entre **Reemplazar** o **Combinar** cuando cambia un archivo de 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 el proceso de protección/desprotección de los archivos es necesario usar UModel.
- Puede poner bajo control de versiones tanto el proyecto principal como sus subproyectos. Si cambia los datos de un subproyecto, recibirá un aviso automático para desproteger el subproyecto.
- Cada acción de protección/desprotección crea un *archivo antecesor común* o de instantánea que se utiliza después durante el proceso de fusión a 3 bandas.

Nota: los archivos de instantánea solo se crean y emplean automáticamente 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 un archivo de instantánea para cada usuario, el historial de edición de instantáneas permite combinar los diferentes cambios en el proyecto. Como resultado del proceso de fusión a 3 bandas, el nuevo nombre de la clase y su nuevo nivel de acceso se pasan al archivo de proyecto de destino (resultado de la fusión).

 © 2018 Altova GmbH

7.7.2 Ejemplo de fusión manual a 3 bandas

Este ejemplo muestra una sencilla fusión de proyecto a 3 bandas. Imaginemos que dos usuarios distintos (Tomás y Alicia) crearon cada uno una copia de un proyecto de UModel y realizaron modificaciones. Ahora existen, por tanto, tres versiones distintas del mismo proyecto: la original, la copia de Tomás y la copia de Alicia. En el contexto de una fusión a 3 bandas, el proyecto original representa el *archivo antecesor común*.

Ahora imaginemos que el archivo antecesor común es el proyecto **Bank_CSharp.ump** (disponible en la carpeta **C:\Usuarios\<usuario>\Documentos\Altova\UModel2018\UModelExamples**). Las copias de Tomás y Alicia deben crearse a mano. Por tanto, crearemos primer estas dos copias del proyecto **Bank_Csharp.ump** en carpetas secundarios dentro de la carpeta **...\UModelExamples**. Las carpetas secundarias recibirán el nombre de **C#_Alicia** y **C#_Tomás**. No es necesario modificar el nombre del archivo de proyecto.

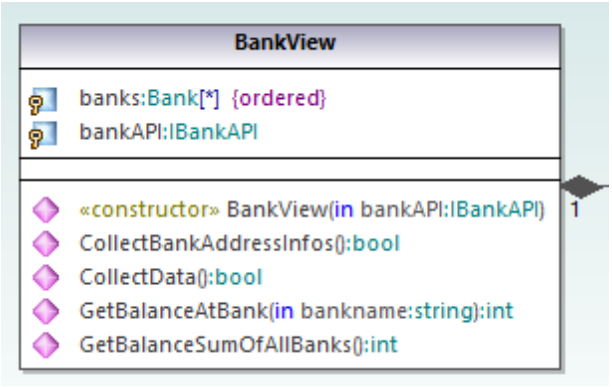
Consejo: use el comando **Proyecto | Guardar como** para crear las copias de Tomás y Alicia. Cuando la aplicación pregunte si desea ajustar las rutas de acceso relativas, haga clic en **Sí**. Así evitará errores de sintaxis en las copias del proyecto.

El objetivo de este ejemplo es que Alicia combine los cambios no solo del original sino también del proyecto de Tomás en un modelo nuevo (la denominada fusión a 3 bandas).

Paso nº1: preparar el proyecto de Tomás

Tomás abre el archivo de proyecto **Bank_CSharp.ump** de la carpeta **C#_Tomás**, abre el diagrama "BankView Main" y realiza cambios en la clase `BankView`.

1. Tomás elimina la operación `CollectAccountInfos():bool` de la clase `BankView`.
2. Tomás cambia el nivel de acceso de la operación `CollectBankAddressInfos():bool` por "public".

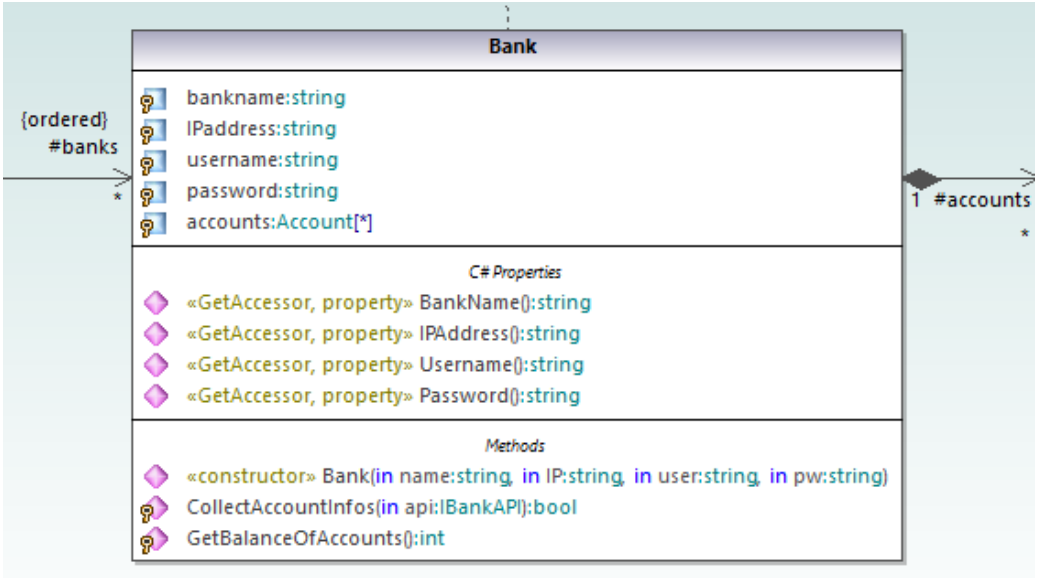


3. Después guarda el proyecto.

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` por "protected".



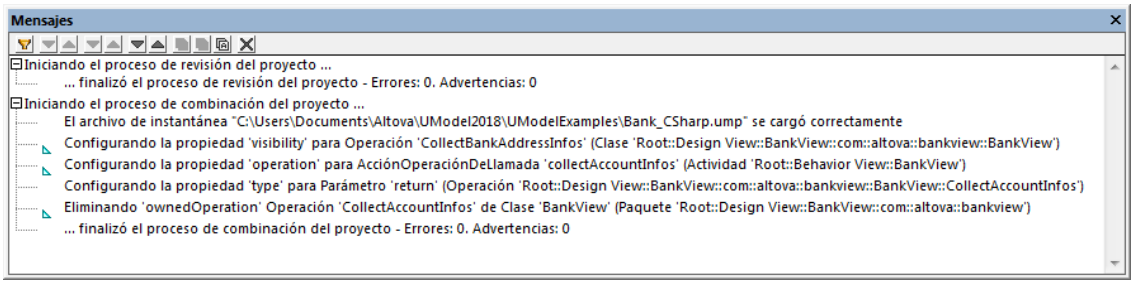
2. Después guarda el proyecto.

Paso nº3: realizar la fusión a 3 bandas

Alicia comienza la fusión a 3 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 3 bandas)** y selecciona el archivo de proyecto de Tomás, que está en la carpeta **C#_Tomás**.
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 3 bandas se inicia y la aplicación vuelve al archivo de proyecto desde el cual se inició la fusión a 3 bandas (es decir, al archivo de proyecto de la carpeta **C#_Alicia**. En la ventana de mensajes aparecen los detalles del proceso de combinación.



Como resultado de la fusión a 3 bandas:

- los cambios que realizó Tomás en el proyecto se recrean 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 3 bandas de los archivos de proyecto de las carpetas **C#_Tomás** y **C#_Alicia**.

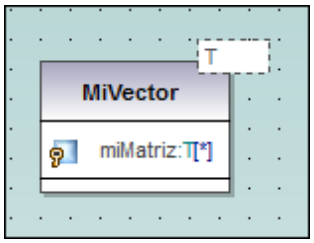
7.8 Plantillas UML

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:

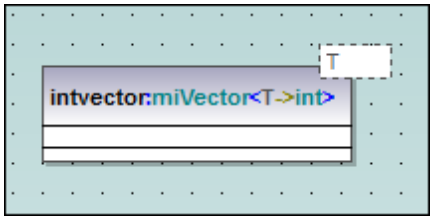
- En la imagen que aparece a continuación puede verse la plantilla de clase **`MiVector`**, cuyo parámetro de plantilla formal (`T`) aparece en la esquina superior derecha en un rectángulo discontinuo.
- Los parámetros formales sin información de tipo (`T`) son clasificadores implícitos: clase, tipo 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.

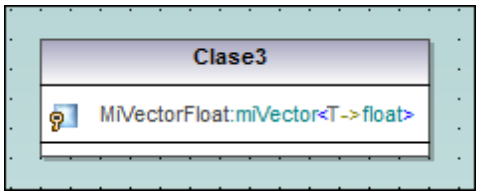
Presentación de enlaces de plantilla en UModel:

- 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.

7.8.1 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>
```

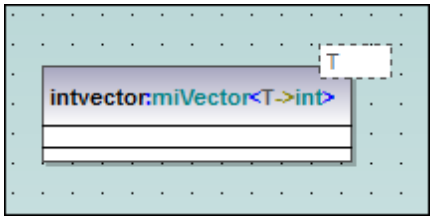
El parámetro de plantilla `T` debe ser de tipo "vector" que contiene objetos que son un supratipo de `unaClaseBase`.

Parámetros de plantilla de extensión

```
T>Comparable<T->T>
```

7.8.2 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)

 © 2018 Altova GmbH

```
vector<T->?>
```

Restringir comodines - límites superiores (extensión de UModel)

```
vector<T->?>unaClaseBase>
```

Restringir comodines - límites inferiores (extensión de UModel)

```
vector<T->?<unaClaseDerivada>
```

7.8.3 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.

Altova UModel 2018

Generar documentación UML

8 Generar documentación UML

Sitio web de Altova: Documentación de proyectos UML

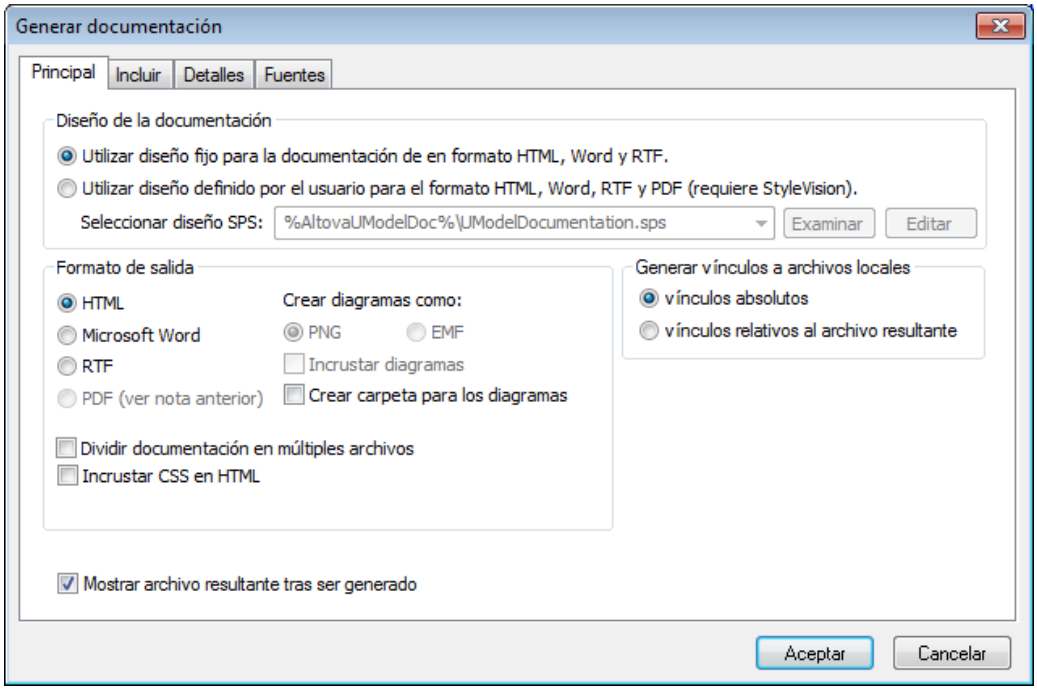
El comando **Generar documentación** genera documentación detallada sobre el proyecto UML en formato HTML, MS Word, RTF y PDF. Una vez generada, la documentación se puede modificar y utilizar sin permiso de Altova.

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 usar una hoja de estilos SPS de StyleVision para personalizarla. Consulte el apartado Hojas de estilos definidas por el usuario para más información.

Nota: si quiere usar una hoja de estilos SPS para generar la documentación, debe tener StyleVision instalado en el equipo. Los elementos relacionados se unen por medio de hipervínculos que permiten navegar de un componente a otro. Para generar documentación en formato Word, necesita tener instalado MS Word 2000 (o superior).

Para este ejemplo utilizaremos el proyecto de muestra `Bank_MultiLanguage.ump`:

1. Abra el proyecto en UModel y seleccione la opción de menú **Proyecto | Generar documentación**.
   Esto abre el cuadro de diálogo "Generar documentación" (*imagen siguiente*) en su configuración predeterminada.



Recuerde que también puede generar documentación de determinados elementos de modelado por separado. Esto se consigue haciendo clic con el botón derecho en uno o varios elementos en la *Estructura del modelo* y seleccionando **Generar documentación** en el menú contextual. El elemento puede ser una carpeta, una clase, una interfaz, etc. En cualquier caso, las opciones de

 © 2018 Altova GmbH

documentación son las mismas en ambos casos.

Los elementos relacionados se unen por medio de hipervínculos que permiten navegar por la documentación de un componente a otro. En la documentación también aparecen todos los hipervínculos creados a mano.

Nota: también se genera documentación para los subproyectos (perfiles) C#, Java y VB **incluidos**, si estos se habilitan en la pantalla *Incluir* del cuadro de diálogo "Generar documentación".

Pestaña *Principal*
A continuación describimos las opciones de esta pestaña.

Diseño de la documentación

- *Utilizar diseño fijo para....* seleccione esta opción si quiere usar la plantilla integrada para generar la documentación.
- *Utilizar diseño definido por el usuario...* seleccione esta opción si quiere usar una hoja de estilos de StyleVision predefinida. Los archivos SPS están disponibles en la carpeta `...\Documentos\Altova\UModel2018\Documentation\UModel\`.
- **Examinar:** haga clic en este botón para buscar el archivo SPS predefinido.
- **Editar:** haga clic en este botón para abrir el SPS seleccionado en StyleVision.

Formato de salida:

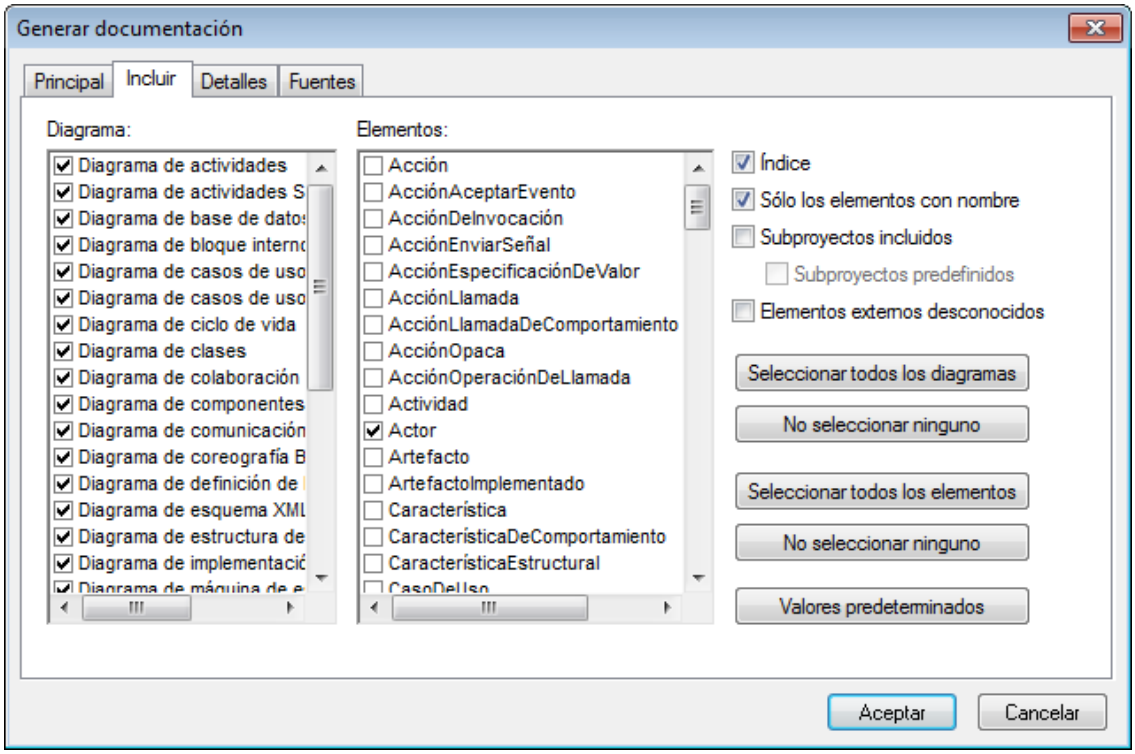
- Aquí se selecciona el formato de salida: *HTML*, *Word*, *RTF* o *PDF*.
  Los documentos de **Microsoft Word** se crean con la extensión de archivo **`.doc`** si utiliza un diseño fijo y con la extensión **`.docx`** si utiliza un archivo SPS.
  El formato **PDF** solamente está disponible si usa un archivo SPS para generar la documentación.
- *Dividir documentación en múltiples archivos:* marque esta casilla si quiere que UModel genere un archivo de salida por cada elemento de modelado de la tabla de contenido (por ejemplo, si hay una clase llamada `C1` que tiene una clase anidada llamada `CNest`, el archivo `C1.html` contiene toda la información sobre `C1` y `CNest`, así como sus atributos, propiedades, etc.).
- *Incrustar CSS en HTML:* marque esta casilla para incrustar un archivo CSS ya existente en la documentación HTML. Si quiere que la referencia al archivo y el archivo CSS propiamente dicho sean externos, desactive esta casilla.
- *Incrustar diagramas:* esta casilla se habilita si el formato de salida elegido es *Microsoft Word* o *RTF*. Si marca esta casilla, los diagramas se incrustan en el archivo generado. Los diagramas se crean como archivo PNG (para HTML) o como archivos PNG/EMF (para MS Word y RTF) y en el archivo de resultado se presentan a través de vínculos de objeto.
- *Crear carpeta para los diagramas:* seleccione esta opción para crear una subcarpeta dentro de la carpeta de salida seleccionada. En esta subcarpeta se guardan todos los diagramas.
- *Mostrar archivo resultante tras ser generado*: esta opción está disponible con todos los formatos de salida. Selecciónela para ver los archivos resultantes en la vista Explorador (para HTML), en MS Word (para formato Word) o en la aplicación predeterminada para archivos RTF y PDF (para RTF y PDF respectivamente).

Generar vínculos a archivos locales

En este grupo de opciones puede indicar si los vínculos generados deben ser absolutos o relativos al archivo de salida.

Pestaña *Incluir*

En esta pestaña (*imagen siguiente*) puede seleccionar qué diagramas y elementos de modelado debe abarcar la documentación.



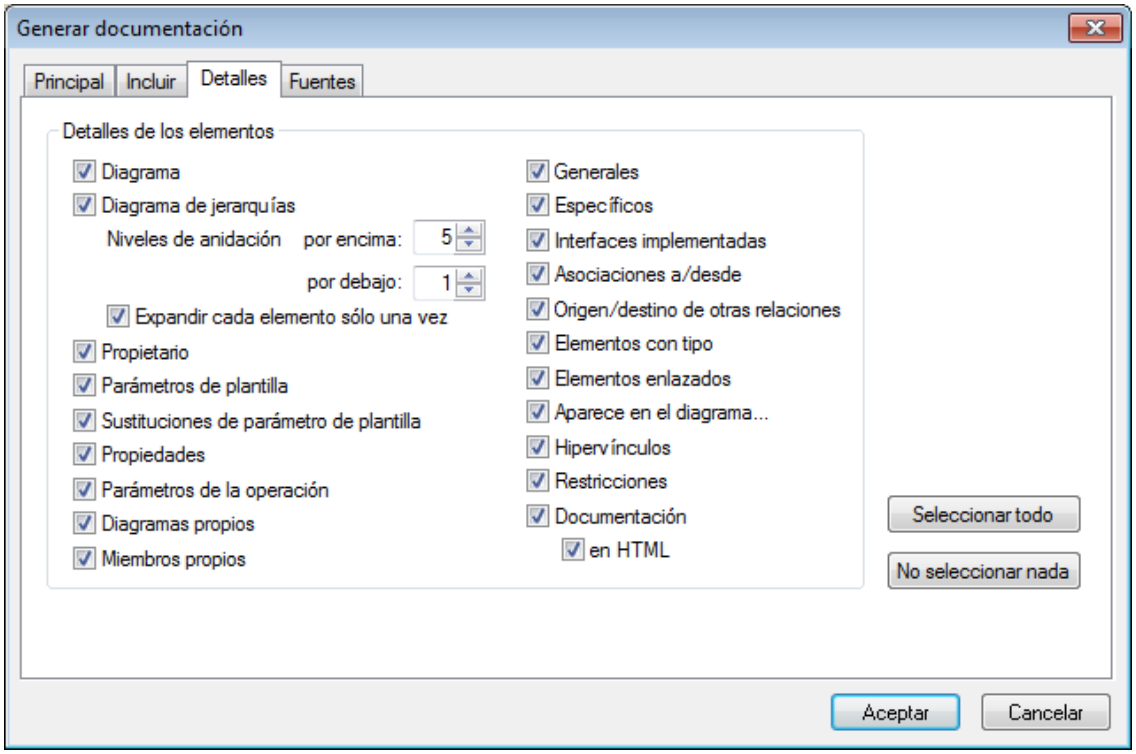
Recuerde que puede deshabilitar la documentación de subproyectos desactivando la casilla *Subproyectos incluidos*.

Pestaña *Detalles*

En esta pestaña (*imagen siguiente*) puede seleccionar los detalles de los elementos que se deben incluir en la documentación.

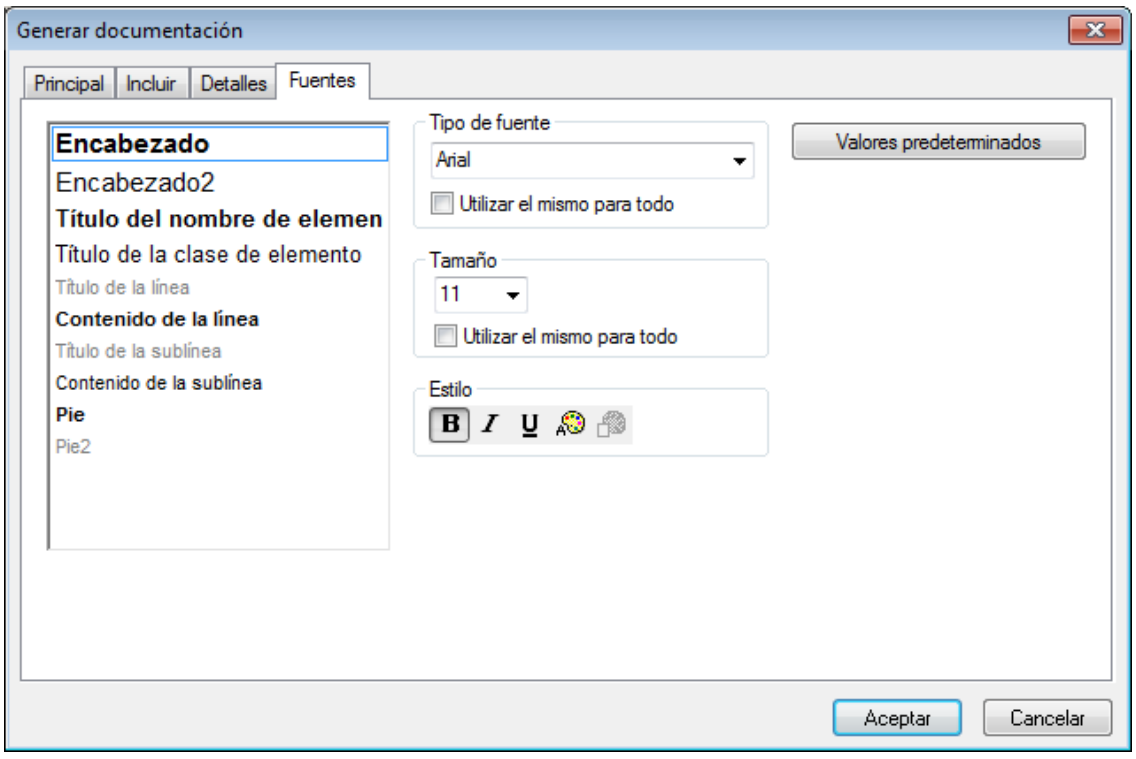
- Si tiene pensado importar el texto de **etiquetas XML** en la documentación, desactive la casilla *en HTML* situada debajo de la opción *Documentación*.
- En los cuadros combinados *por encima* y *por debajo* puede definir cuántos niveles se debe anidar la clase actual en el diagrama de jerarquías.
- Marque la casilla *Expandir cada elemento solo una vez* si quiere que solo un clasificador de cada tipo se pueda expandir en la misma imagen/en el mismo diagrama.

© 2018 Altova GmbH



Pestaña *Fuentes*

En esta pestaña (*imagen siguiente*) puede configurar la fuente utilizada para el contenido de texto y para los títulos y encabezados de la documentación.



© 2018 Altova GmbH

Las imágenes que aparecen a continuación muestran la documentación generada para el archivo `Bank_MultiLanguage.ump` (directorio `...\UModelExamples`) utilizando el diseño fijo.

La primera imagen muestra la documentación generada con los índices de diagramas y de elementos al principio del archivo HTML.

La segunda imagen muestra los detalles de la clase **`Account`** y sus relaciones con otras clases. Observe que los atributos y propiedades de los diagramas de clases también están unidos a sus definiciones por medio de hipervínculos. Haga clic en una propiedad para ver su definición. Las clases de la jerarquía también incluyen hipervínculos (así como el texto subrayado).

Bank_MultiLanguage.ump

ubicación del proyecto C:\Documents\Altova\UModel2014\UModelExamples\Bank_MultiLanguage.ump

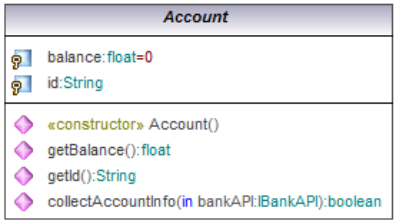
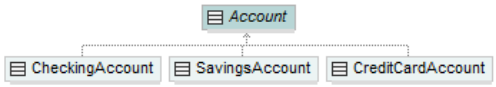
Índice de diagramas:

Diagrama de actividades	collectData Draft		
Diagrama de procesos de negocio BPMN 1	DiagramadeprocesosdenegocioBPMN11		
Diagrama de componentes	Bank realizations	Overview	
Diagrama de estructura de un compuesto	Account Transfer		
Diagrama de implementación	Deployment		
Diagrama de secuencia	Collect Account Information	Connect to BankAPI	
Diagrama de máquina de estados	BankAPI Draft	Query BankServer Draft	

Índice de elementos:

Actividad	Actividad1	BankView	
Artefacto	BankAddresses.ini	BankAPI.jar	BankServer
Colaboración	Account Transfer	Bank Account Transfer	
Componente	BankView GUI		
Interacción	Collect Account Information	Connect to BankAPI	
Nodo	Bank	Home PC	My Home PC
Paquete	Account Transfer Deployment View Root	BankView Design View	Behavior View Interaction View
MáquinaDeEstados	BankAPI	BankServer	

Clase **Account**

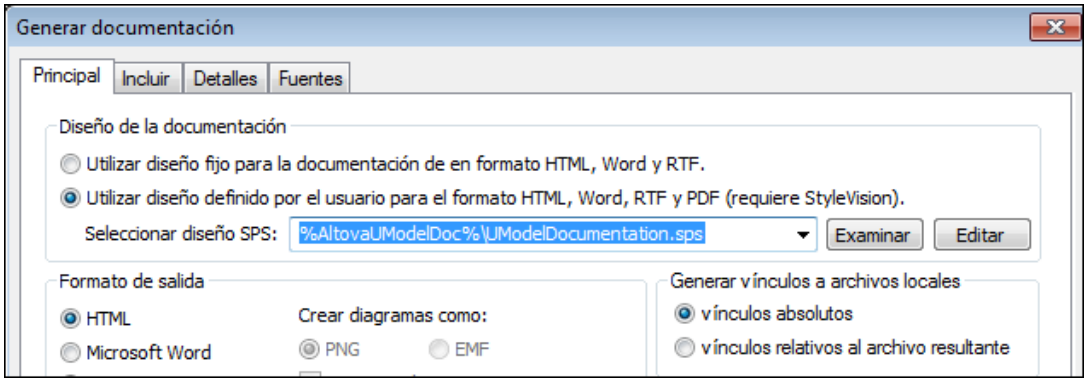
diagrama	
jerarquía	
propietario	**bankview**
propiedades	nombre completo **Design View::BankView::com::altova::bankview::Account** nivel de acceso **public** leaf **false** abstract **true** isFinalSpecialization **false** activo **false** nombre del archivo de código **Account.java** ruta de acceso del archivo de código **C:\Documents\Altova\UModel2014\UModelExamples\Bank_MultiLanguage_Java\Src\Account.java** «annotations» **false** «static» **false** «strictfp» **false**
miembroPropio	**Account balance collectAccountInfo getBalance getId id**
específico	**CheckingAccount CreditCardAccount SavingsAccount**
asociaciones desde	desde Nombre de la asociación **Bank (accounts)**
destino de la relación	RealizaciónDeComponente **BankView**
elementosConTipo	Clase **Bank** Propiedad **accounts** Interacción **Collect Account Information** Propiedad **b**
aparece en el diagrama	**BankView Main Hierarchy of Account**

© 2018 Altova GmbH

8.1 Con una hoja de estilos SPS predeterminada

Para generar documentación con la hojas de estilo SPS predeterminada:

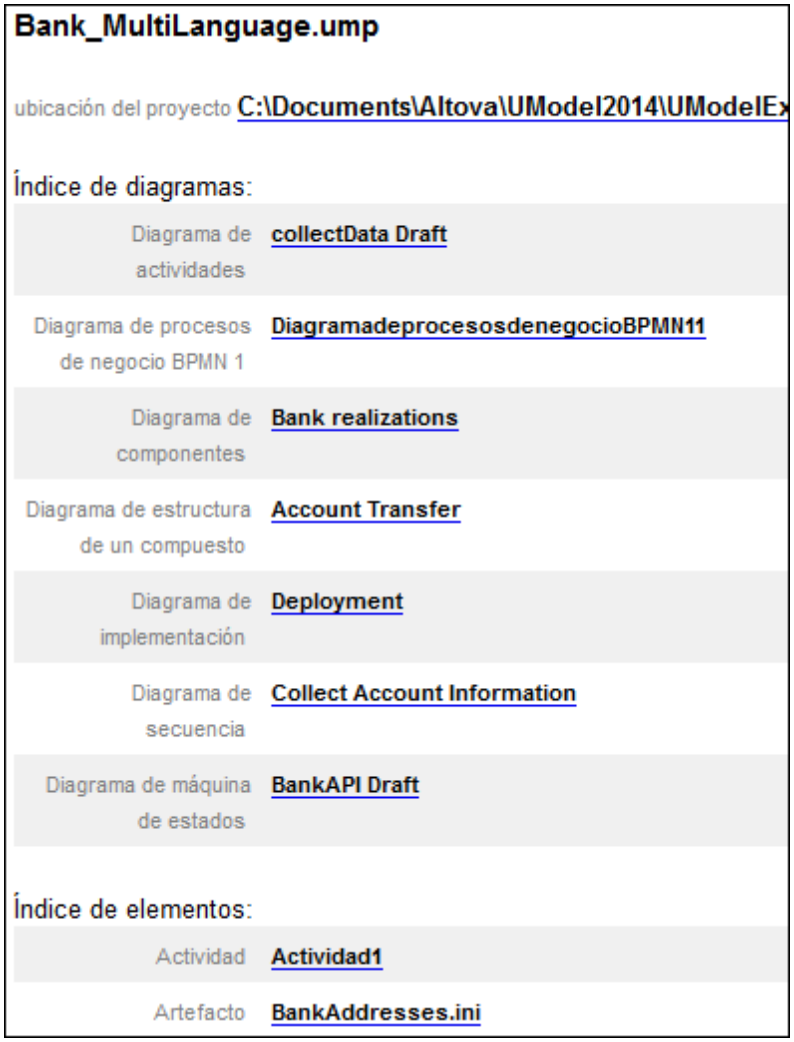
1. Seleccione el comando de menú **Proyecto | Generar documentación.**
2. En el cuadro de diálogo "Generar documentación" seleccione el botón de opción *Utilizar diseño definido por el usuario...* (en la pestaña *Principal*).
3. Seleccione la hoja de estilos `UModelDocumentation.sps` (si no está seleccionada todavía).



Aparece un aviso pidiendo que guarde el archivo.

4. Escriba el nombre del archivo y seleccione la ubicación donde desea guardarlo.

En la documentación generada (*imagen siguiente*) haga clic en un vínculo para navegar al elemento de modelado.



Bank_MultiLanguage.ump

ubicación del proyecto **C:\Documents\Altova\UModel2014\UModelEx**

Índice de diagramas:

Diagrama de actividades	**collectData Draft**
Diagrama de procesos de negocio BPMN 1	**DiagramadeprocesosdenegocioBPMN11**
Diagrama de componentes	**Bank realizations**
Diagrama de estructura de un compuesto	**Account Transfer**
Diagrama de implementación	**Deployment**
Diagrama de secuencia	**Collect Account Information**
Diagrama de máquina de estados	**BankAPI Draft**

Índice de elementos:

Actividad	**Actividad1**
Artefacto	**BankAddresses.ini**

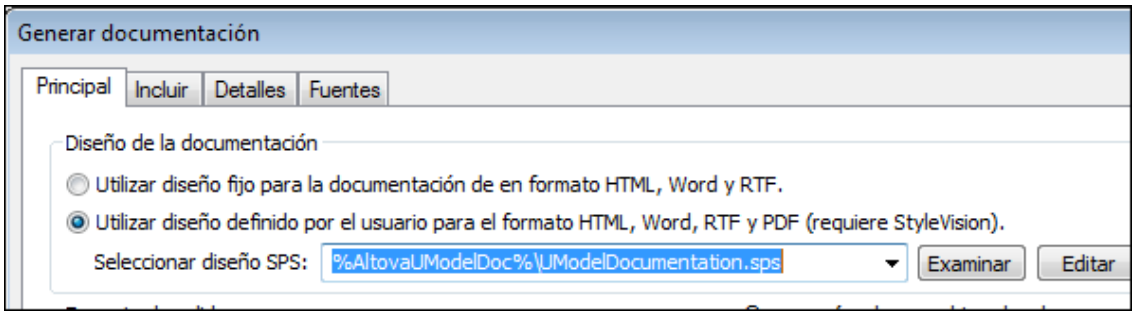
© 2018 Altova GmbH

8.2 Con hojas de estilos predefinidas por el usuario

Si lo prefiere, en lugar de usar el diseño fijo o la hoja de estilos SPS predeterminada, puede crear un diseño personal para su documentación UML. El diseño se crea en una hoja de estilos SPS de StyleVision. Recuerde que también puede editar la hoja de estilos SPS que viene con UModel.

Seleccionar el diseño SPS para generar la documentación

Debe especificar qué diseño SPS se utiliza para generar la documentación en el cuadro de diálogo "Generar documentación" (**Proyecto | Generar documentación**). Seleccione el botón de opción *Utilizar diseño definido por el usuario...* y después seleccione un archivo SPS en el cuadro combinado. Si el SPS no está en el cuadro combinado, haga clic en el botón **Examinar** para buscarlo. El SPS predeterminado disponible en este cuadro combinado es `UModelDocumentation.sps`, que está en la carpeta `...\Documentation\UModel`.



Nota: para poder generar la documentación usando un SPS necesita tener StyleVision instalado en el equipo.

¿Cómo se crea una hoja de estilos SPS?

Las hojas de estilos Power Stylesheet de StyleVision (o archivos SPS) se crean con la aplicación Altova StyleVision. El SPS utilizado para generar documentación UML de UModel debe estar basado en un esquema XML que especifique la estructura del documento XML que almacenará la documentación de UModel.

Se trata del esquema `UModelDocumentation.xsd` que viene con el paquete de instalación de UModel y está disponible en la carpeta `...\Documentos\Altova\UModel2018\Documentation\UModel`.

Para crear el diseño SPS en StyleVision, coloque los nodos del esquema `UModelDocumentation.xsd` en el diseño y asígneles estilos y propiedades. Recuerde que el esquema `UModelDocumentation.xsd` **ya incluye** el archivo `Documentation.xsd` situado en la carpeta superior.

También puede añadir al diseño componentes adicionales, como vínculos e imágenes. Para más información consulte el *Manual del usuario de Altova StyleVision*.

La ventaja de usar diseños SPS para generar la documentación es que permite un mayor control sobre el diseño. Además recuerde que el formato de salida PDF solamente está disponible si utiliza un diseño SPS para generar la documentación. En otras palabras, la documentación no se puede generar en formato PDF si usa el diseño fijo.

Altova UModel 2018

Diagramas UML

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 2.4 y con los diagramas de esquema XML.

- Diagramas de comportamiento: actividades, máquina de estados, máquina de estados de protocolos, casos de uso, interacción, comunicación, interacción global, secuencia y ciclo de vida.
- Diagramas de estructura: clases, estructura de un compuesto, componentes, implementación, objetos y paquetes.
- Otros diagramas: diagramas de esquema XML.

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).

© 2018 Altova GmbH

9.1 Diagramas de comportamiento

Este tipo de diagramas ilustran las características de la conducta de un sistema o proceso de negocio e incluye un subconjunto de diagramas que subrayan cómo interactúan los objetos.

Diagramas de comportamiento

- 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:

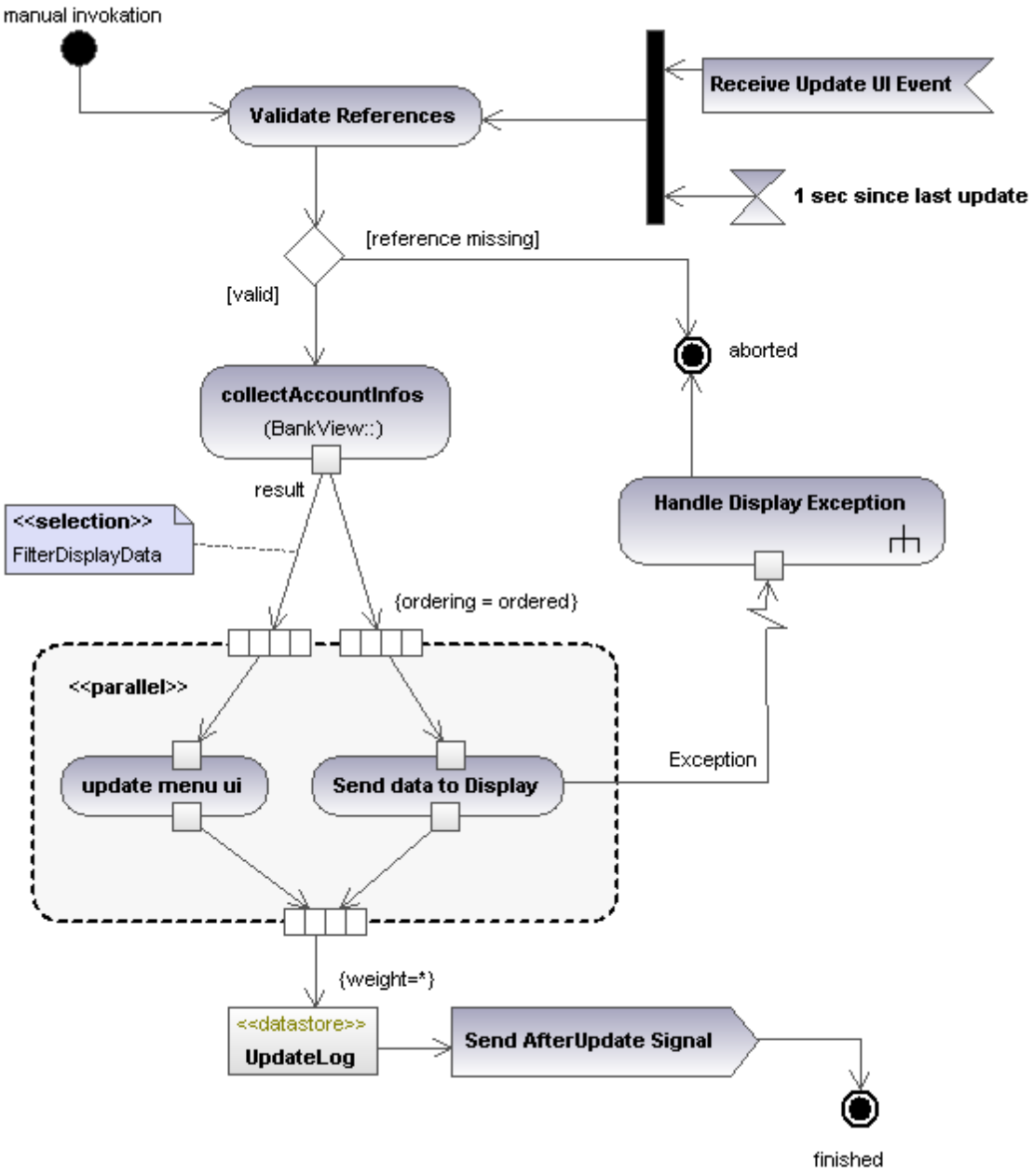
- Diagrama de comunicación
- Diagrama global de interacción
- Diagrama de secuencia
- Diagrama de ciclo de vida

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.

Nota: 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 con los iconos de las barras de herramientas:

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. Si mantiene pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo podrá insertar varios elementos del tipo seleccionado.

 © 2018 Altova GmbH

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 actividades.

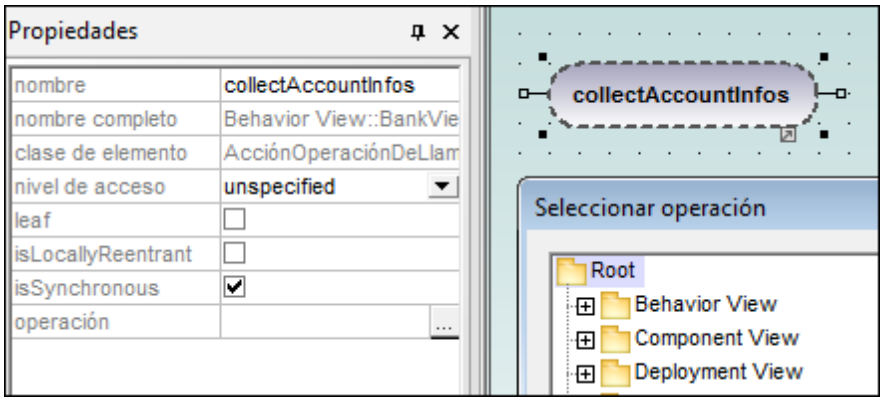
Para insertar una acción (ComportamientoDeLlamada):

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.

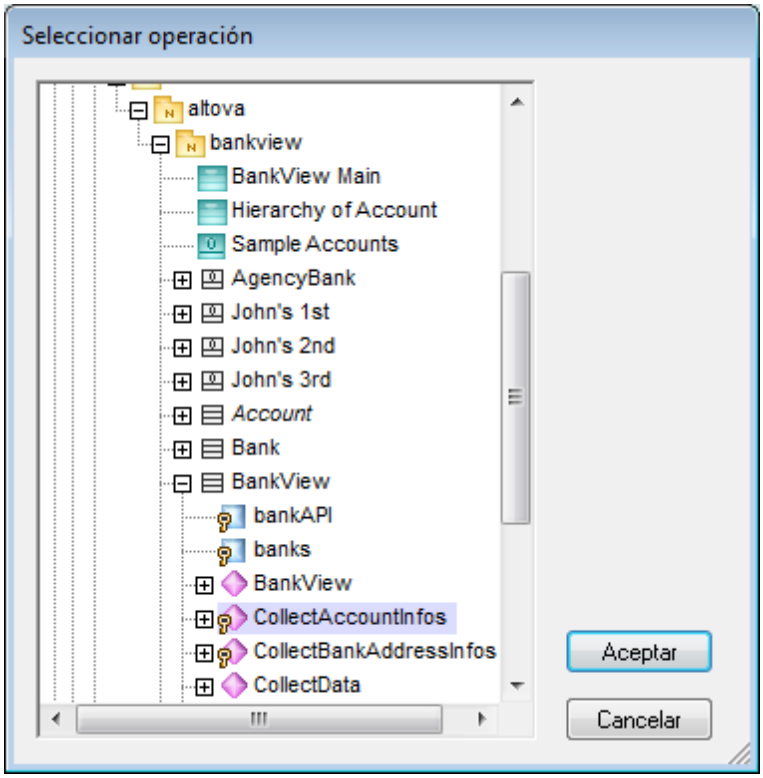
Para insertar una acción (OperaciónDeLlamada) y seleccionar una operación determinada:

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`.

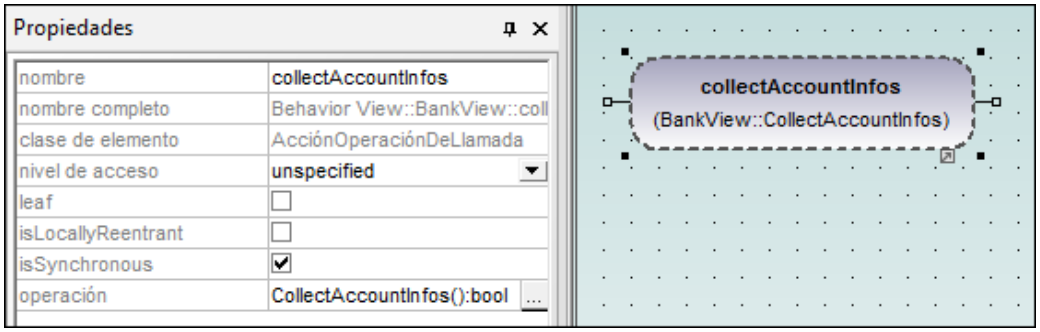


Esto abre el cuadro de diálogo "Seleccionar operación", donde puede seleccionar una operación determinada.

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

Para crear una rama (flujo alterno):

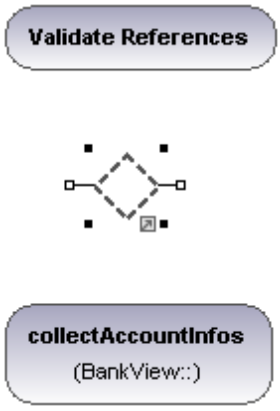
Una rama tiene un flujo de entrada y varios flujos de salida protegidos por guardas. Solo se puede recorrer uno de esos flujos de salida, así que los guardas deben excluirse mutuamente.

En el ejemplo que utilizamos a continuación vamos a validar las referencias de **`BankView`**:

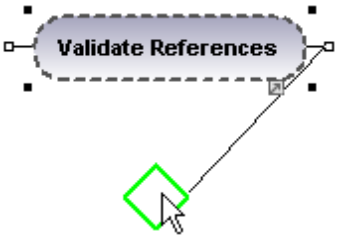
- la rama1 tiene el guarda `reference missing`, que pasa a la actividad `abort`.
- la rama2 tiene el guarda `valid`, que pasa a la actividad `collectAccountInfos`.

 © 2018 Altova GmbH

1. Haga clic en el icono **NodoDeDecisión** de la barra de herramientas y haga clic en el área de trabajo del diagrama de actividades.

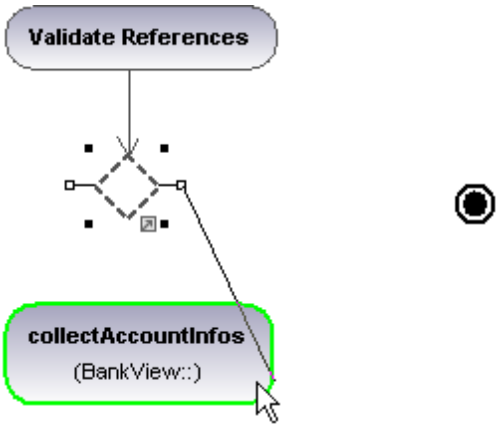


2. Haga clic en el icono **NodoFinalDeActividad** , que representa la actividad `abort`, e insértelo en el diagrama de actividades.
3. Haga clic en la actividad `Validate References` y después haga clic en su conector derecho (el controlador `FlujoDeControl`). Ahora arrastre el conector hasta el elemento `NodoDeDecisión`.

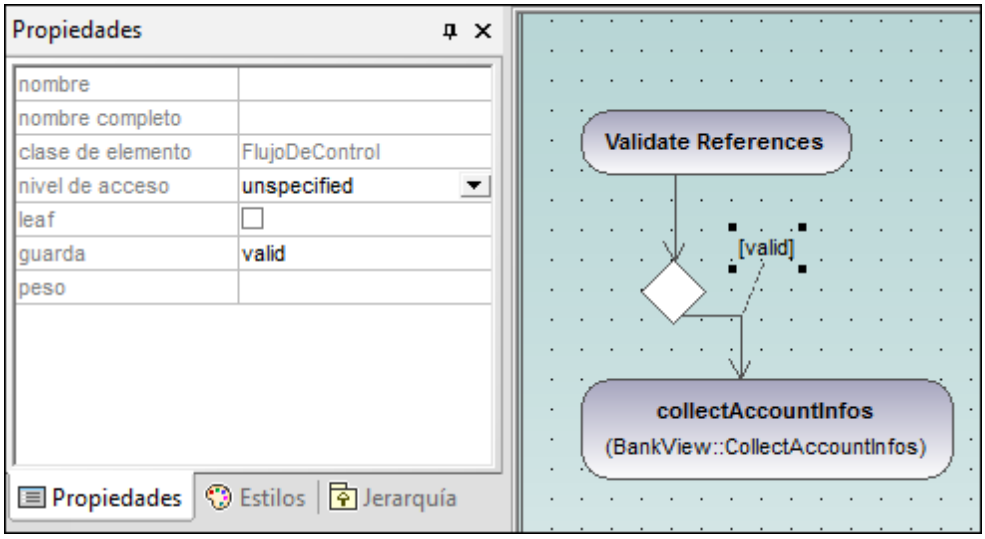


El elemento se resalta cuando sea posible colocar el conector.

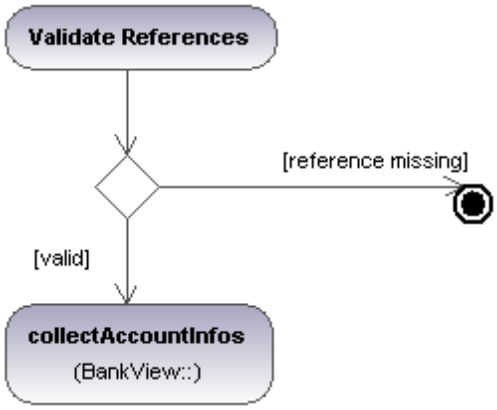
4. Haga clic en el elemento `NodoDeDecisión` y después en su conector derecho (el controlador `FlujoDeControl`). Arrástrelo hasta la acción `collectAccountInfos`. Consulte el apartado Insertar una acción (OperaciónDeLlamada) para obtener más información.



5. En la ventana *Propiedades* seleccione el valor `valid` para la propiedad `guarda`.



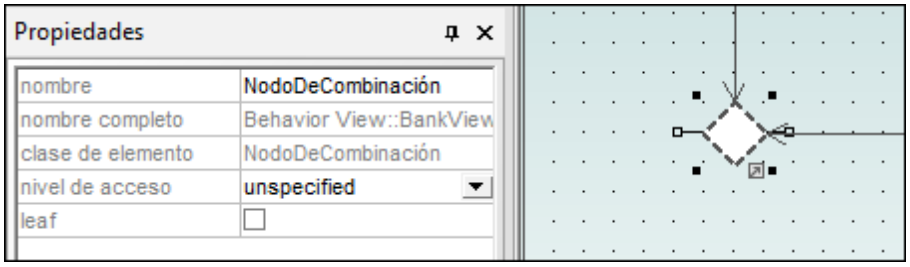
6. Haga clic en el elemento **`NodoDeDecisión`** y después en su conector derecho (el controlador `FlujoDeControl`). Arrástrelo hasta el elemento **`NodoFinalDeActividad`**. La condición de guarda de esta transición se define automáticamente como "else". Haga doble clic en la condición de guarda del diagrama para cambiarla por "reference missing".



Nota: recuerde que UModel no valida ni revisa el número 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.



 © 2018 Altova GmbH

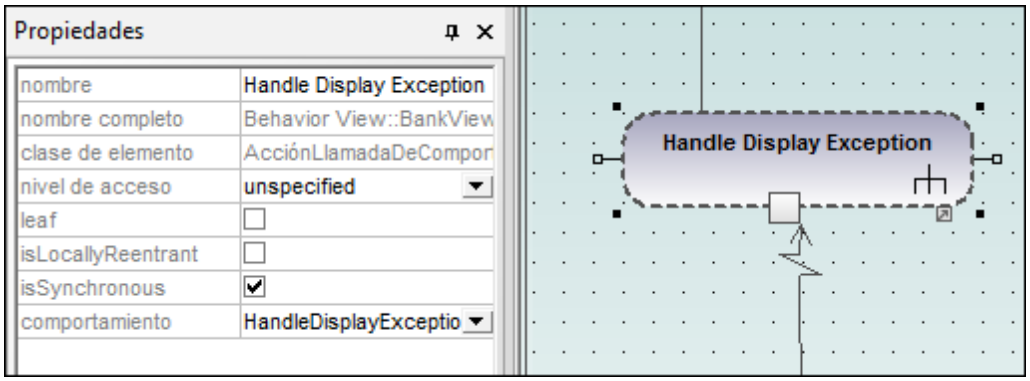
2. Haga clic en el conector `FlujoDeControl` (`FlujoDeObjetos`) de las acciones que desea combinar y arrástrelas hasta el símbolo del `NodoDeCombinación`.

9.1.1.3 Elementos

Acción (ComportamientoDeLlamada)

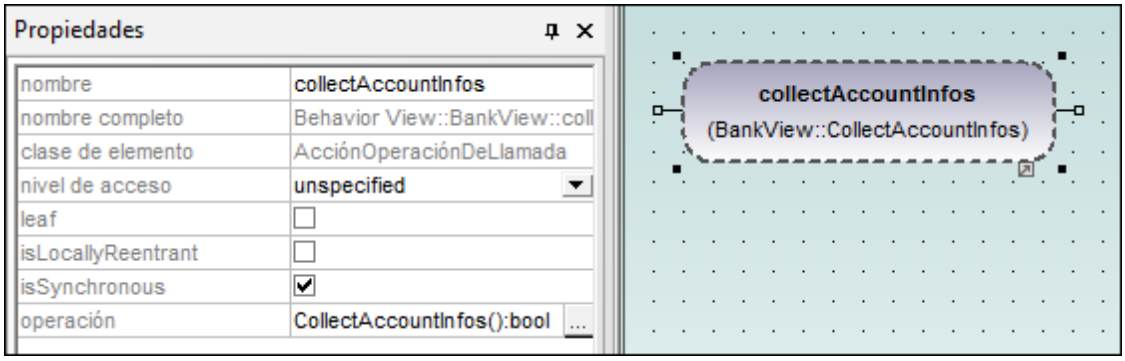
Inserta el elemento **`AcciónComportamientoDeLlamada`**, que invoca un comportamiento concreto directamente.

Si selecciona un comportamiento ya existente 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)

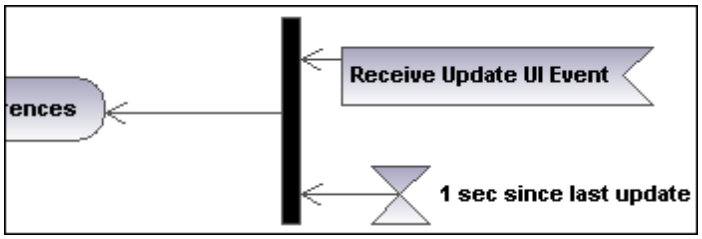
Un tipo de acción que evalúa (o genera) un valor determinado en el pin de salida. Viene definido por las propiedades (p. ej. `upperBound` para el límite superior.)

AcciónAceptarEvento

Inserta la acción **`AceptarEvento`**, que espera que tenga lugar un evento que cumpla determinados requisitos.

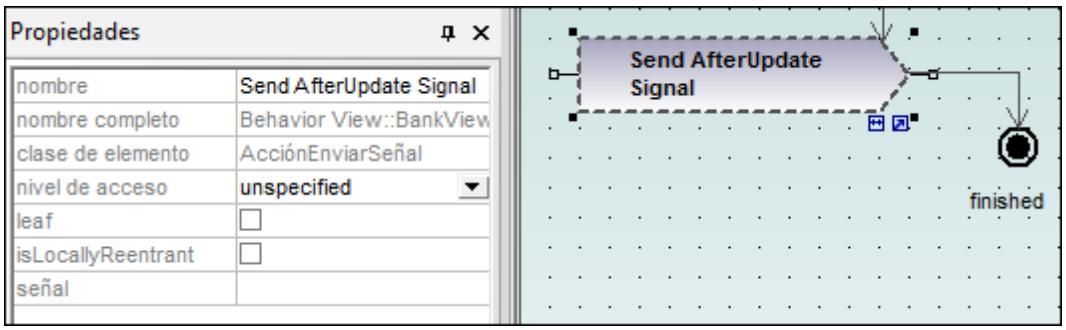
AcciónAceptarEvento (EventoDeTiempo)

Inserta una acción **`AceptarEvento`**, que está desencadenada por un evento de tiempo (un instante de tiempo que viene dado por una expresión).



AcciónEnviarSeñal

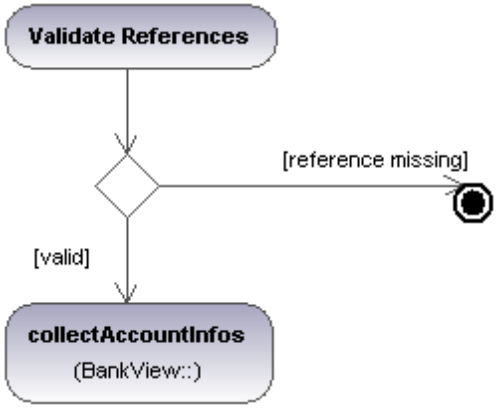
Inserta la acción **`EnviarSeñal`**, que crea una señal desde sus entradas y transmite la señal al objeto de destino, donde puede provocar la ejecución de una actividad.



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.

© 2018 Altova GmbH



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

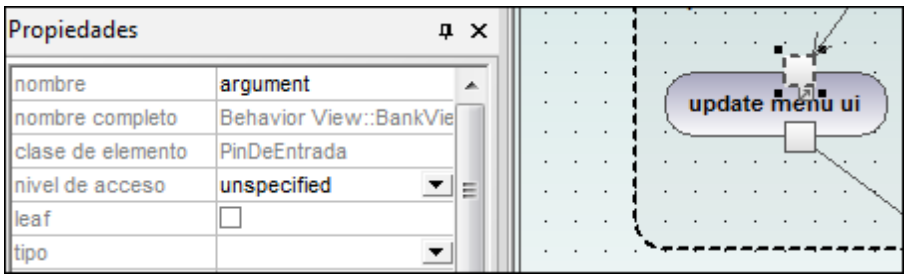
Inserta un nodo de reunión vertical. Sirve para sincronizar varios flujos definidos por un nodo de bifurcación.

NodoDeReunión (horizontal)

Inserta un nodo de reunión horizontal. Sirve para sincronizar varios flujos definidos por un nodo de bifurcación.

PinDeEntrada

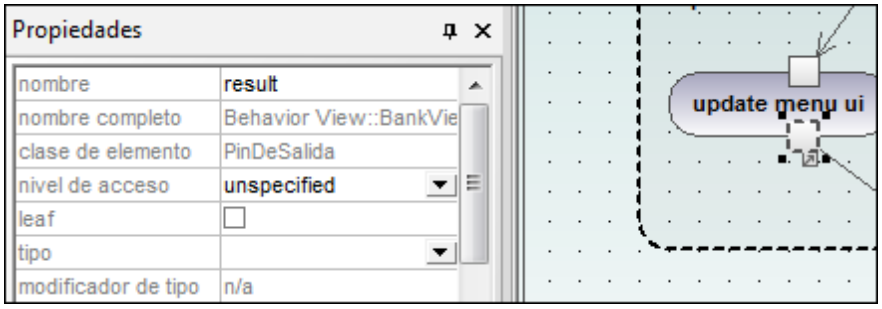
Inserta un pin de entrada en un **`ComportamientoDeLlamada`** o en una **`OperaciónDeLlamada`**. Los pins de entrada aportan los valores de entrada que utiliza la acción. A los pins de entrada se les asigna un nombre predeterminado (argumento) de forma automática.



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

 © 2018 Altova GmbH

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

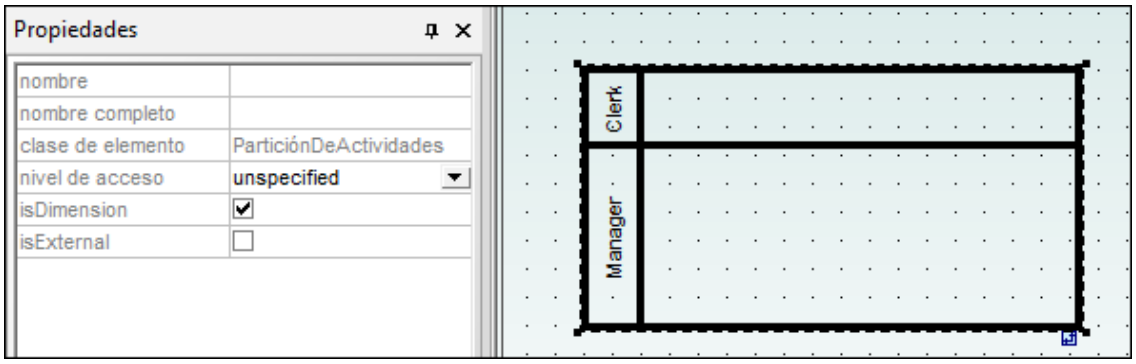
Inserta un nodo de búfer central que funciona de búfer para varios flujos de entrada y salida de otros nodos de objeto.

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.



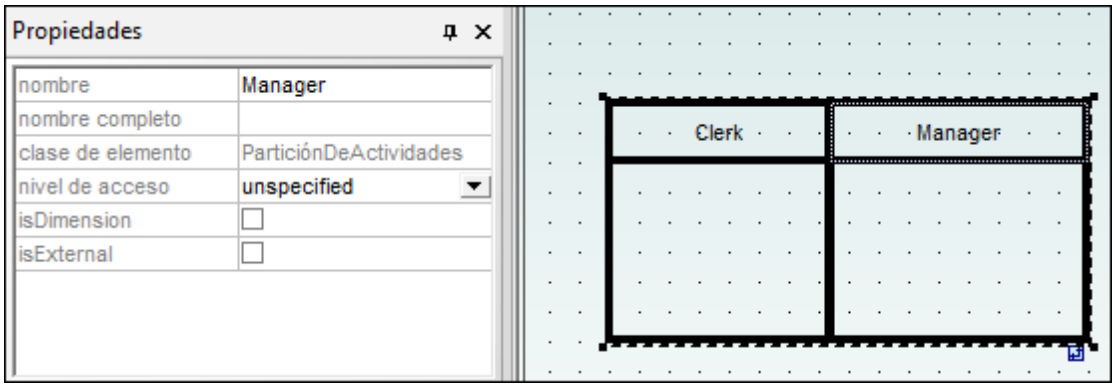
Para editar una etiqueta, haga doble clic en ella. Para orientar el texto correctamente, pulse **Entrar**.

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.

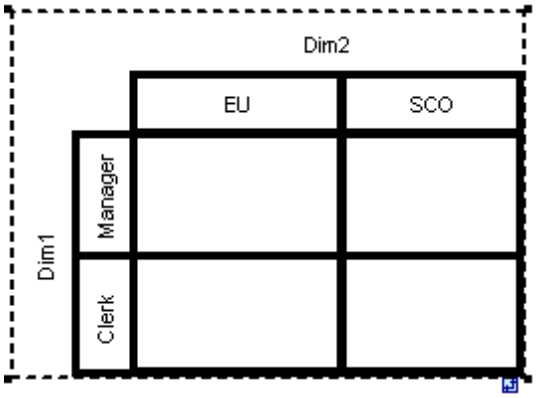
ParticiónDeActividades (vertical)

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)

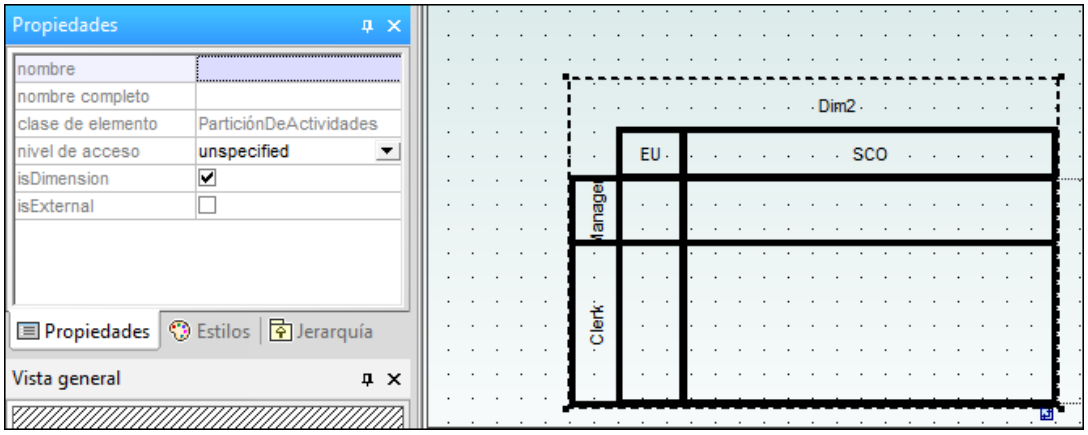
Inserta una partición de actividades bidimensional, un tipo de grupo de actividades que sirve para identificar acciones que tienen características en común. Las etiquetas de los dos ejes son editables.



Para quitar las etiquetas de las dimensiones Dim1 y Dim2:

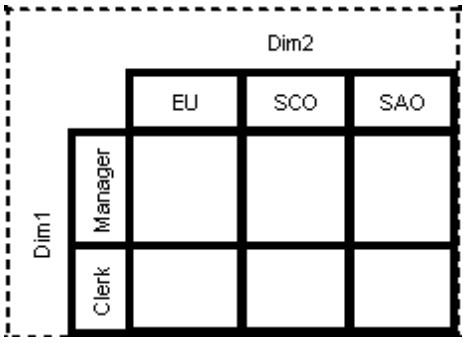
1. Haga clic en la etiqueta que desea eliminar (p. ej. `Dim1`).
2. En la ventana *Propiedades* haga doble clic en la entrada `Dim1`, elimínela y pulse **Entrar** para confirmar.

© 2018 Altova GmbH



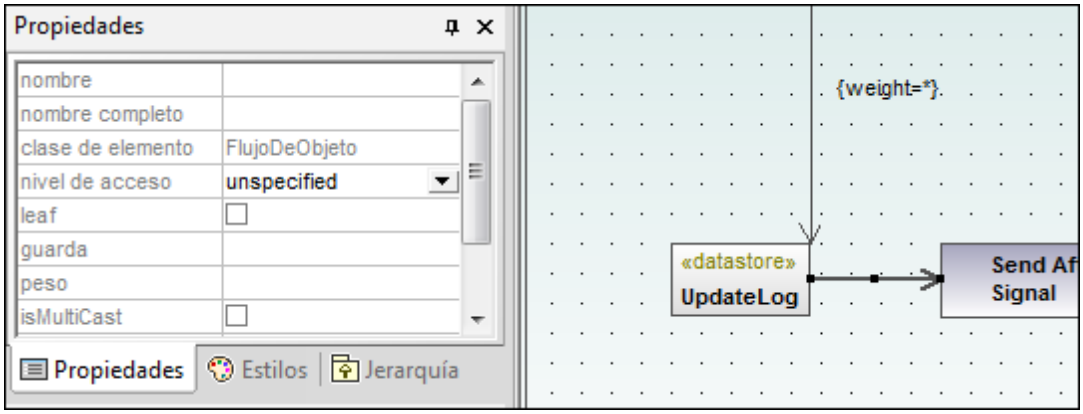
Para anidar particiones de actividades:

1. Haga clic con el botón derecho en la etiqueta de la dimensión donde desea insertar un partición nueva.
2. Seleccione **Nuevo/a | ParticiónDeActividades**.



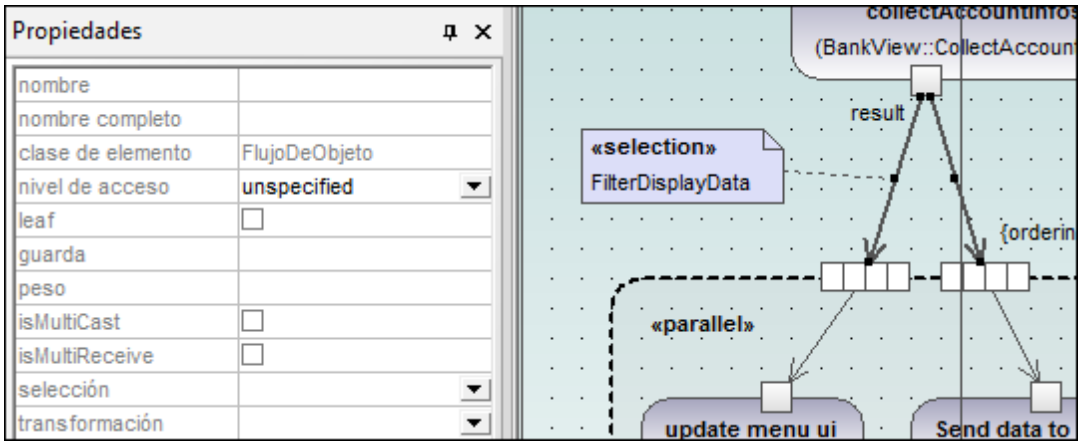
FlujoDeControl

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.



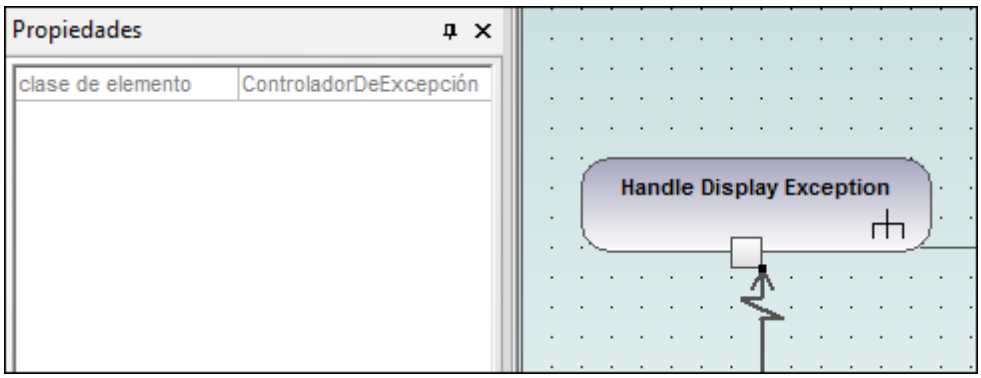
FlujoDeObjeto

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.

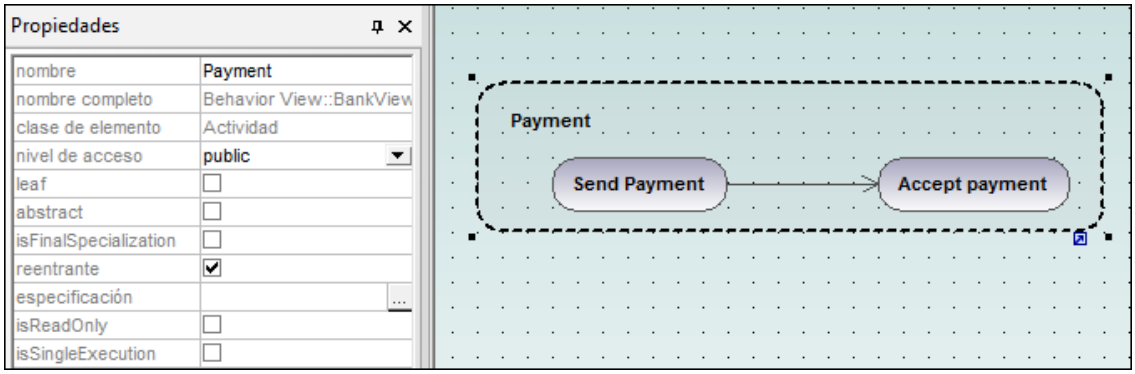


Los controladores de excepción solo se pueden colocar en el pin de entrada de una acción.

Actividad

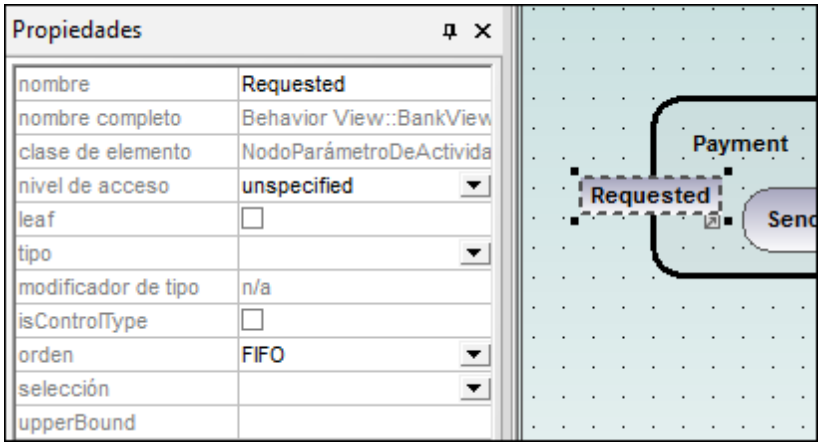
Inserta una actividad en el diagrama de actividades.

© 2018 Altova GmbH



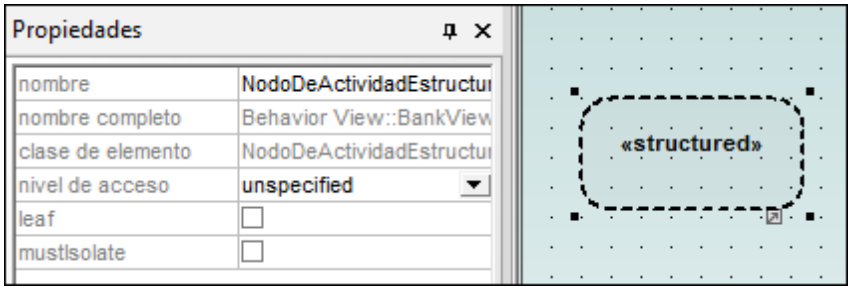
NodoParámetroDeActividad

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.



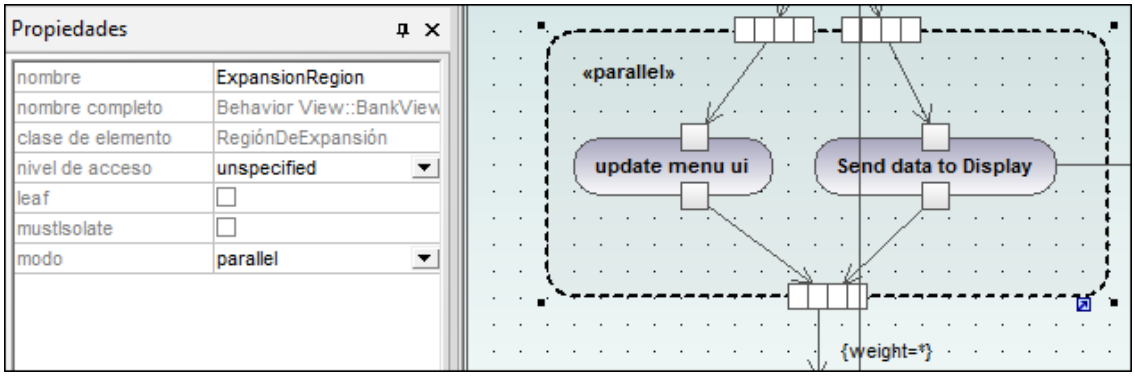
NodoDeActividadEstructurada

Inserta un nodo de actividad estructurada, que es una parte estructurada de la actividad que no se comparte con ningún otro nodo estructurado.



RegiónDeExpansión

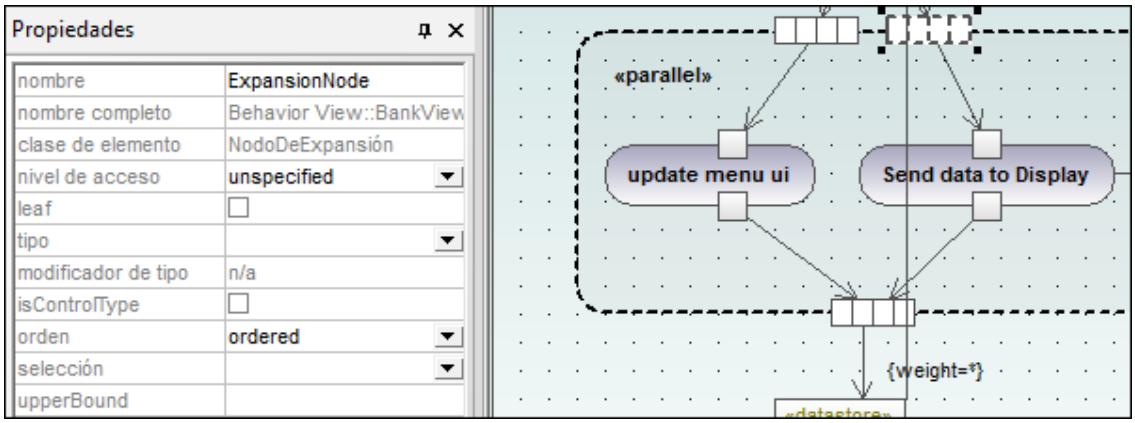
Una región de expansión es una región de una actividad que tiene entradas y salidas explícitas (usando **`NodosDeExpansión`**). Cada entrada es una colección de valores.



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.



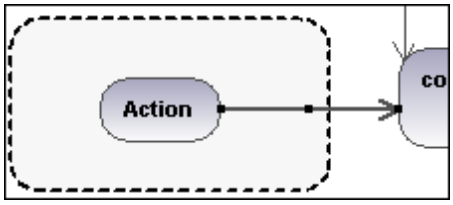
RegiónDeActividadInterrumpible

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.

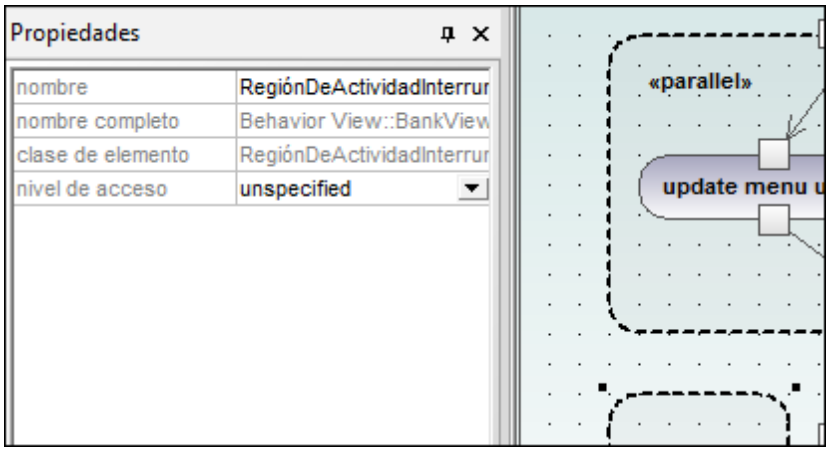
Para añadir un encadenamiento interruptor:

Primero debe comprobar que hay un elemento **`Acción`** en la **`RegiónDeActividadInterrumpible`**, así como un flujo de control de salida hacia otra acción:

 © 2018 Altova GmbH



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).

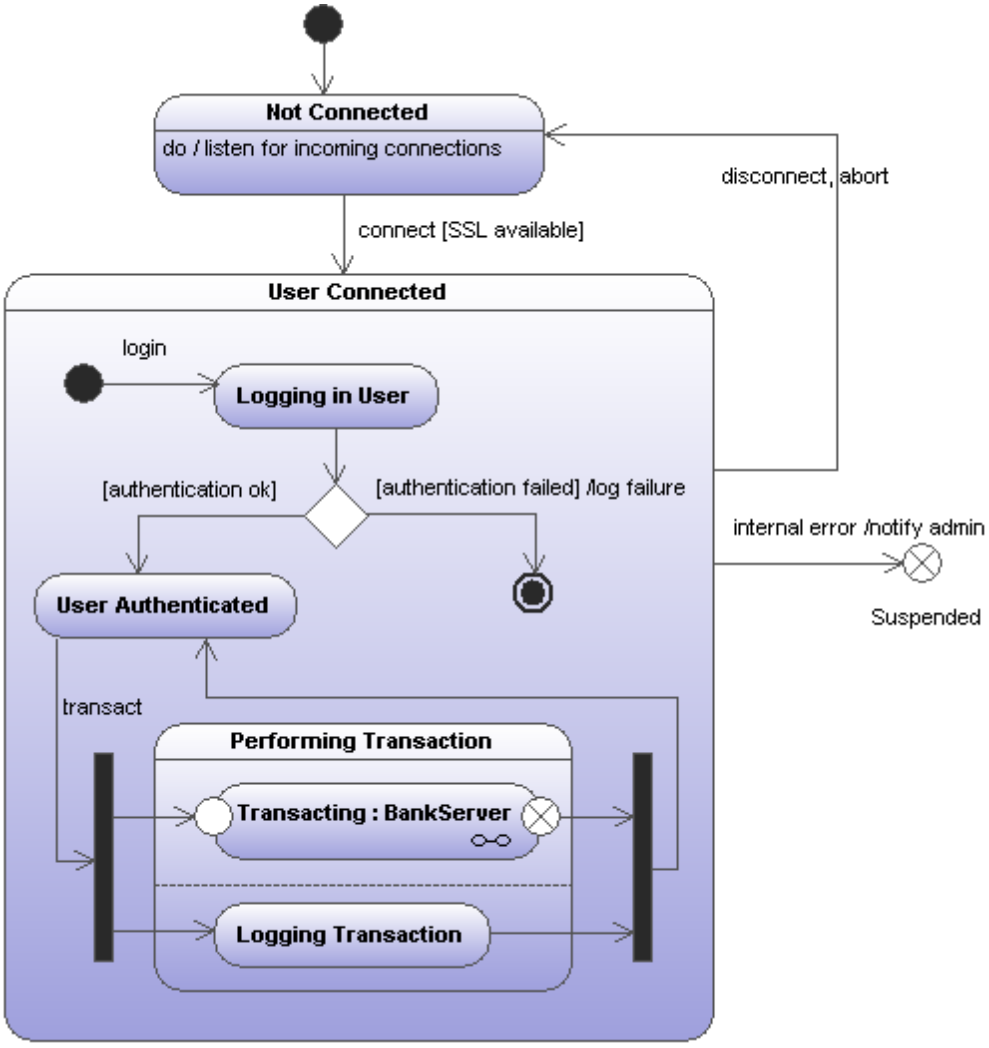


Diagrama de máquina de estados de ejemplo

El diagrama de estados anterior se encuentra en la carpeta del proyecto de ejemplo de UModel **C:\Usuarios\<usuario>\Documentos\Altova\UModel2018\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 un icono de la barra de herramientas Diagrama de máquina de estados.
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.

 © 2018 Altova GmbH

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 máquina de estados.

9.1.2.2 Crear estados, actividades y transiciones

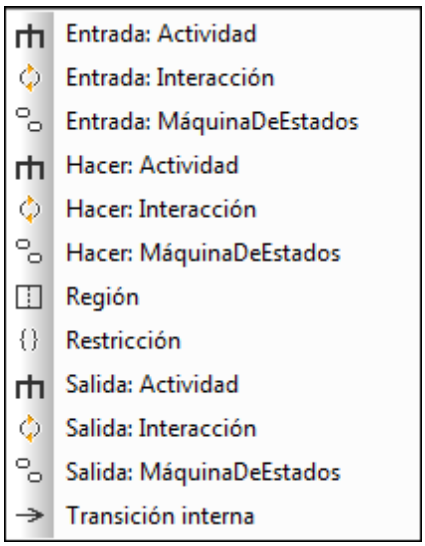
Para agregar un estado simple:

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:

1. Haga clic con el botón derecho en el estado. En el menú contextual seleccione **Nuevo/a** y después una de las opciones del submenú.

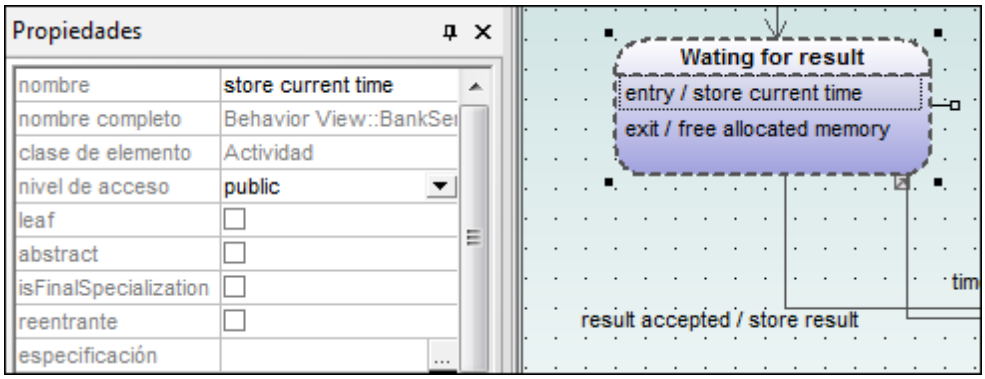
   Las actividades Entrada, Salida y Hacer están asociadas con uno de estos comportamientos posibles: "Actividad", "Interacción" y "MáquinaDeEstados". Por tanto, las opciones de este menú contextual son:



   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`).

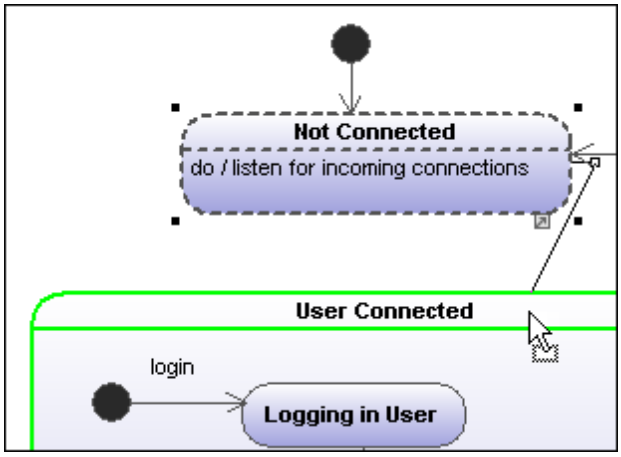


Para eliminar una actividad:

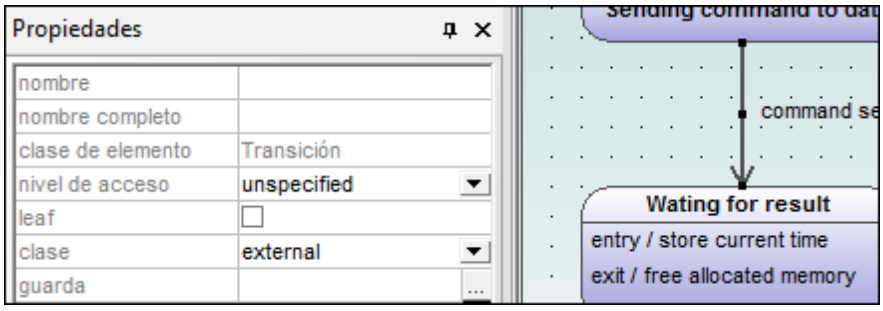
1. Haga clic en la actividad del estado y pulse la tecla **Supr**.

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.



 © 2018 Altova GmbH

Las transiciones pueden tener un disparador de eventos, una condición de protección y una acción con el formato **disparadorEvento [condición de protección] /actividad**.

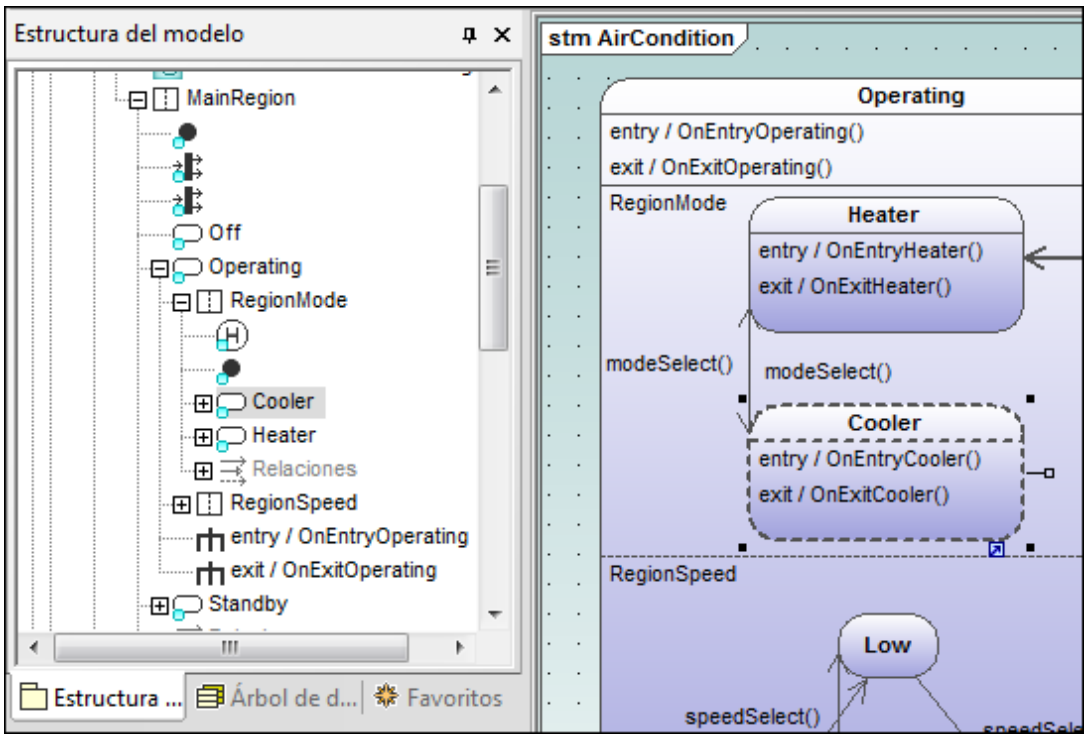
Para crear operaciones desde transiciones automáticamente:

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()`).

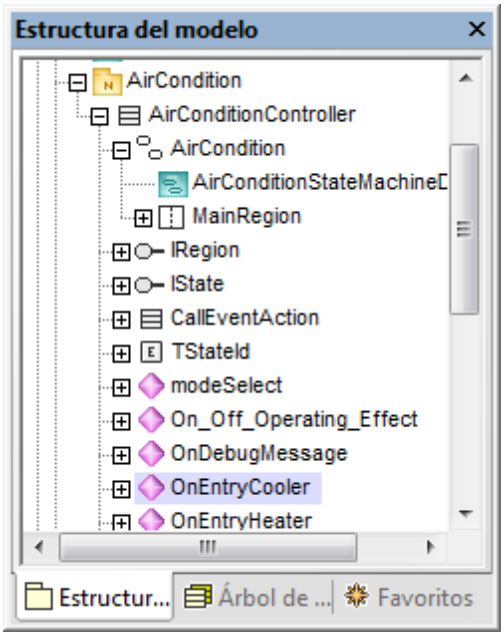
Nota: solamente se pueden crear operaciones automáticamente cuando la máquina de estados está dentro de una clase o de una interfaz.

Para crear operaciones automáticamente desde actividades:

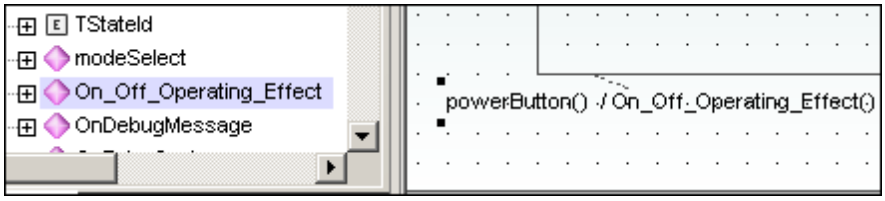
1. Haga clic con el botón derecho en el estado y seleccione la actividad/acción que desea insertar (**Nuevo/a | Entrada:Actividad**).
2. Escriba el nombre de la actividad, asegurándose de que termina con ().



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`.

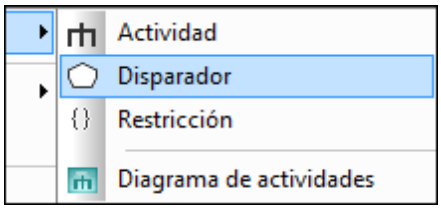


Nota: se añaden operaciones automáticamente para: **Hacer:Actividad, Entrada:Actividad, Salida:Actividad**.



Para crear un disparador de transición:

1. Haga clic con el botón derecho en una transición (en la flecha).
2. Selección **Nuevo/a | Disparador**.



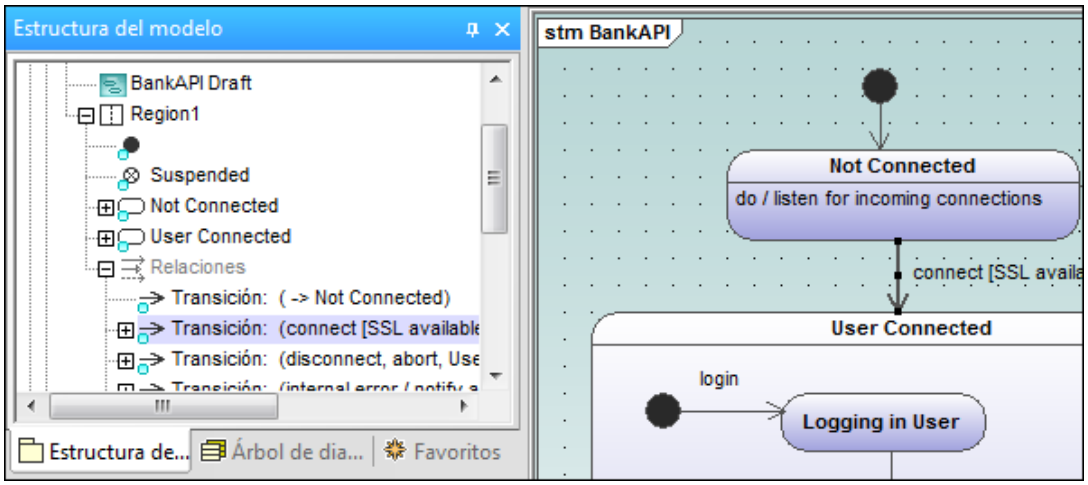
   Si se trata del primer disparador del diagrama, en la etiqueta de la transición, situada encima de la flecha, aparece el carácter "a". Los disparadores tienen asignados valores predeterminados en forma de letra, estado de origen -> estado de destino.
3. Haga doble clic en el nuevo carácter e inserte 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

 © 2018 Altova GmbH

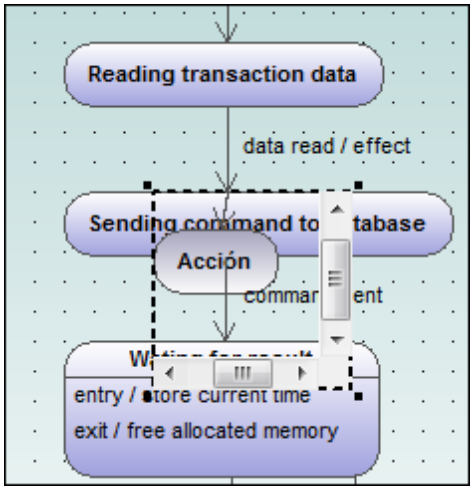
transición y seleccione **Seleccionar en la Estructura del modelo**. El evento, la actividad y los elementos de restricción aparece debajo de la transición seleccionada.



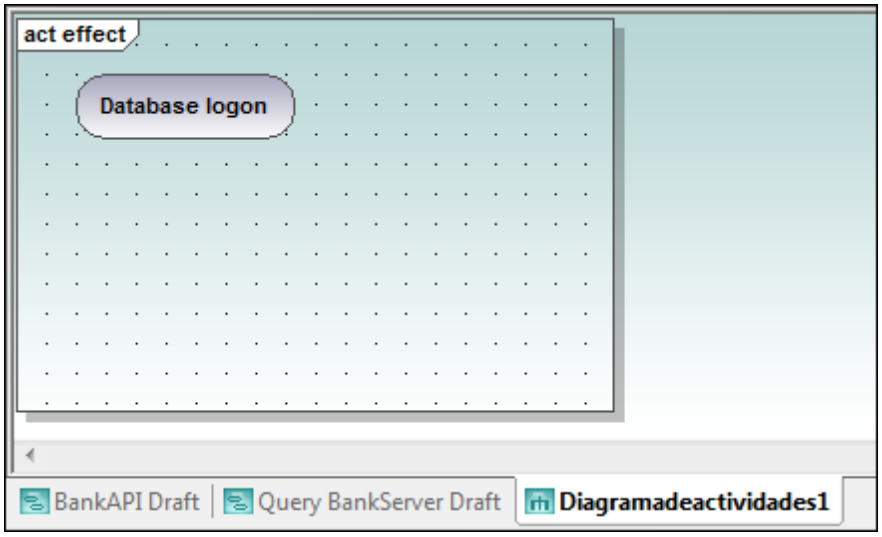
Para agregar un diagrama de actividades a una transición:

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.

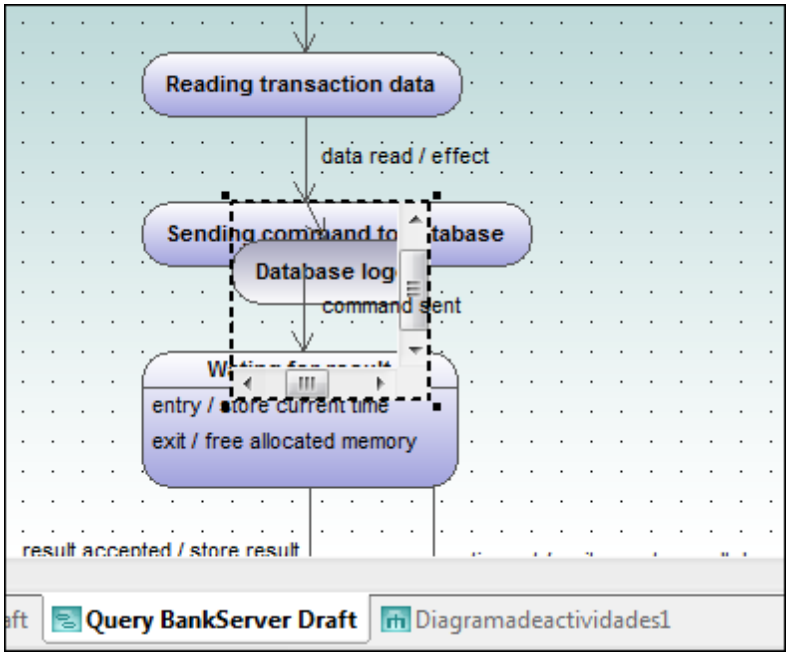


3. Haga doble clic en la ventana para abrir el diagrama de actividades en otra pestaña y seguir definiendo la transición (p. ej. cambiando el nombre de la acción a `Database logon`).



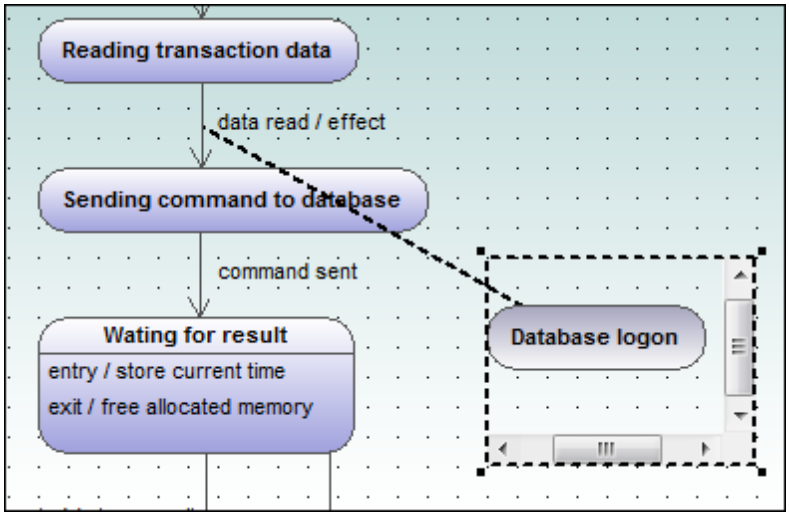
Al proyecto se añade un diagrama de actividades nuevo. Para aprender a añadir elementos de modelado de actividades nuevos al diagrama, consulte el apartado Diagrama de actividades.

4. Haga clic en la pestaña del diagrama de máquina de estados para ver la transición actualizada.

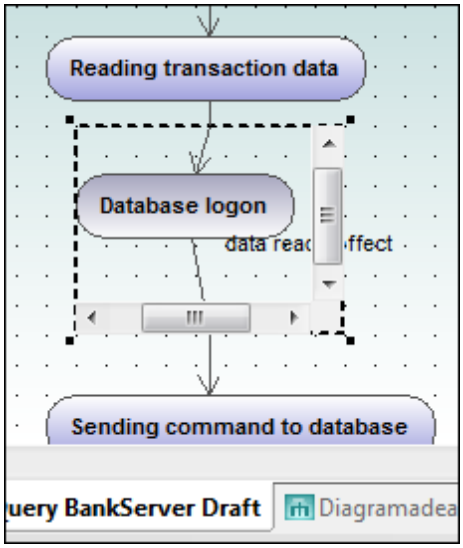


5. Arrastre la ventana de la actividad a una posición nueva donde no moleste y ajuste el tamaño de la ventana si es necesario.

© 2018 Altova GmbH



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 compartimento más, formado por una región. Dentro de esta región puede colocar un número ilimitado de estados.

Para añadir una región a un estado compuesto:

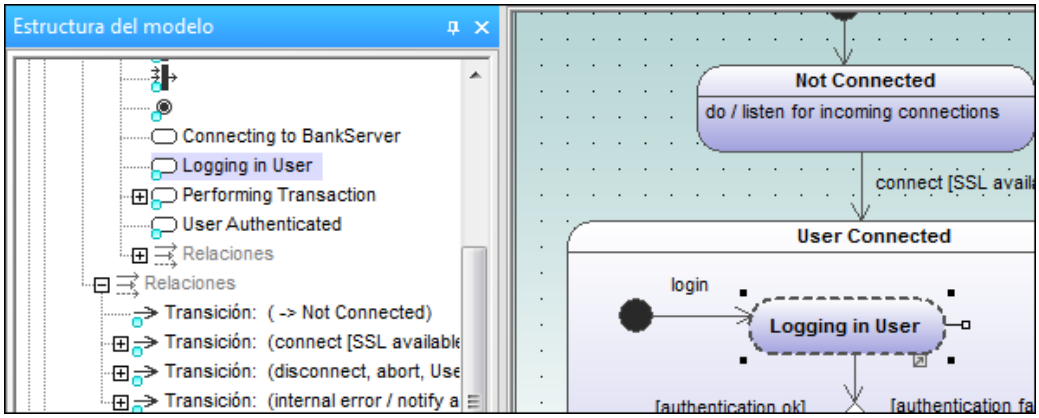
1. Haga clic con el botón derecho en el estado compuesto y seleccione **Nuevo/a | Región** del menú contextual.
   Al estado se le añade una región nueva. Las regiones se dividen con líneas discontinuas.

Para eliminar una región:

1. 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:

1. 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*.

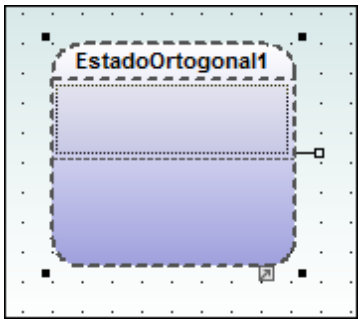


Cuando se mueve el estado compuesto, también se mueven los estados que están dentro de él.

Estado ortogonal

Este tipo de estado contiene un 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 regiones nuevas.



 © 2018 Altova GmbH

Para mostrar/ocultar el nombre de las regiones:

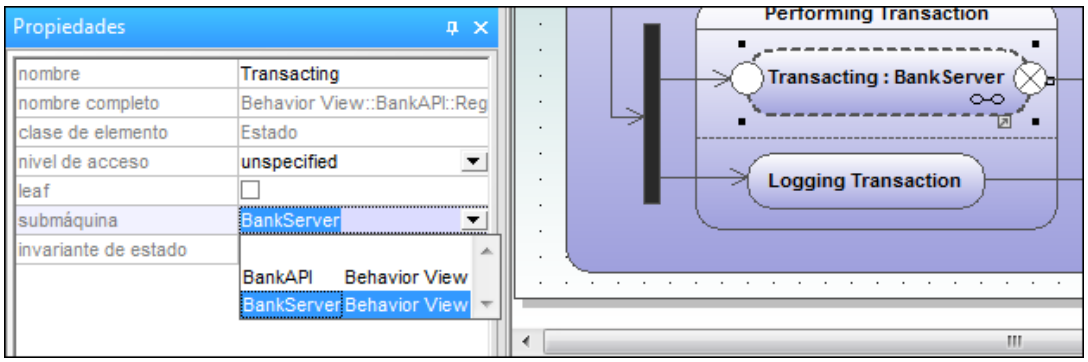
- Haga clic en la ventana *Estilos*, desplácese hasta el estilo `Mostrar los nombres de región en los estados` y seleccione el valor `verdadero/falso`.

Estado de submáquina

Este estado sirve para ocultar los detalles de una máquina de estados. Este estado 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.

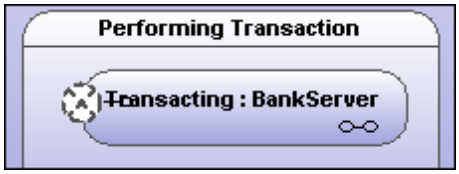


Observe que en la submáquina aparece automáticamente un icono de hipervínculo. Al hacer clic en este icono se abre la máquina de estados a la que se hace referencia (**`BankServer`**, por ejemplo).

Para añadir puntos de entrada/ salida a un estado de submáquina:

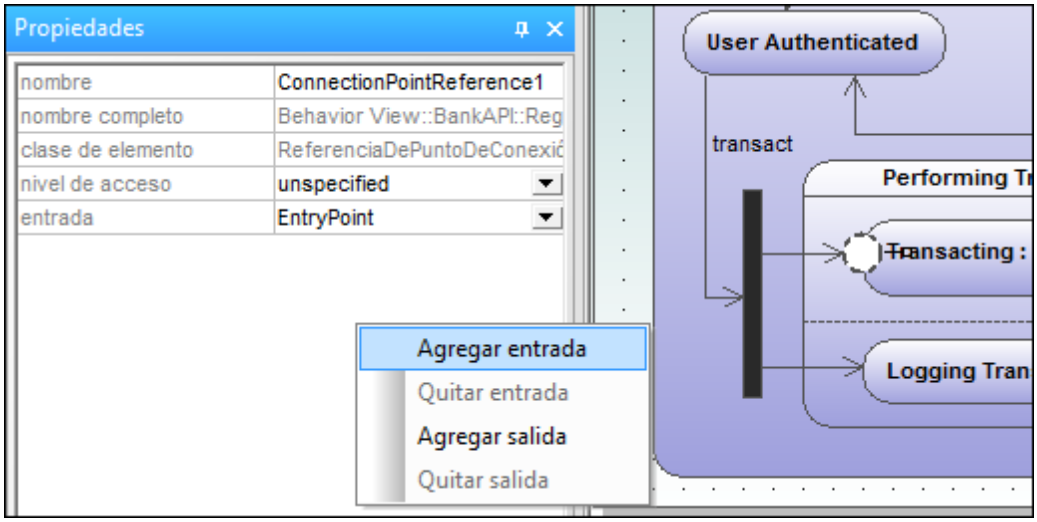
- El estado de submáquina al que está conectado el punto de entrada/salida debe hacer referencia a una máquina de estados (visible en la ventana *Propiedades*).
- Esta submáquina debe contener un punto de entrada y de salida como mínimo.

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 salida.



El comando **Agregar entrada** añade un punto de entrada (`EntryPoint`) nuevo en la ventana *Propiedades* y cambia el aspecto de la referencia de punto de conexión `ConnectionPointReference`.

3. Use el mismo método para insertar un punto de salida (`ExitPoint`) con la opción **Agregar salida** del menú emergente.

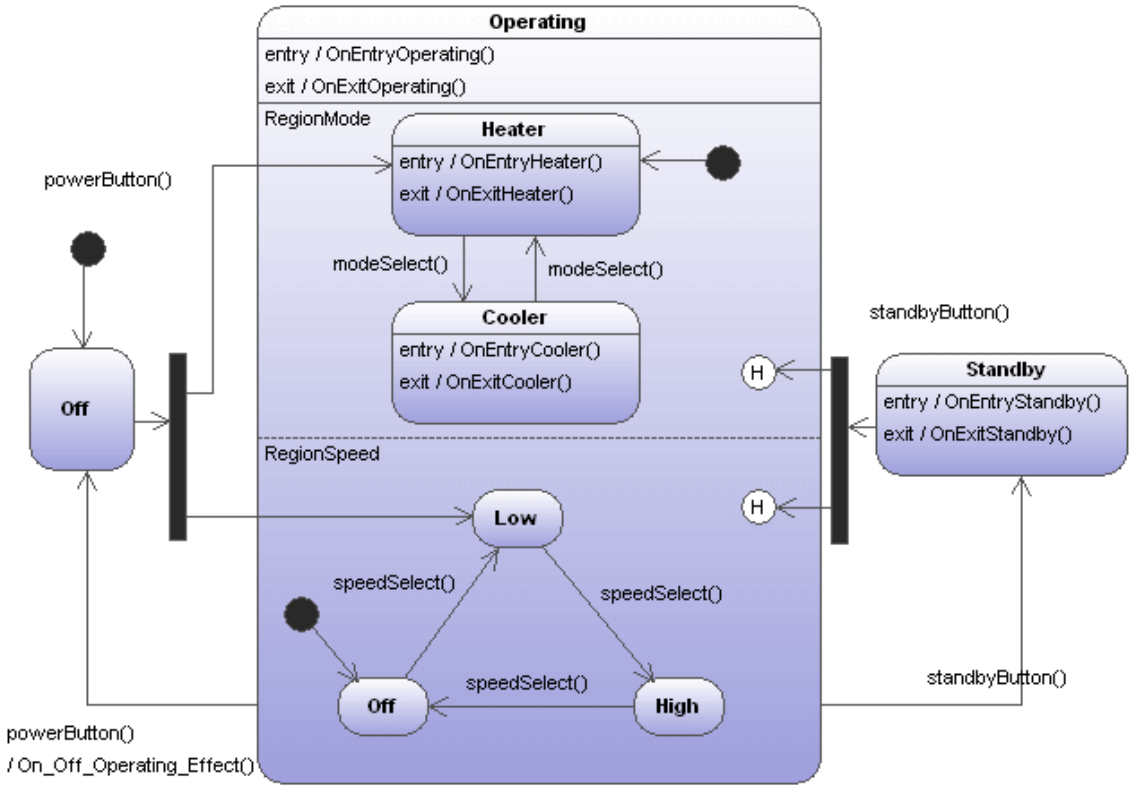
9.1.2.4 Generar código a partir de diagramas de máquina de estados

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 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 durante el proceso de ingeniería directa.

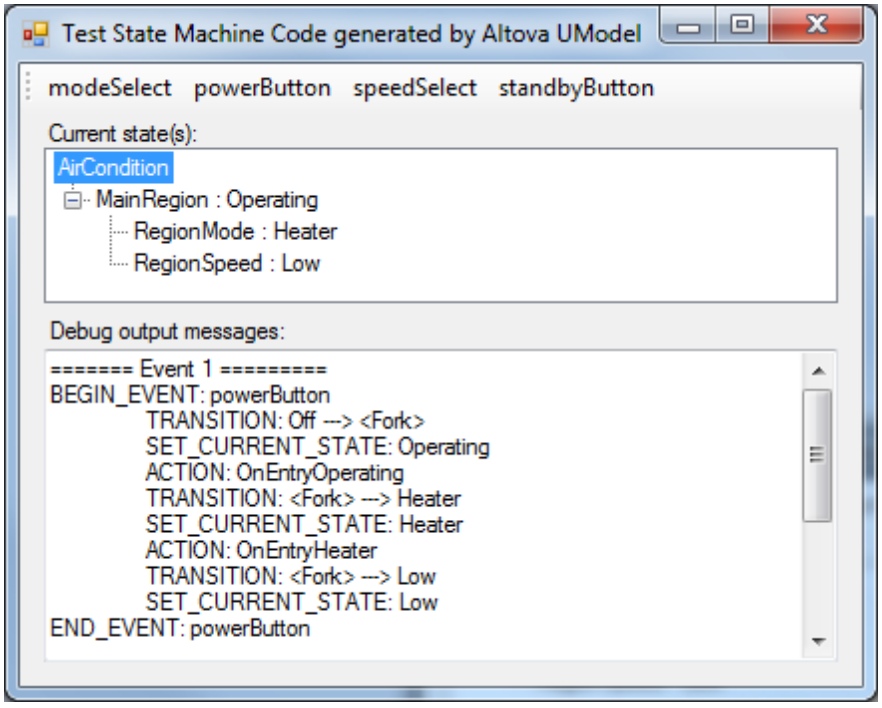
 © 2018 Altova GmbH



La imagen anterior muestra el diagrama de máquina de estados **`AirCondition`** de la carpeta `..\StateMachineCodeGeneration` del directorio `...\UModelExamples`. Por cada lenguaje de programación hay una carpeta.

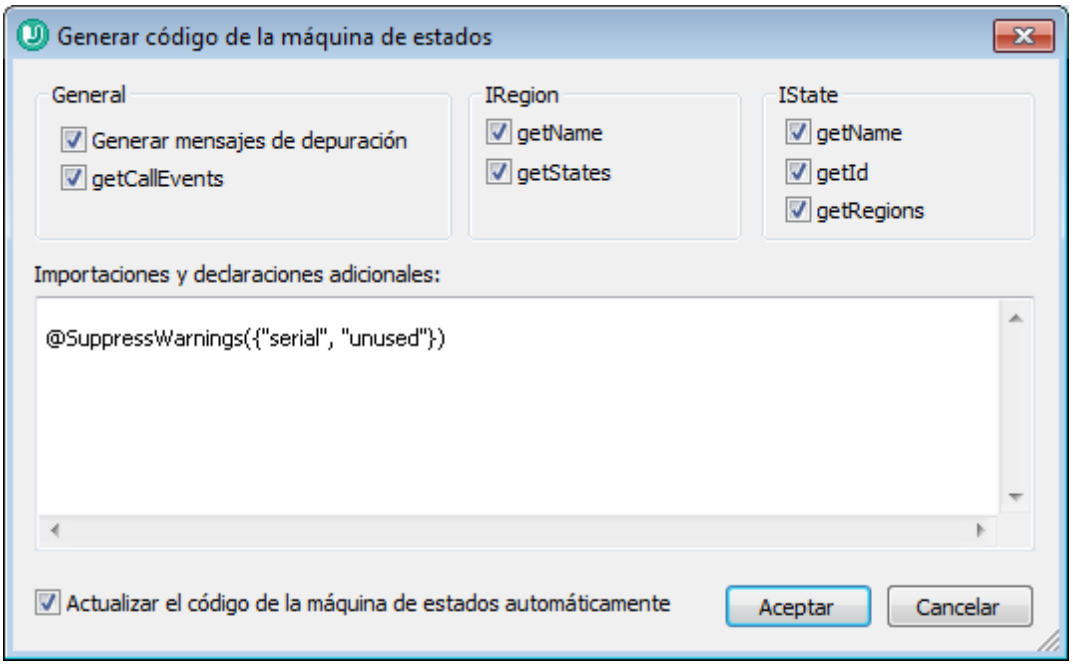
Cada directorio contiene dos carpetas: `AirCondition` y `Complex`. Cada una contiene el proyecto de UModel correspondiente, los archivos de proyecto del lenguaje de programación y los archivos de código generados. El archivo de proyecto `Complex.ump` contiene casi todos los elementos y funciones de modelado 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**.

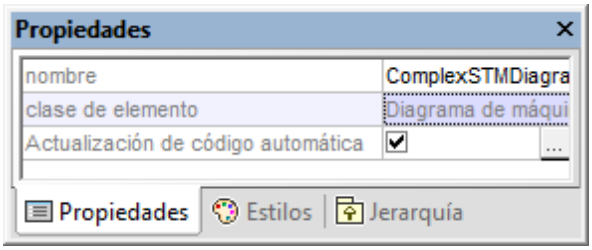


Aparece un cuadro de diálogo (*imagen siguiente*). Si es necesario, ajuste las opciones de configuración predeterminadas y haga clic en **Aceptar** para generar el código.

 © 2018 Altova GmbH

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** del 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

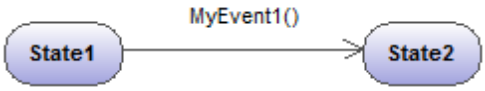
UModel admite estados compuestos y estados ortogonales, así que no hay un solo estado actual: cada región (de cualquier nivel jerárquico) puede tener un estado actual.

En el proyecto de ejemplo `AirCondition.ump` puede ver cómo se pueden recorrer las regiones hasta llegar a los estados actuales:

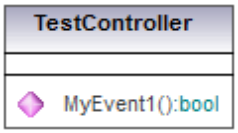
```
TreeNode rootNode = m_CurrentStateTree.Nodes.Add(m_STM.getRootState().getName());
UpdateCurrentStateTree(m_STM.getRootState(), rootNode);

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:

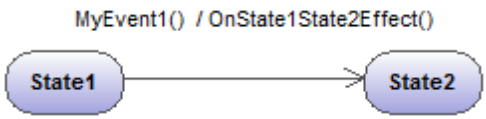
```
private class CTestStateMachine : IState
{
        …
        public bool MyEvent1()
```

 © 2018 Altova GmbH

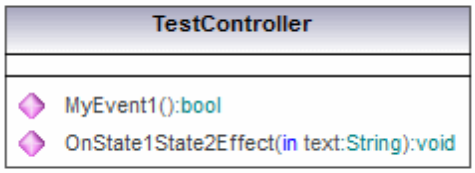
```
        {
                ...
        }
}
```

- 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



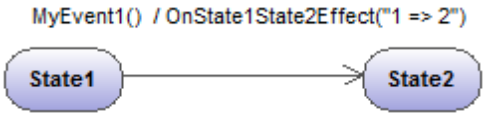
Método generado en el código:

```
private class CTestStateMachine : IState
{
        ...
        // Override to handle entry/exit/do actions, transition effects,...:
        public virtual void OnState1State2Effect() {}
}
```

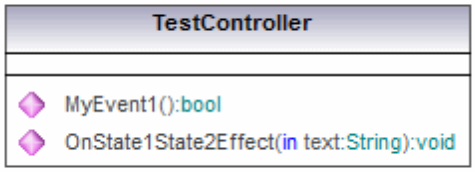
- La implementación de la máquina de estados llamará a "OnState1State2Effect()" cuando se dispare la transición del estado "State1" al estado "State2".
- Para reaccionar a este efecto "OnState1State2Effect()" debería sobrescribirse en una clase derivada de "CTestStateMachine".
- "CTestStateMachine:: 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:

```
// Overwrite 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.

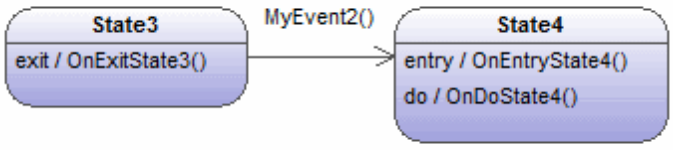


Método generado en el código:

```
private class CTestStateMachine : IState
{
        …
        // Additional defined operations of the controller class:
        public virtual void OnState1State2Effect(String text)
        {
        }
}
```

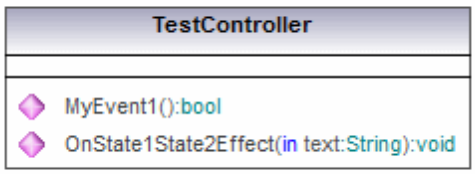
- Para llevar a cabo las operaciones (creadas automáticamente por UModel), puede añadir parámetros manualmente (UModel no puede conocer el tipo necesario).
- En este ejemplo el parámetro "text:String" se añadió al método Effect de TestController. Es necesario especificar un argumento adecuado cuando 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.

© 2018 Altova GmbH



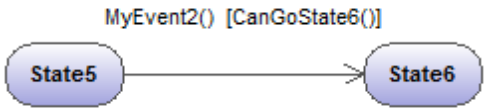
Método generado en el código:

```
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() {}
}
```

- 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 verdaderamente.

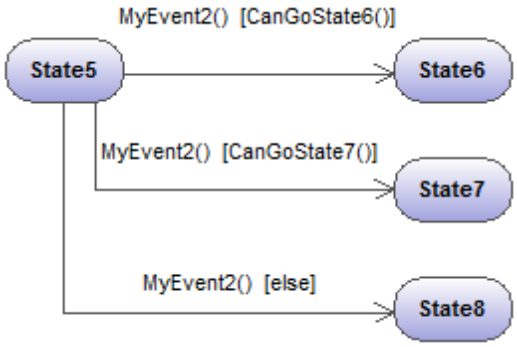


La operación correspondiente se genera automáticamente en UModel.

Método generado en el código:

```
private class CTestStateMachine : IState
{
        …
        // Additional defined operations of the controller class:
        public virtual bool CanGoState6()
        {
            return true; // Override!
        }
}
```

- 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 secuencia de los procesos.

 © 2018 Altova GmbH

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 todas las actividades deben completarse.

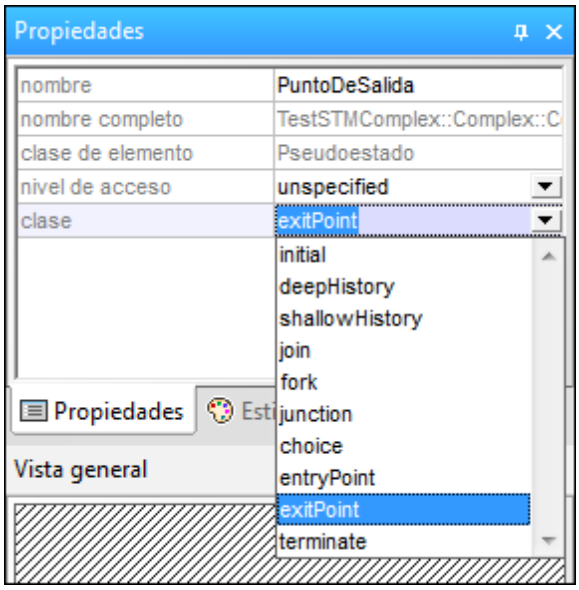
Reunión horizontal (pseudoestado)
Reúne/combina subsecuencias definidas previamente. Para poder continuar todas las actividades deben completarse.

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 en la referencia de máquina de estados al que hace referencia el estado de submáquina.

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`**

 © 2018 Altova GmbH

- **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.

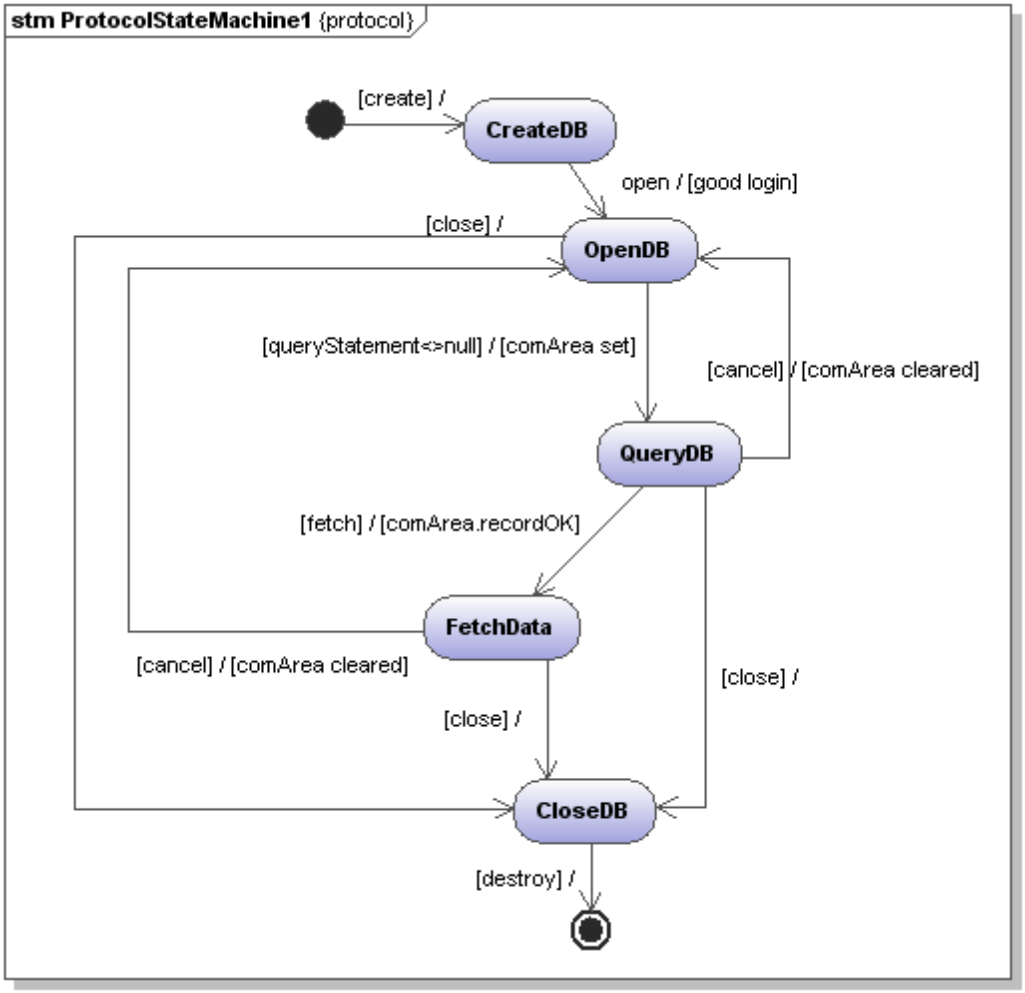
9.1.3 Diagrama de máquina de estados de protocolos

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.

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 máquina de estados de protocolos.

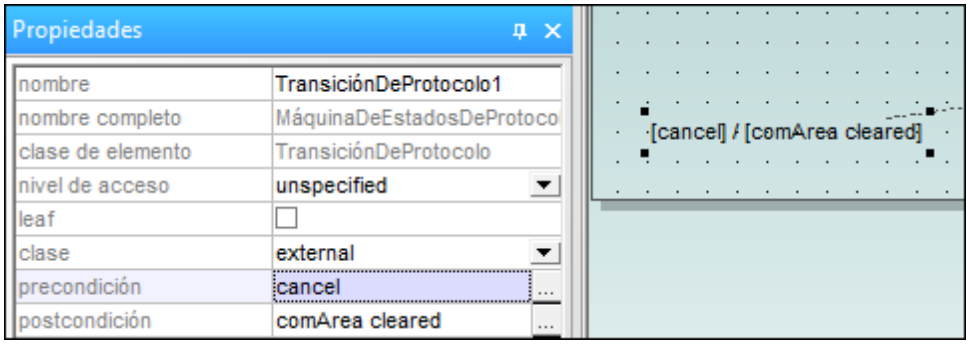
 © 2018 Altova GmbH

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:

Véase 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.

 © 2018 Altova GmbH

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:

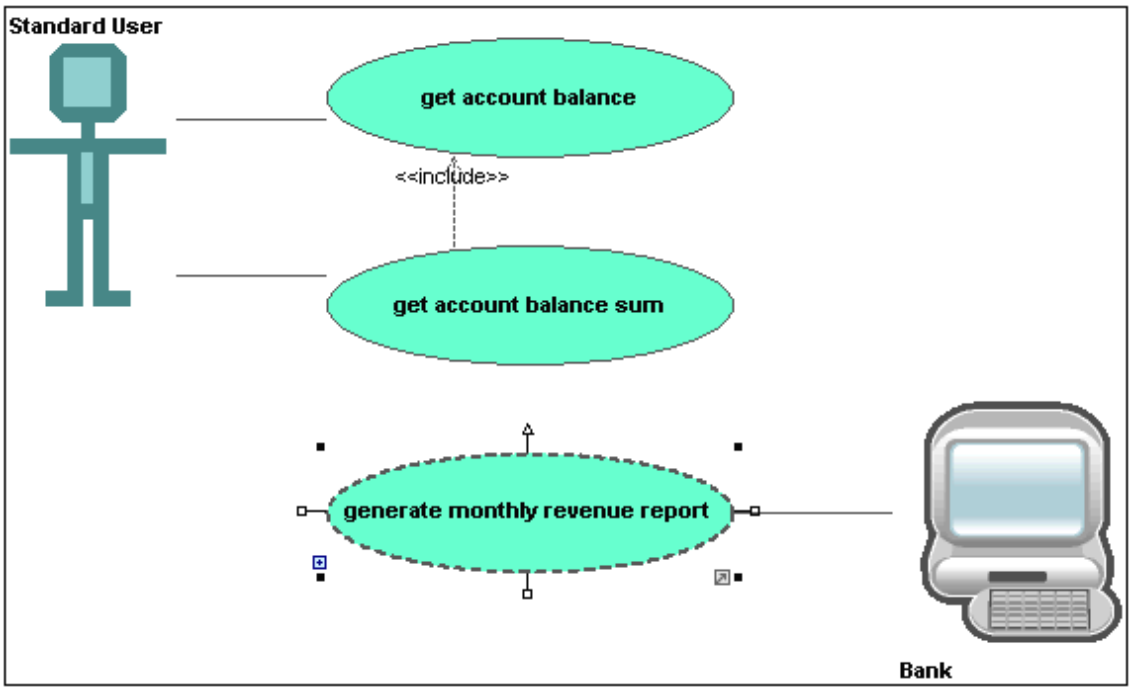
- El estado al que está conectado el punto debe hacer referencia a una máquina de estado de submáquina (visible en la ventana *Propiedades*).
- Esta submáquina debe contener un punto de entrada y otro de salida como mínimo.

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

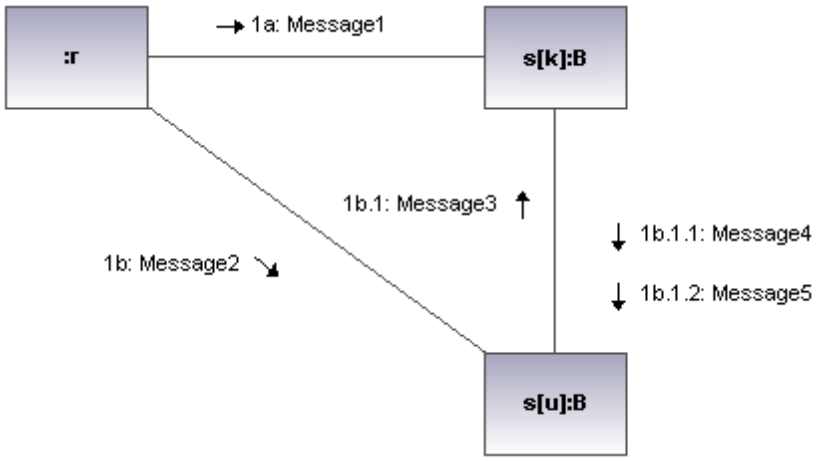
Sitio web de Altova: Diagramas de comunicación UML

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.

© 2018 Altova GmbH



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.

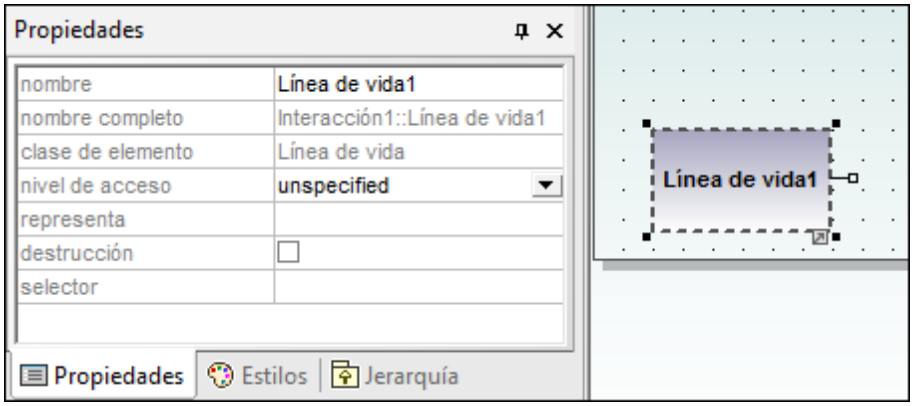
Línea de vida

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`.

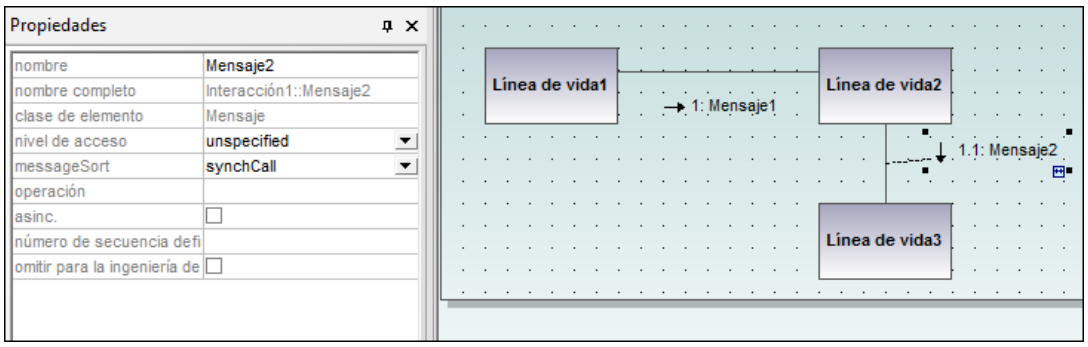
Mensajes

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).

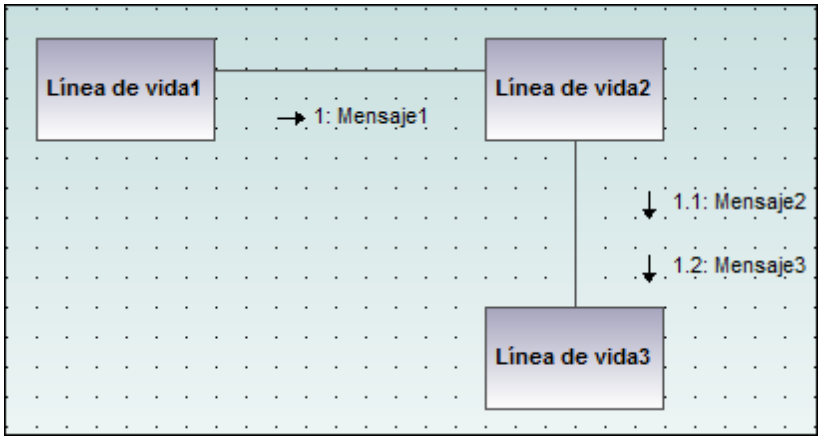


Nota: mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo para insertar varios mensajes.

Para insertar más mensajes:

1. Haga clic con el botón derecho en una línea de comunicación del diagrama y seleccione **Nuevo/a | Mensaje**.

 © 2018 Altova GmbH



Nota: la dirección en la que se arrastra la flecha define la dirección del mensaje. Los mensajes de respuesta pueden apuntar en ambas direcciones.

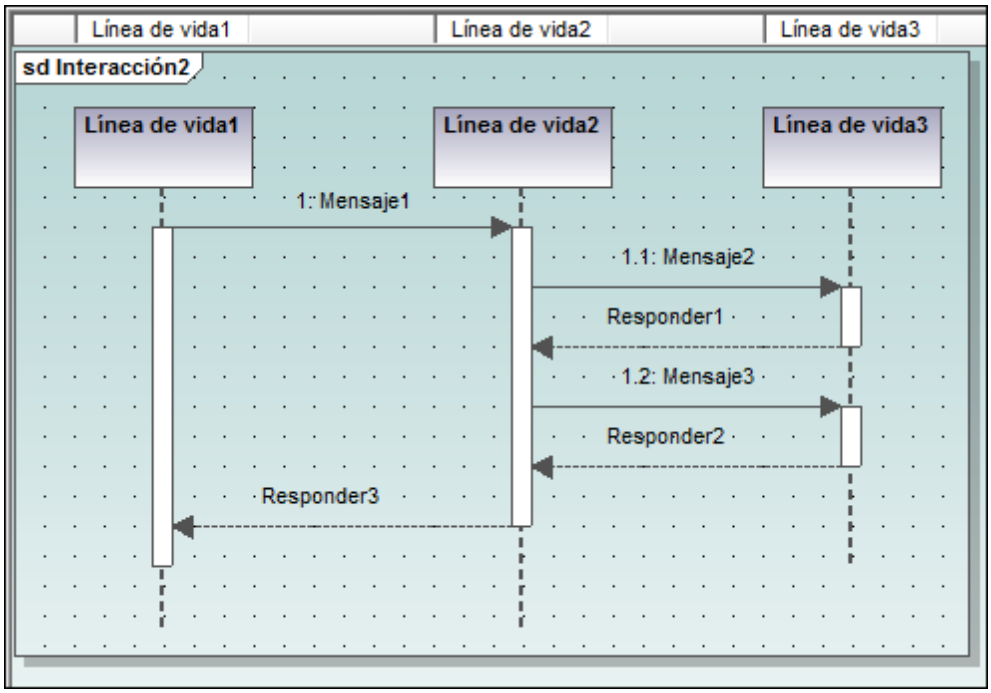
Numeración de los mensajes

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:

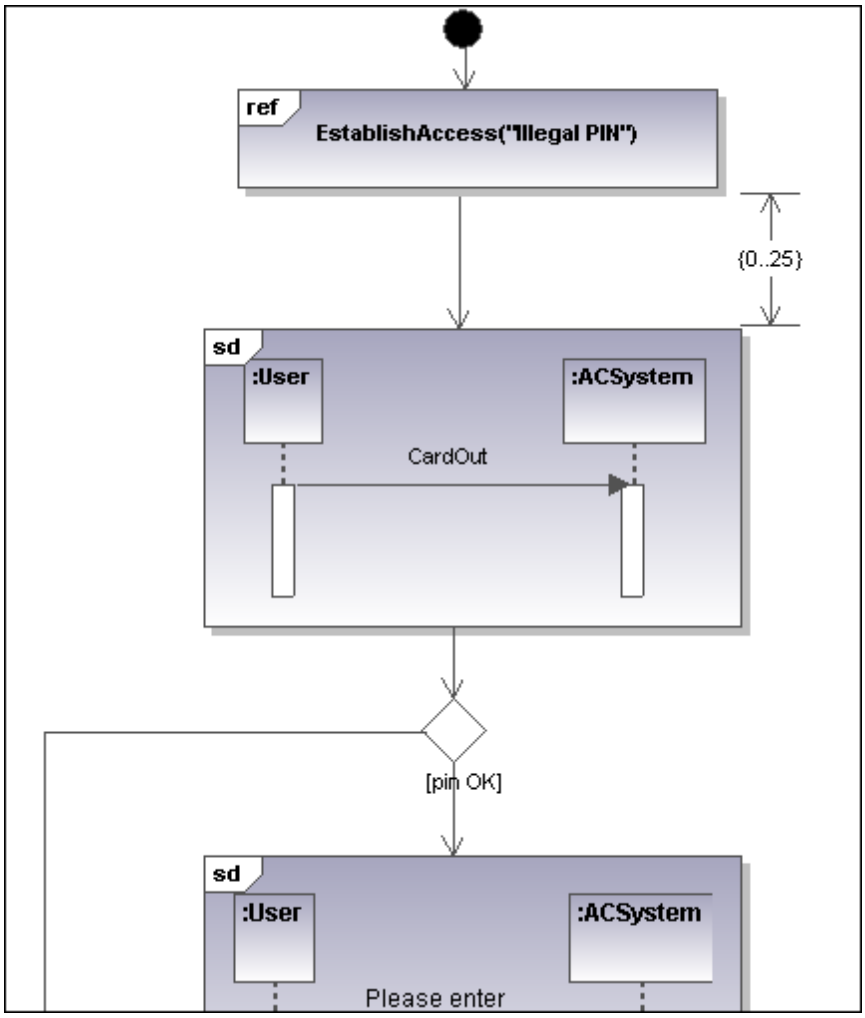
1. Haga clic con el botón derecho en el área de trabajo del diagrama de comunicación.
2. Seleccione **Generar diagrama de secuencia** en el menú contextual.



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.



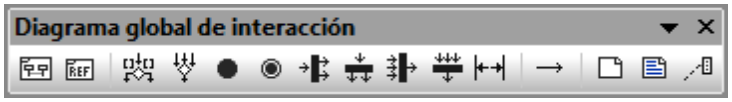
En lugar de actividades, este diagrama utiliza dos tipos distintos de interacciones: **Interacción** y **UsoDeInteracción**.

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.

 © 2018 Altova GmbH

9.1.6.1 *Insertar elementos*



Para insertar elementos con los iconos de la barra de herramientas:

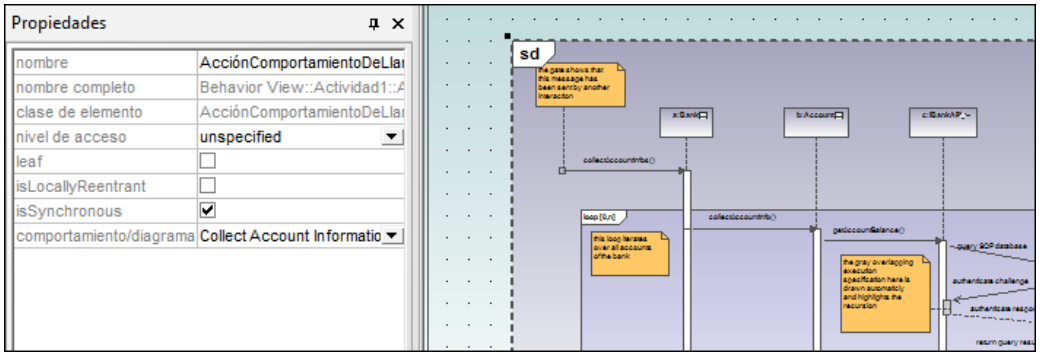
1. Haga clic en un icono de la barra de herramientas Diagrama global de interacción.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

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.

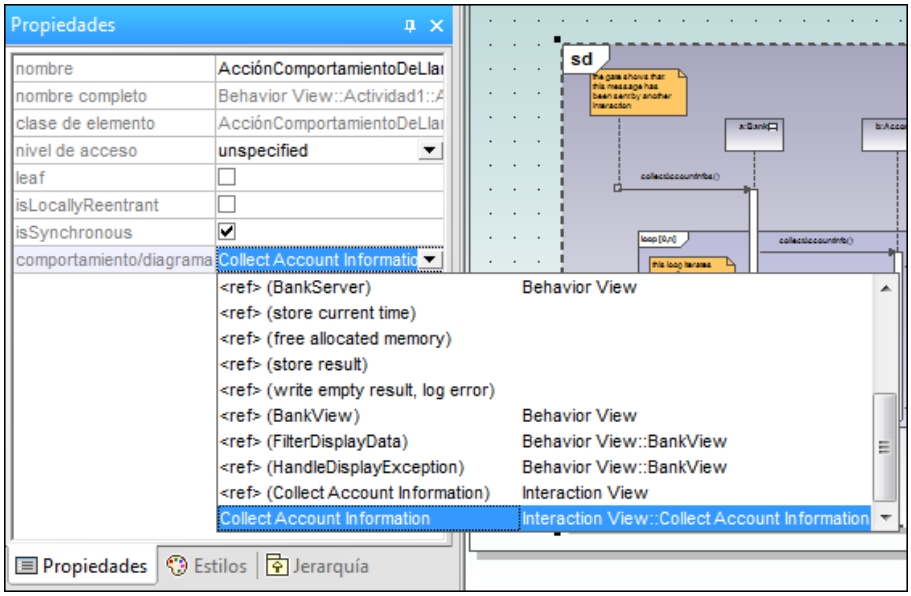
Para insertar un elemento Interacción:

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.

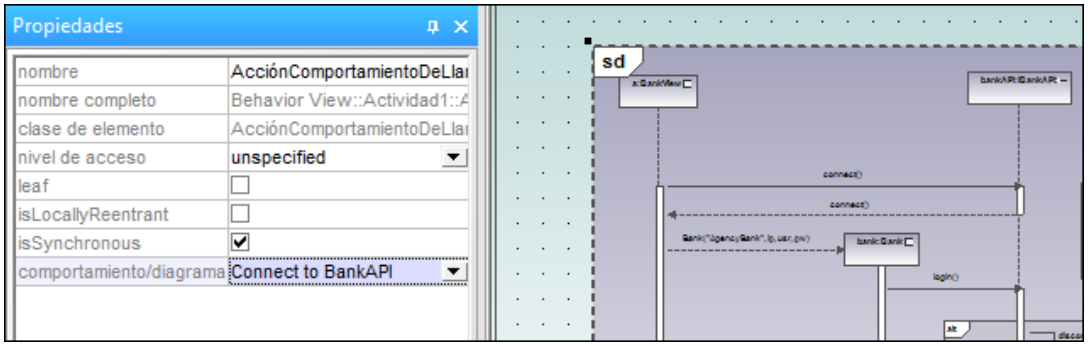


Si usa el archivo de ejemplo `Bank_MultiLanguage.ump` de la carpeta `...\UModelExamples`, el diagrama de secuencia **`Collect Account Information`** se inserta automáticamente. El primer diagrama de secuencia de la *Estructura del modelo* se selecciona por defecto.

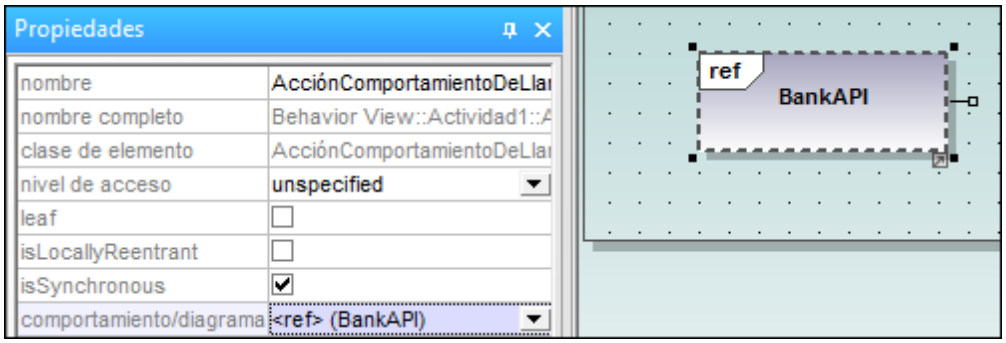
2. Para cambiar el elemento **Interacción** predeterminado, haga clic en el cuadro combinado `comportamiento/diagrama` de la ventana *Propiedades*. La lista desplegable incluye todos los elementos que se pueden insertar.



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**.

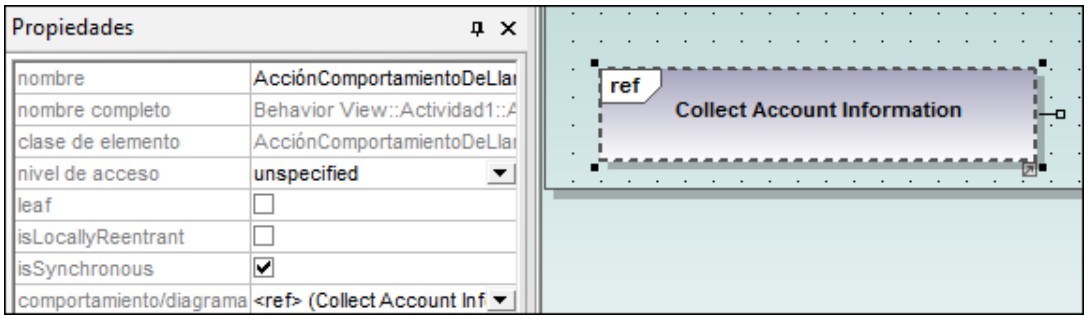


Para insertar una instancia del elemento Interacción:

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

 © 2018 Altova GmbH

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.
   Recuerde que 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 con 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.

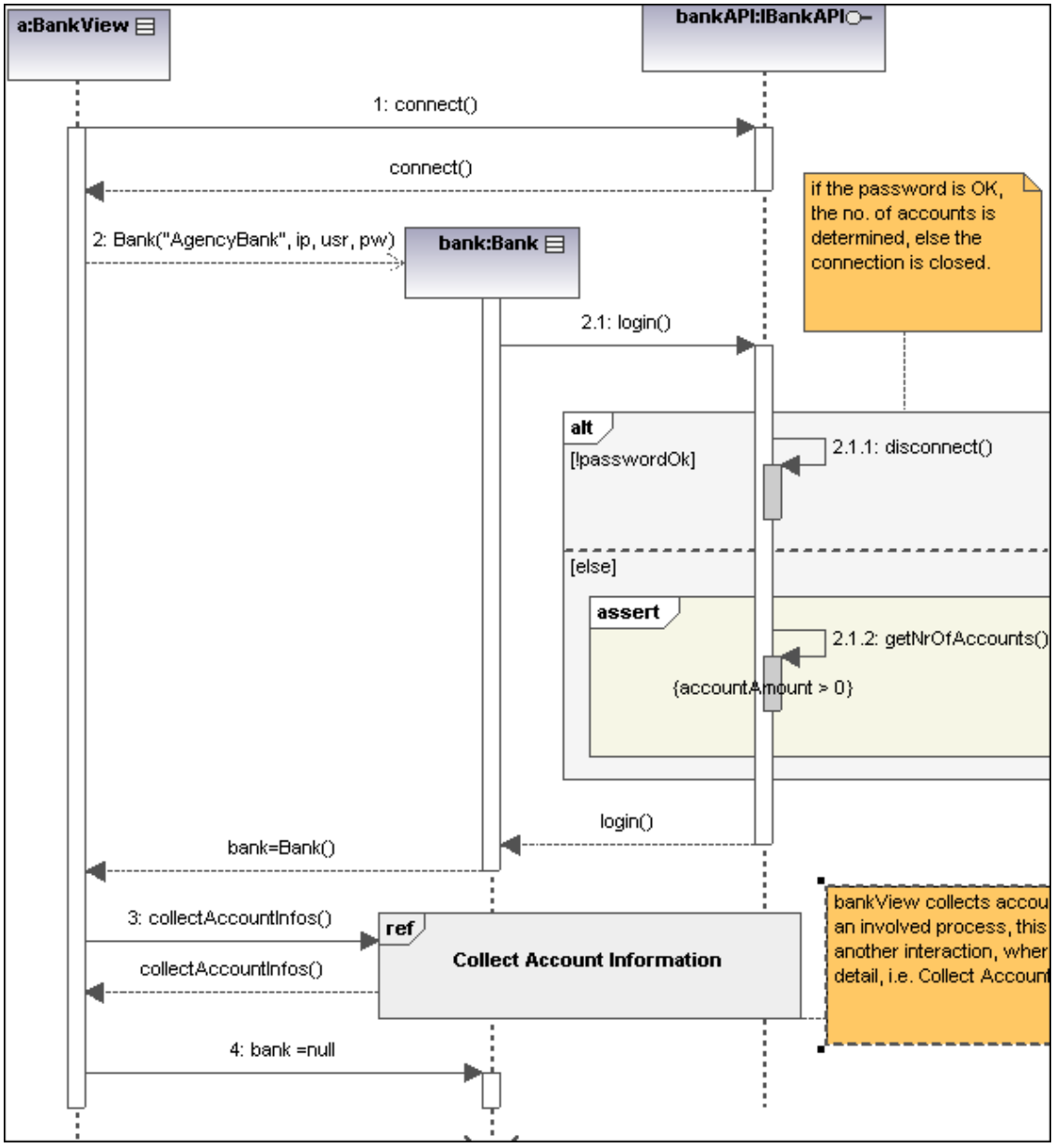
9.1.7 Diagrama de secuencia

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`.

Nota: también puede generar diagramas de secuencia a partir de código fuente. Para más información consulte el apartado Generar diagramas de secuencia .

 © 2018 Altova GmbH



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.

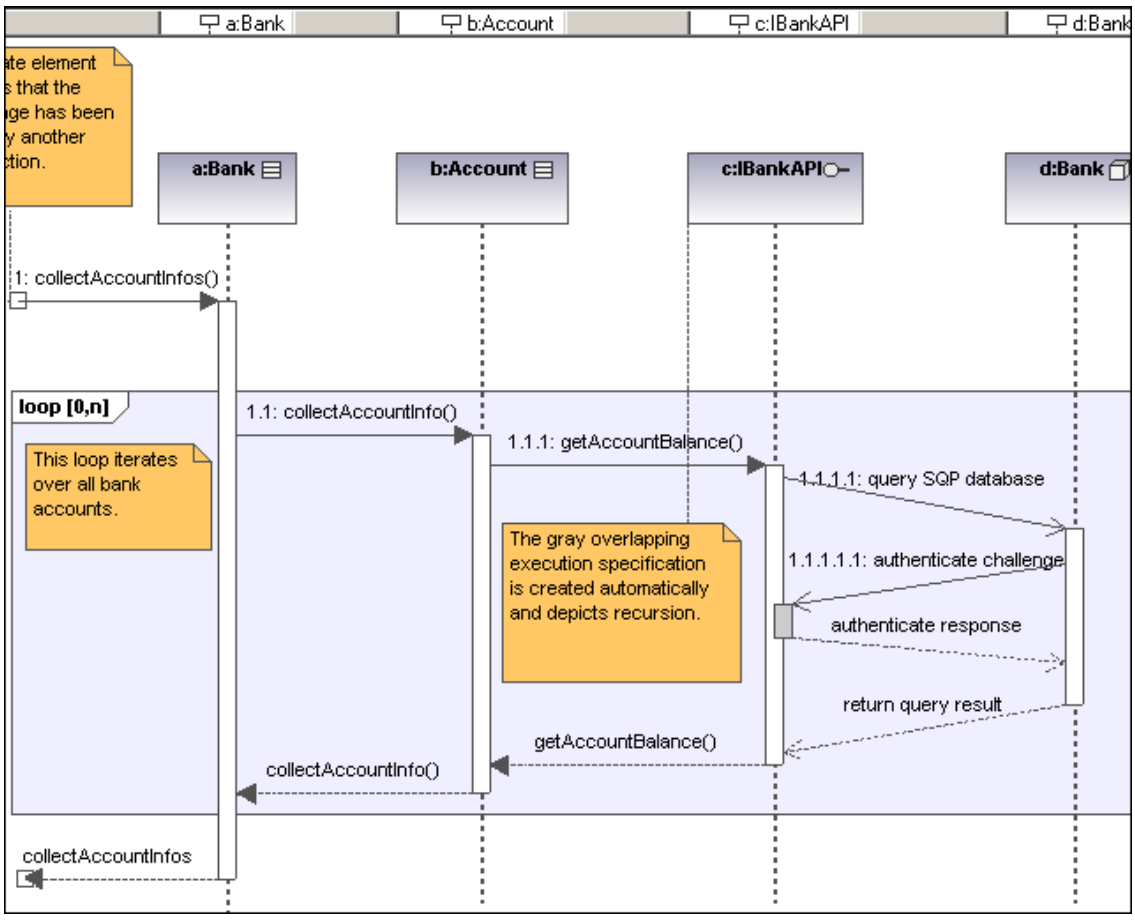
- Los **mensajes** se envían de un objeto a otro, se dibujan en forma de flecha y tienen una etiqueta de texto. Pueden tener un número secuencial y otros atributos opcionales como listas de argumentos, etc. Los mensajes pueden ser condicionales, opcionales y

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.



Para 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.

© 2018 Altova GmbH

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 diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de secuencia.

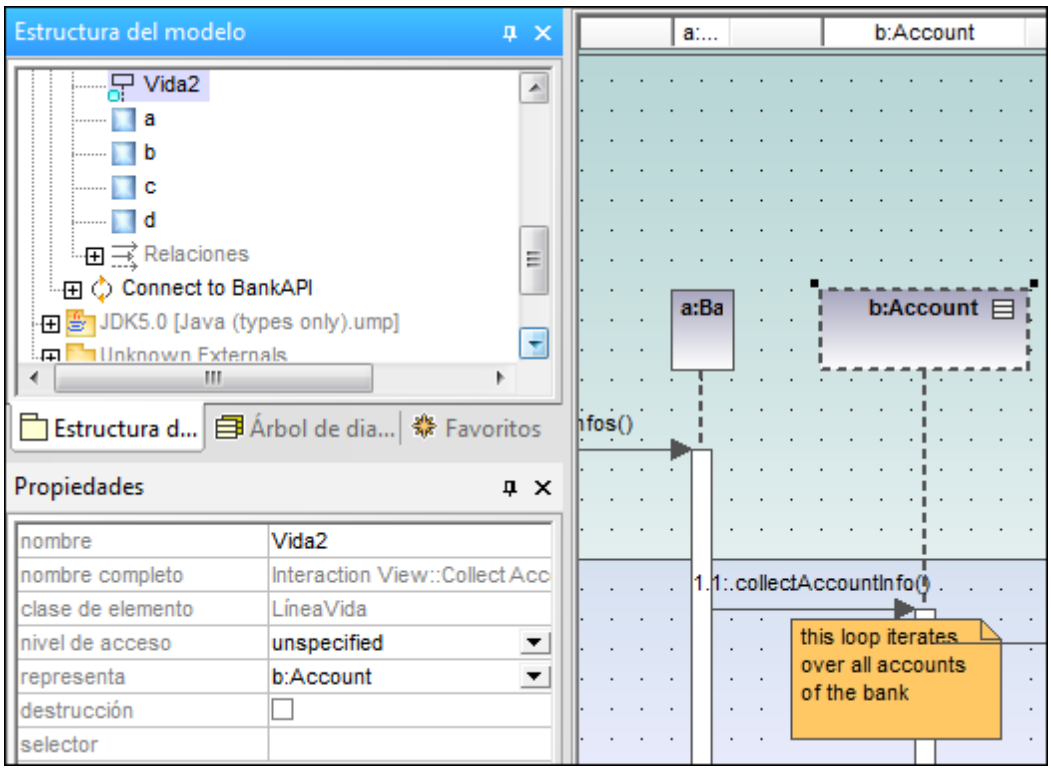
9.1.7.1.1 Líneas de vida

Línea de vida

Una 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:

1. 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).

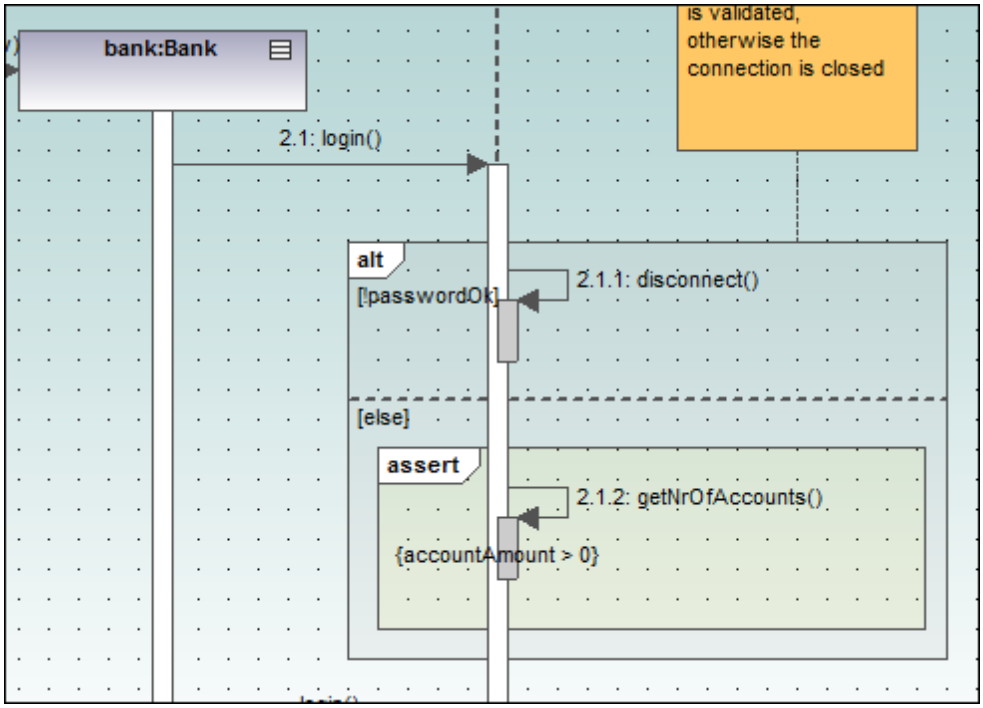
Ir a una línea de vida

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*.

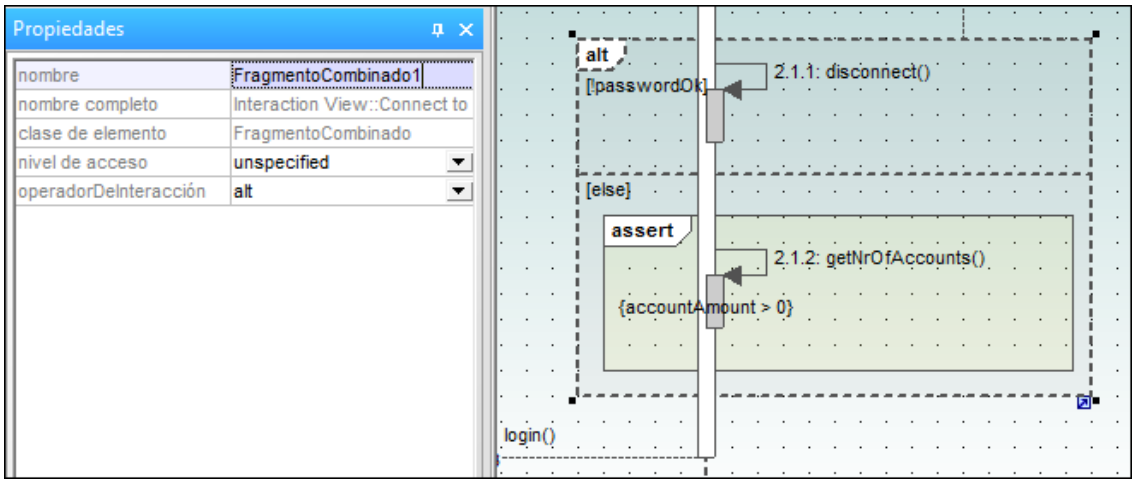
9.1.7.1.2 Fragmentos combinados

FragmentoCombinado

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.



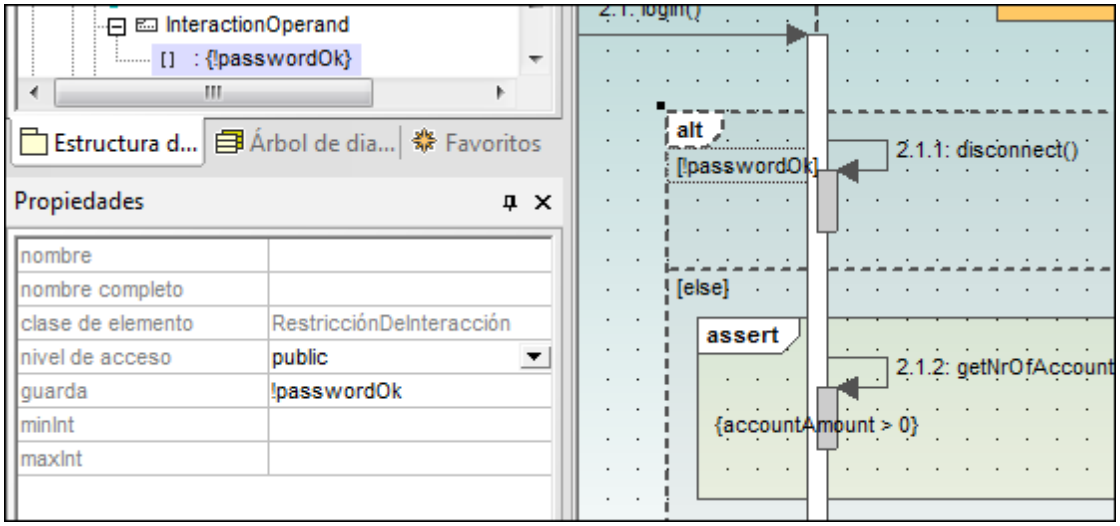
OperadoresDeInteracción

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 todos los 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.

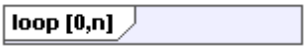
Opción **opt**
Representa una opción entre ejecutar el operando o no hacer nada.

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**

 © 2018 Altova GmbH

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.

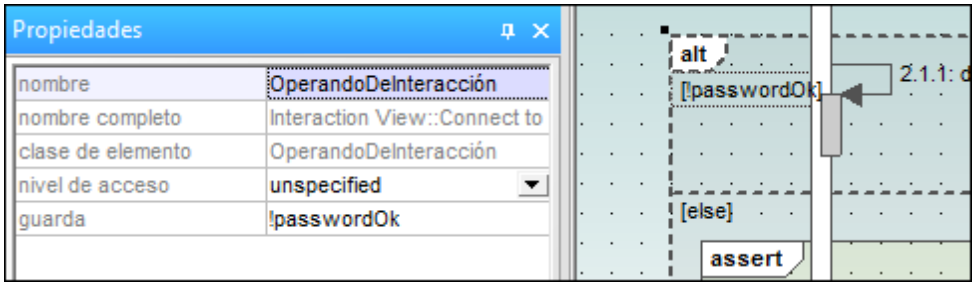
Aserción **assert**
Designa el fragmento combinado válido y sus secuencias. Se suele usar junto con los operandos **consider** o **ignore**.

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.

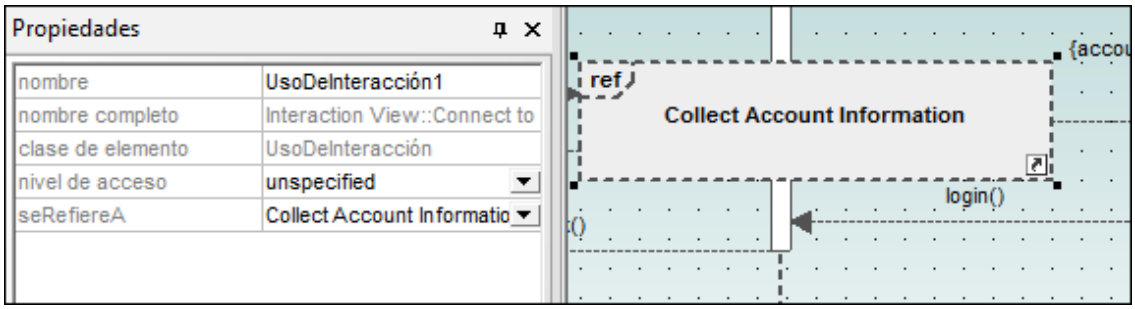
Para eliminar operandos de interacción:

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

UsoDeInteracción

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.

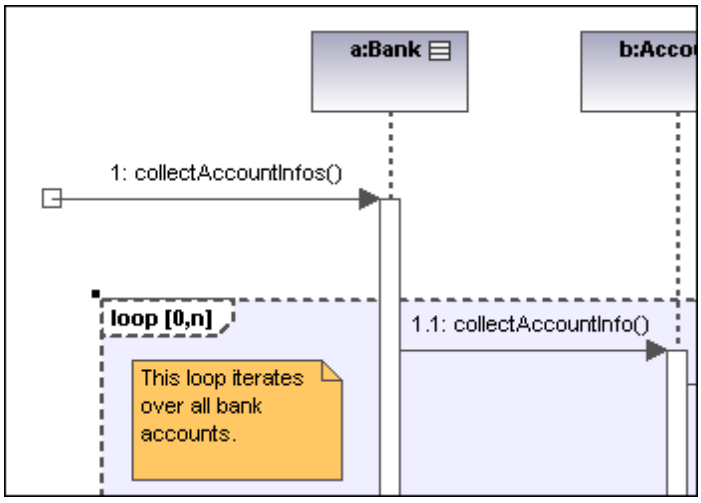
Nota: también puede arrastrar un **`UsoDeInteracción`** desde la *Estructura del modelo* hasta el área de trabajo del diagrama.

9.1.7.1.4 Puertas

Puerta

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.



© 2018 Altova GmbH

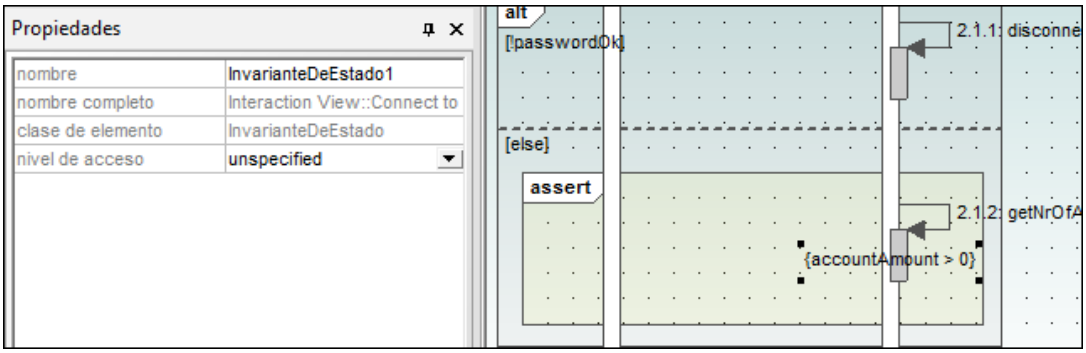
9.1.7.1.5 Invariantes de estado

InvarianteDeEstado

Una invariante de estado 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.

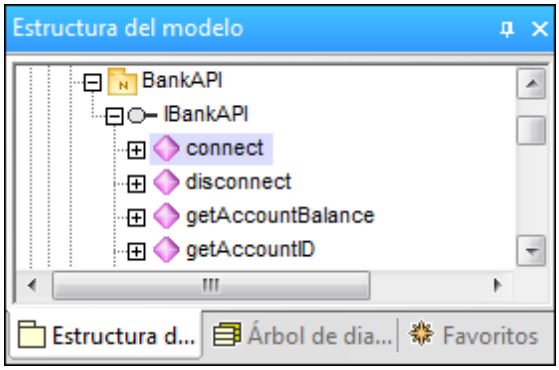
Para eliminar un mensaje:

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.

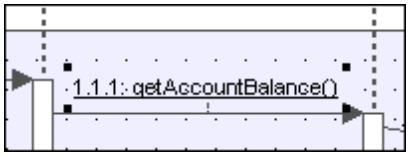
"Ir a la operación" para mensajes de llamada

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.

 © 2018 Altova GmbH

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.

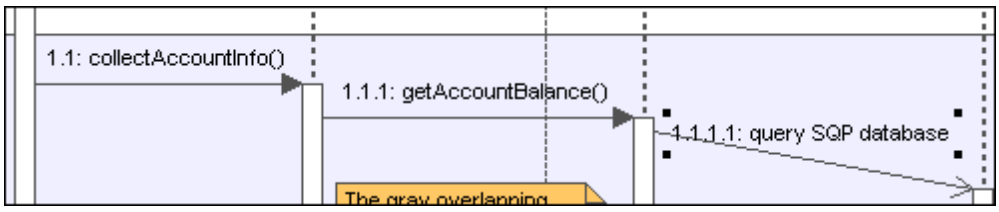
Para crear mensajes de respuesta automáticamente:

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:

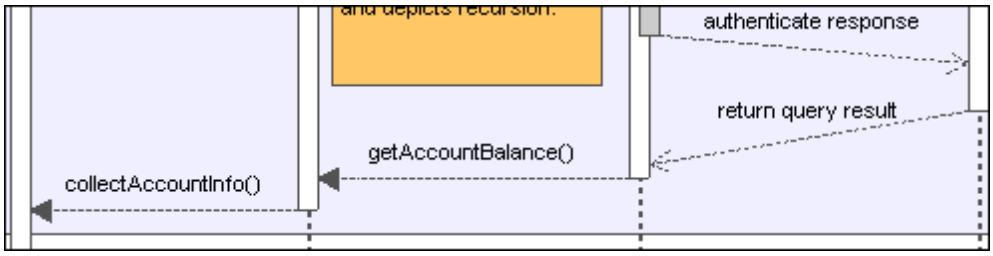
Hay dos formas de seleccionar el tipo de numeración:
- Con el icono correspondiente de la barra de herramientas Diagrama de secuencia.
- En la ventana *Estilos*.

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.

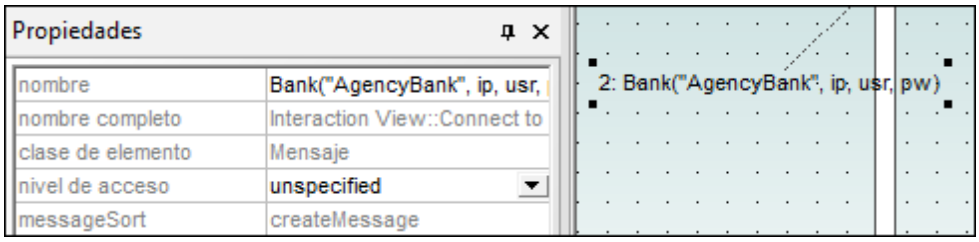


Por lo general, los mensajes de respuesta vienen señalados por la parte inferior del cuadro de activación. Si los cuadros de activación están deshabilitados (panel *Estilos*, `Mostrar especificaciones de ejecución=false`), entonces se recomienda usar flechas de respuesta para evitar ambigüedades.

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.

Para 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.

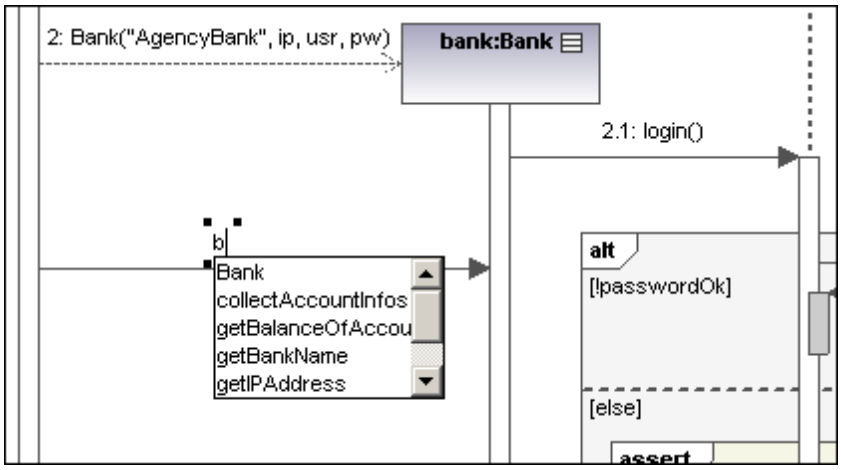


Para enviar mensajes a métodos/operaciones de clases de diagramas de secuencia:

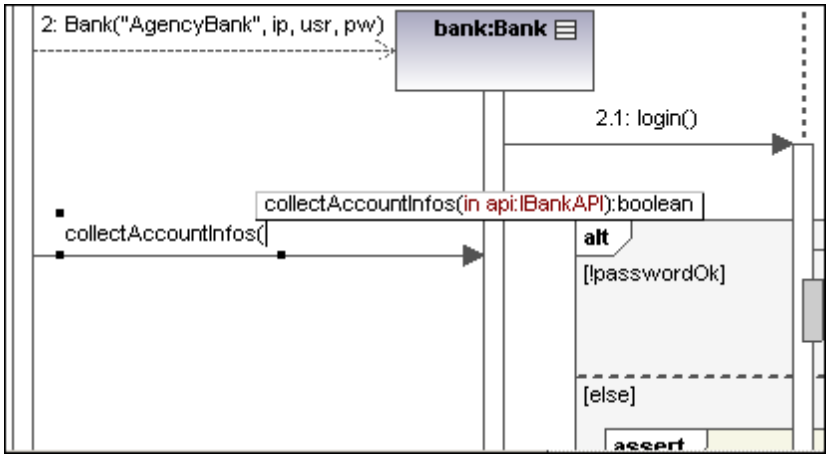
Tras insertar una clase de la *Estructura del modelo* en un diagrama de secuencia, puede crear un mensaje entre una línea de vida y un método de la clase destinataria (línea de vida). Para ello puede usar la ayuda sintáctica y de las funciones de finalización automática de UModel.

 © 2018 Altova GmbH

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 ya existentes.



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

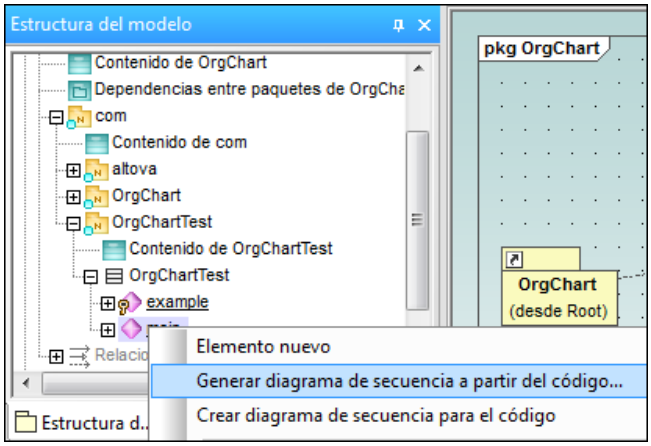
9.1.7.2 Generar diagramas de secuencia a partir de código fuente

UModel puede modelar diagramas de secuencia a partir de código.

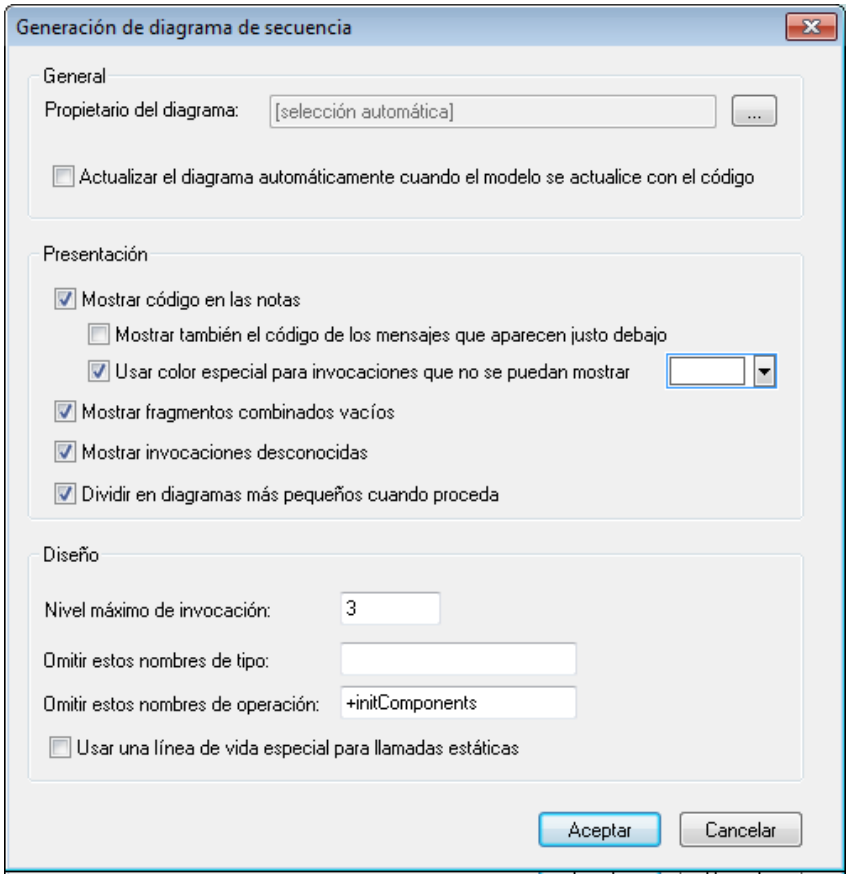
Con el siguiente ejemplo puede aprender a crear un diagrama de secuencia a partir de un método automáticamente. Si quiere, puede hacer lo mismo con su propio código. El método está en el paquete `orgChart`, que se importó al modelo con el comando **Proyecto | Importar directorio de código fuente** (para más información consulte el apartado Ingeniería de ida y vuelta (código - modelo - código).

1. 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.

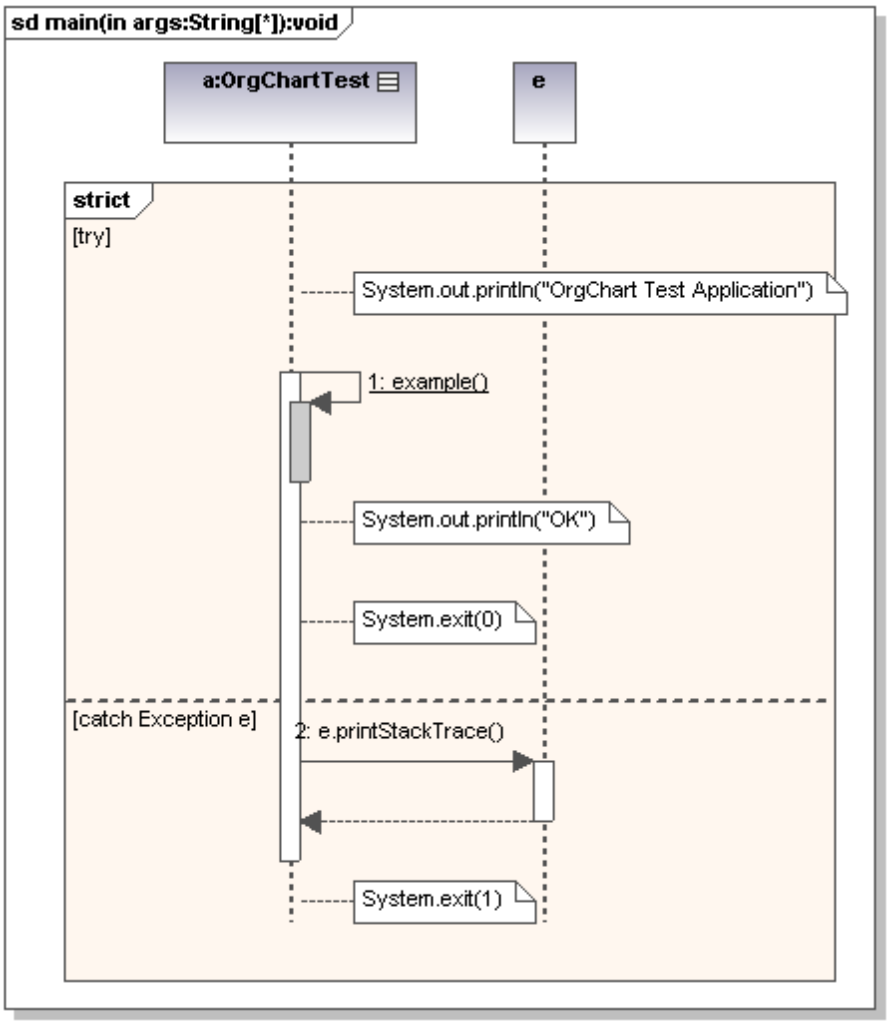
 © 2018 Altova GmbH



Esto abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede configurar la generación.



2. 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.



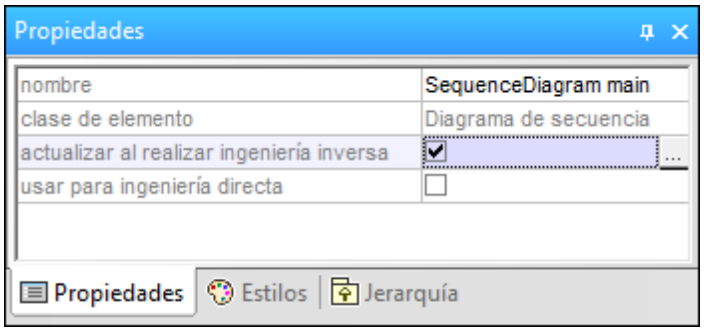
Notas:

- Puede asignar un color distinto a las invocaciones que no se puedan mostrar.
- El nivel máximo de invocación define el nivel de recursión que se debe usar en el diagrama.
- La opción *Omitir estos nombres de tipo:* sirve para crear una lista delimitada por comas de los tipos que no deben aparecer en el diagrama de secuencia generado.
- La opción *Omitir estos nombres de operación:* sirve para crear una lista delimitada por comas de las operaciones que no deben aparecer en el diagrama de secuencia generado (**`initComponents`** se añade automáticamente a la lista).

  Al añadir nombres de operación a la lista (p. ej. **`InitComponents`**), se omite la operación completa. Si añade el carácter + delante del nombre de la operación (p. ej. **`+InitComponent`**), las llamadas de la operación aparecen en el diagrama, pero sin su contenido.
- La opción *Dividir en diagramas más pequeños cuando proceda* divide automáticamente los diagramas de secuencia en varios subdiagramas y genera hipervínculos entre ellos para poder navegar por ellos con facilidad.

 © 2018 Altova GmbH

Los diagramas de secuencia se actualizan automáticamente cuando se actualiza el proyecto de UModel entero. Si, por el contrario, solo se actualizan las clases o algunos archivos desde una aplicación externa, los diagramas no se actualizan. No obstante, esto puede configurarse marcando la casilla del campo `Actualización de código automática`.



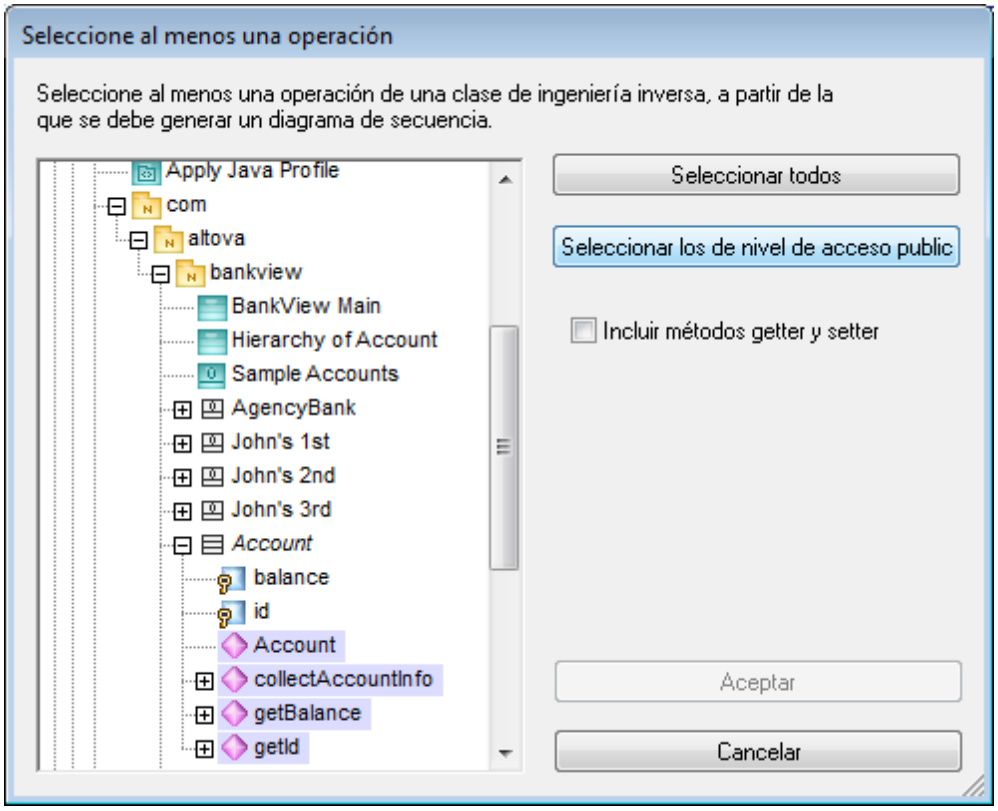
Haga clic en el icono ... del campo `Actualización de código automática` para abrir el cuadro de diálogo "Generación de diagrama de secuencia" y cambiar las opciones de generación.

9.1.7.3 Generar varios diagramas de secuencia a partir de propiedades

Además de crear diagramas de secuencia a partir de operaciones, UModel puede crear diagramas de secuencia a partir de propiedades Getter/Setter.

Crear varios diagramas de secuencia a partir de varias 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".



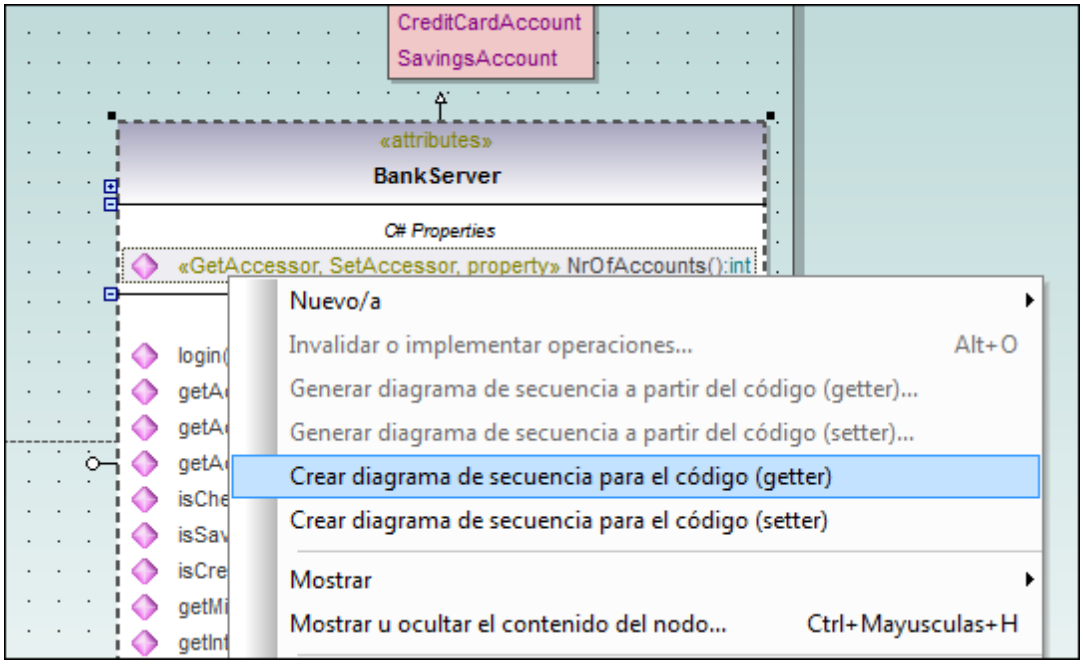
2. Seleccione las operaciones para las que desea generar un diagrama de secuencia y haga clic en **Aceptar** (si quiere puede usar los botones **Seleccionar todos** y **Seleccionar los de nivel de acceso public**). Al hacer clic en **Aceptar** se abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede elegir las opciones de generación.
3. Haga clic en **Aceptar** para generar los diagramas de secuencia. Por cada operación seleccionada se genera un diagrama de secuencia distinto.

   Nota: cuando genere diagramas de secuencia también puede elegir si se incluyen o excluyen los getter y setter.

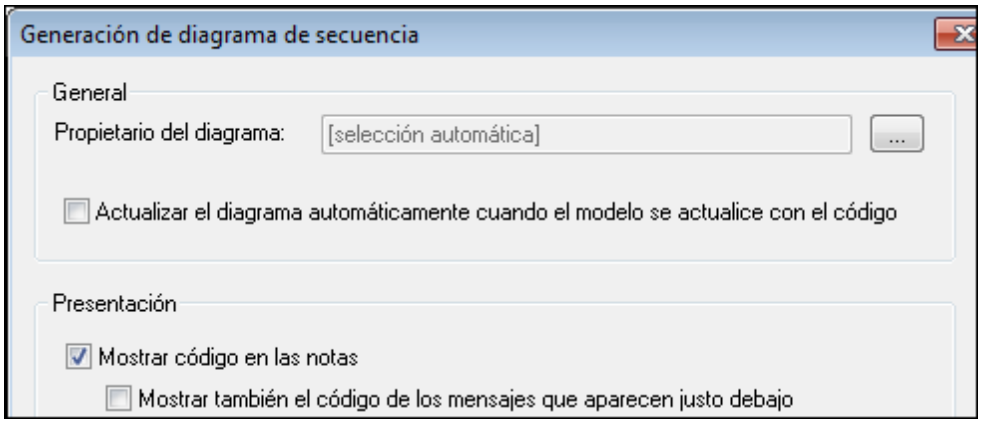
Crear un diagrama de secuencia a partir de propiedades getter/setter (C#, VB .NET)

1. Haga clic con el botón derecho en una operación que tenga el estereotipo `GetAccessor/SetAccessor`.

 © 2018 Altova GmbH



2. Seleccione la opción del menú contextual correspondiente (p. ej. **Crear diagrama de secuencia para el código (getter)**).
   Esto abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede configurar la presentación del diagrama de secuencia que se generará.



3. Haga clic en **Aceptar** para generar el diagrama de secuencia.

9.1.7.4 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**).

Nota: cuando use como base un diagrama de secuencia creado por ingeniería inversa, compruebe que la opción *Mostrar código en las notas* esté activa durante el proceso de ingeniería inversa. Así no perderá código cuando vuelva a iniciar el proceso de ingeniería directa.

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:

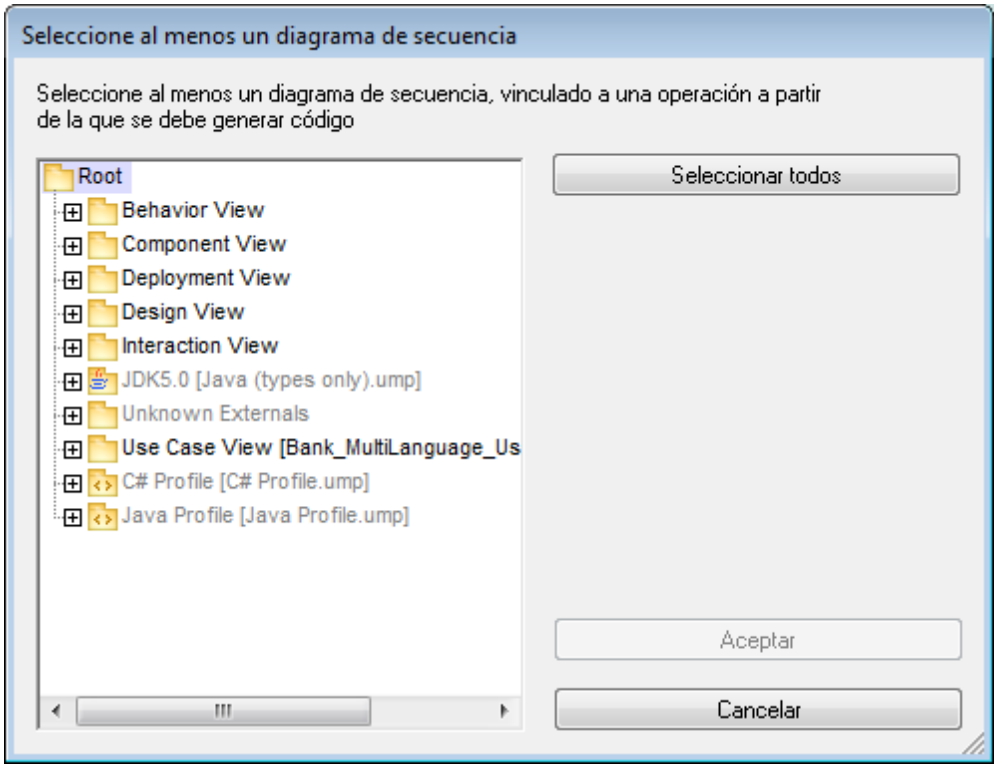
- Active la casilla `Usar para ingeniería directa` de la ventana *Propiedades*.

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.

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.

© 2018 Altova GmbH



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.

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.

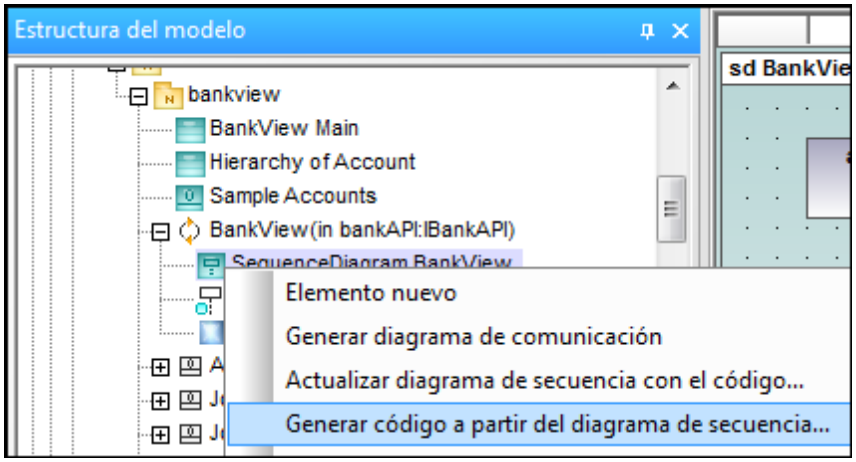
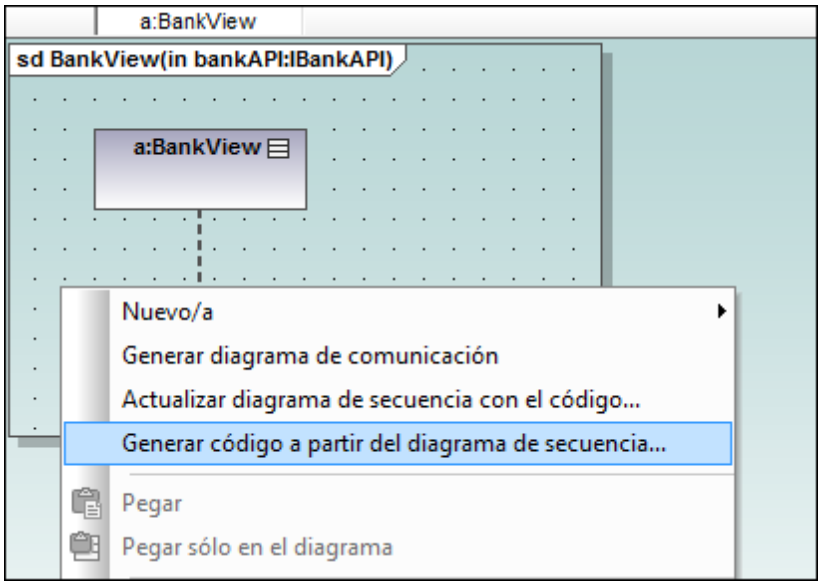


Diagrama de secuencia con código de una operación

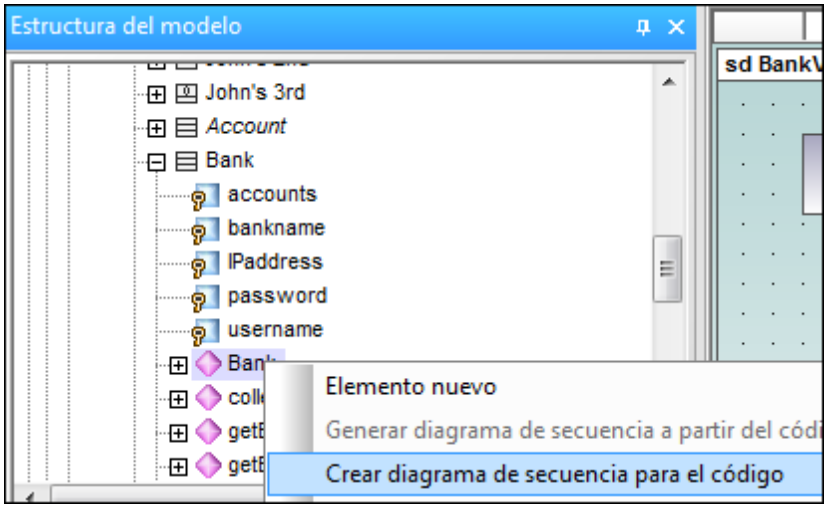
1. Haga clic con el botón derecho en el fondo del diagrama de secuencia que contiene el código de una operación.
2. Elija la opción **Generar código a partir del diagrama de secuencia**.



Este comando inicia el proceso de ingeniería directa.

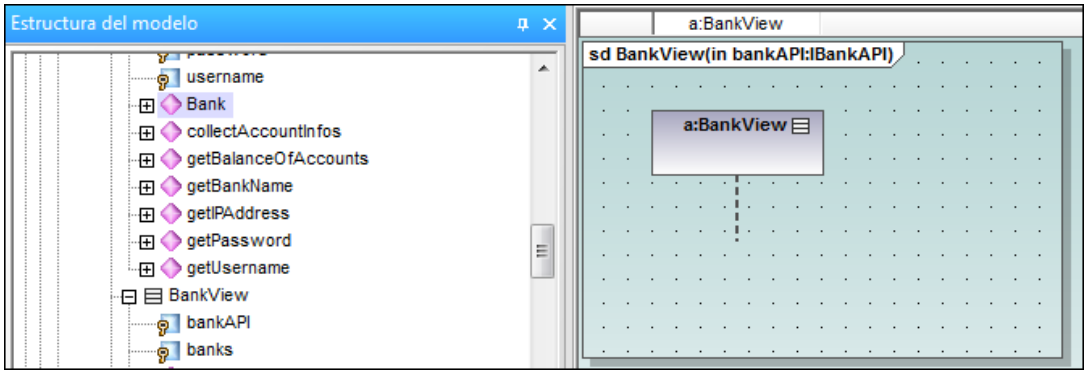
Para crear un diagrama de secuencia para el código (ingeniería):

1. 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.

 © 2018 Altova GmbH



El resultado es un diagrama de secuencia nuevo que contiene la línea de vida de esa clase.

9.1.7.4.1 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.

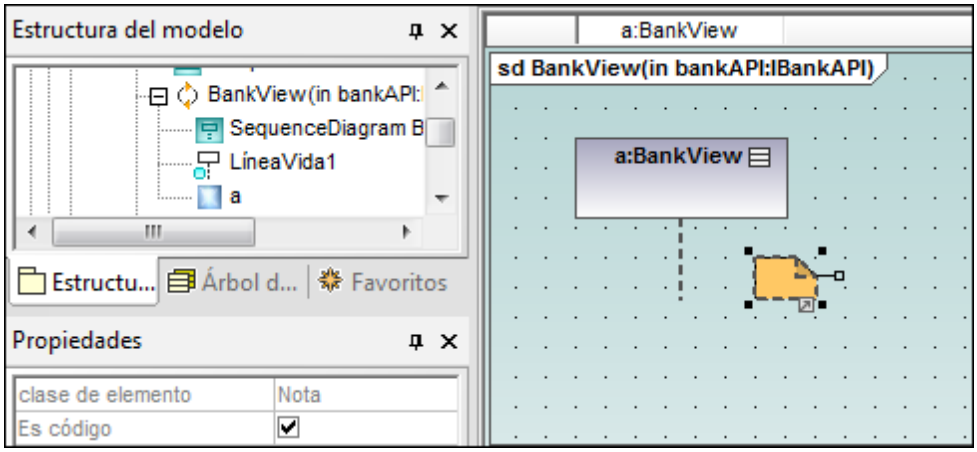
Cuando se aplica ingeniería inversa a código, los elementos estándar de los diagramas de secuencia (p. ej. los `FragmentosCombinados`) se asignan a los elementos del código (p. ej. instrucciones `if`, bucles, etc.).

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:

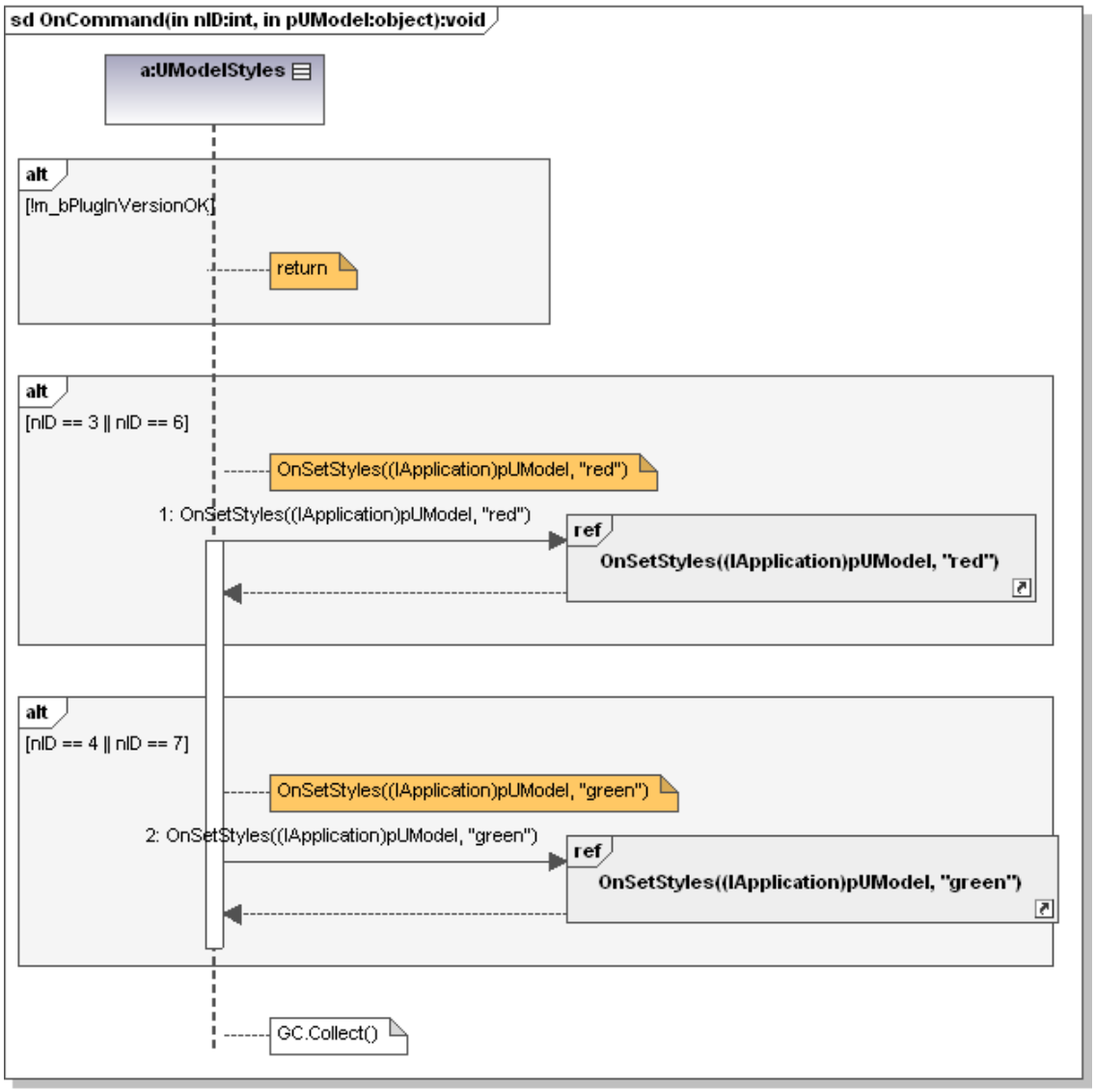
Notas para generación de código:

(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.

 © 2018 Altova GmbH



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();
}
```

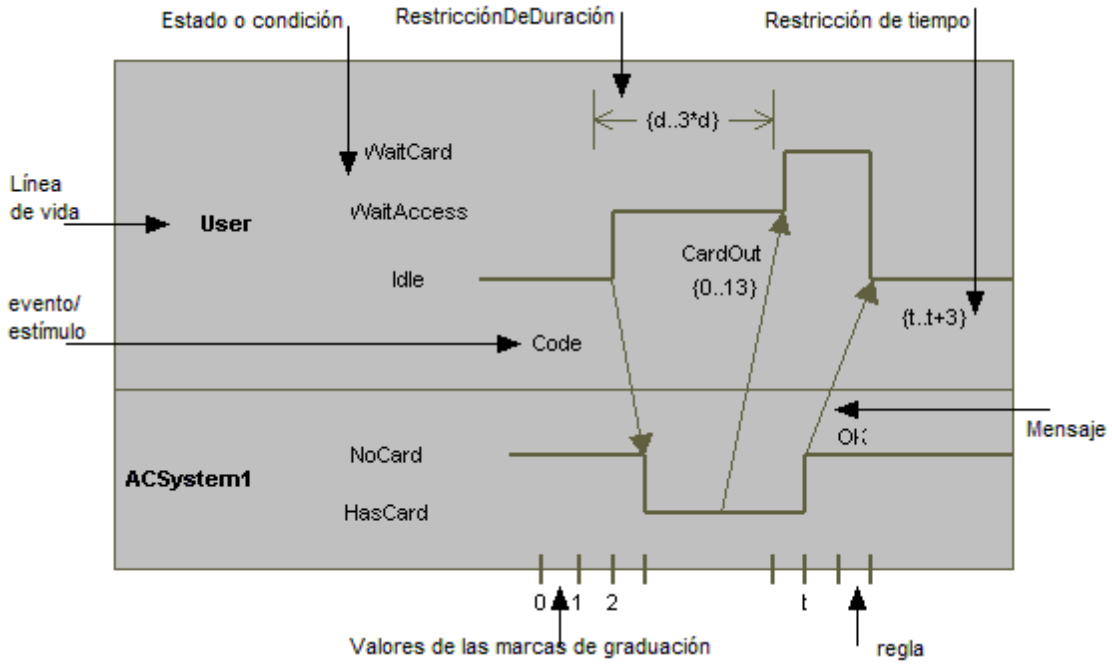
9.1.8 Diagrama de ciclo de vida

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.

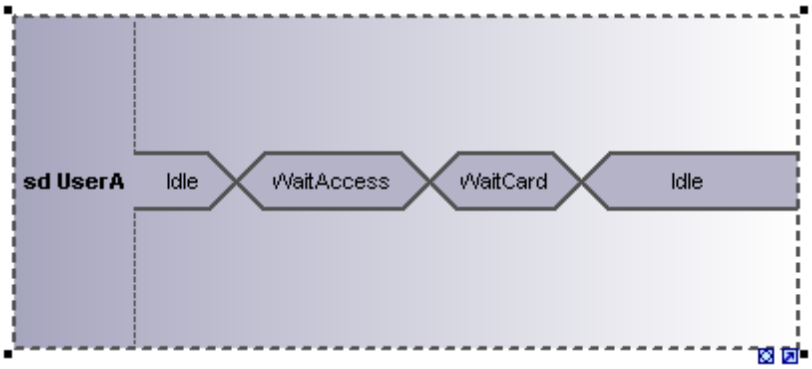
Los diagramas de ciclo de vida son un tipo especial de diagrama de secuencia. La diferencia es que los ejes están invertidos, es decir, el tiempo aumenta de izquierda a derecha, y las líneas de vida aparecen por separado en compartimentos apilados verticalmente.

Además, este tipo de diagramas suelen utilizarse para el diseño de software integrado o de sistemas en tiempo real.

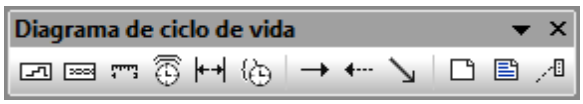


Hay dos tipos de diagramas de ciclo de vida: los que contienen la escala de tiempo del estado/de la condición (*imagen anterior*) y los que muestran el ciclo de vida general (*imagen siguiente*).

© 2018 Altova GmbH



9.1.8.1 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 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.

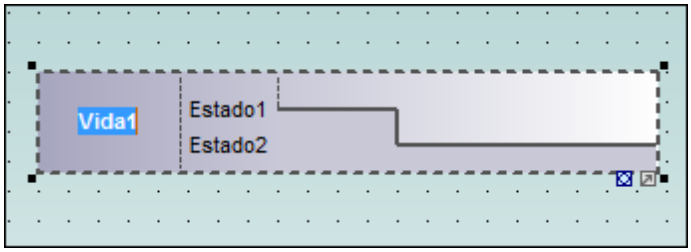
9.1.8.2 Línea de vida

o **Línea de vida**

La línea de vida es un participante de la interacción y tiene dos representaciones distintas: un **Estado/Condición** o un **Valor general**. Pulse **Ctrl+Entrar** para crear una línea nueva en el nombre de la línea de vida.

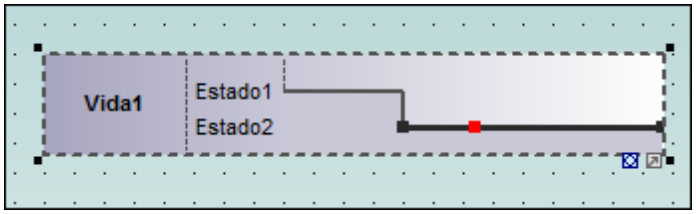
Para insertar un Estado/Condición (InvarianteDeEstado) y definir cambios de estado:

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.



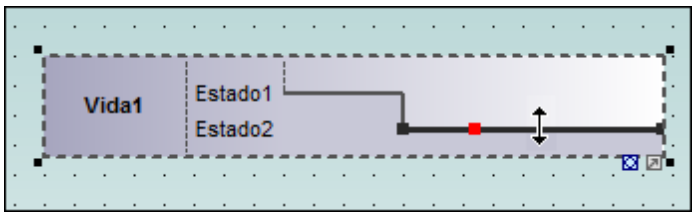
2. Escriba el nombre de la línea de vida o utilice el nombre predeterminado **`Líneadevida1`**.
3. Haga clic en una sección de la línea de tiempo para seleccionarla.
4. Haga clic en la posición de la línea de tiempo donde quiere que se produzca el cambio de estado.

   En este momento aparece una flecha con dos puntas.

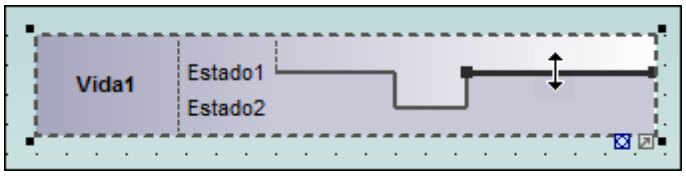


   Y además aparece un recuadro rojo en la posición donde hizo clic, que divide la línea por ese punto.

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.

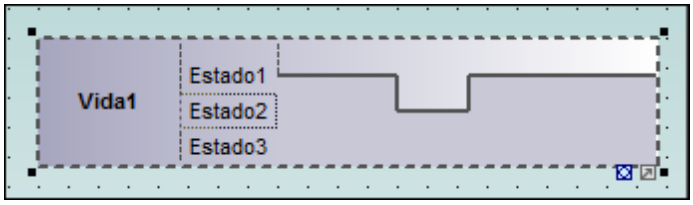


   Puede definir un número ilimitado de cambios de estado en una línea de vida. El recuadro rojo de la línea desaparece al hacer clic en otra parte del diagrama.

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.

 © 2018 Altova GmbH



Para cambiar la posición de un estado (dentro de una línea de vida):

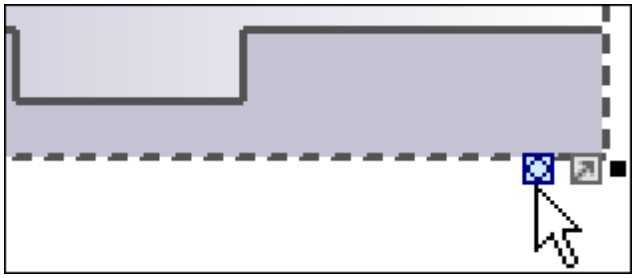
1. Haga clic en la etiqueta del estado.
2. Arrástrela hasta la posición nueva.

Para eliminar un estado de una línea de vida:

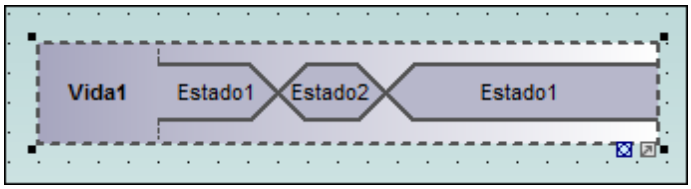
1. Haga clic en la etiqueta del estado y pulse la tecla **Supr**. También puede hacer clic con el botón derecho en el estado y seleccionar el comando **Eliminar**.

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.



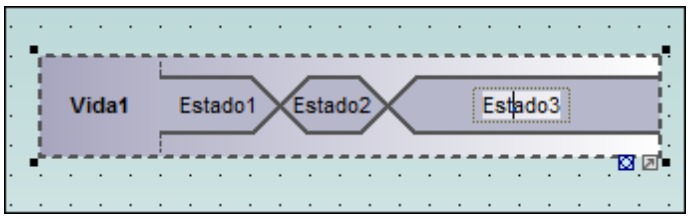
Ahora la línea de vida se presenta con su `Valor general`. Cada punto donde se cruzan las líneas es un cambio de estado/valor.



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:

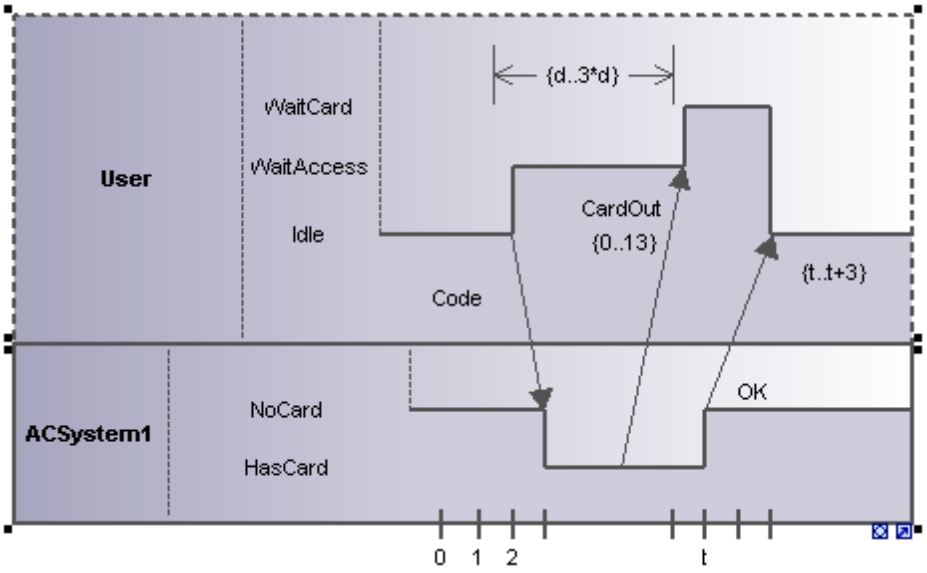
1. Haga clic con el botón derecho en la línea de vida y seleccione **Nuevo/a | Estado o condición (InvarianteDeEstado)**.
2. Edite el nombre del estado nuevo y pulse **Entrar** para confirmar.



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*

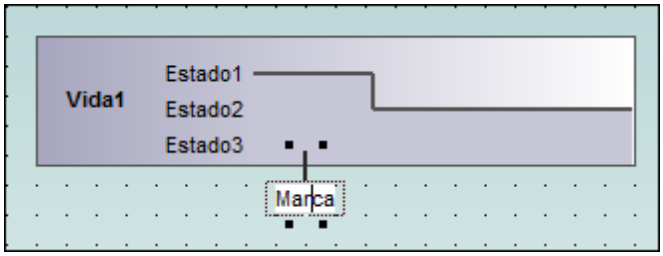
MarcaDeGraduación

Este icono 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.

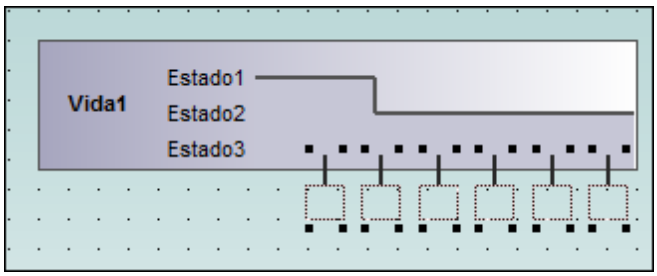
 © 2018 Altova GmbH



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.



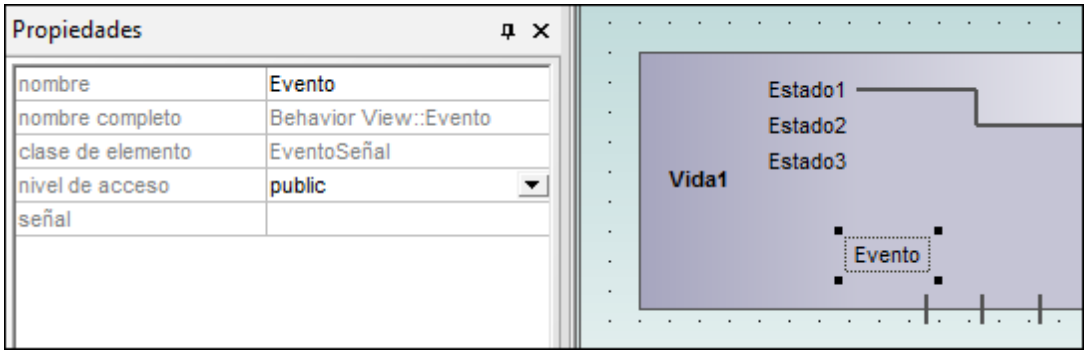
9.1.8.4 Evento/estímulo

Evento/ Estímulo

Este icono 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*.

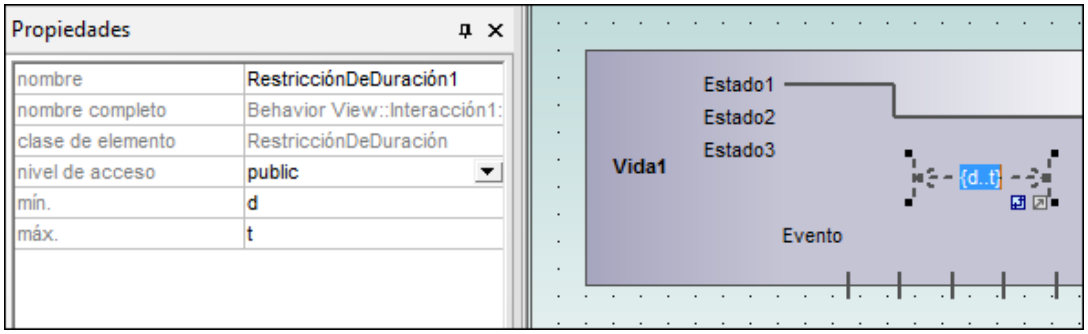
9.1.8.5 Restricción de duración

RestricciónDeDuración

Una duración 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.



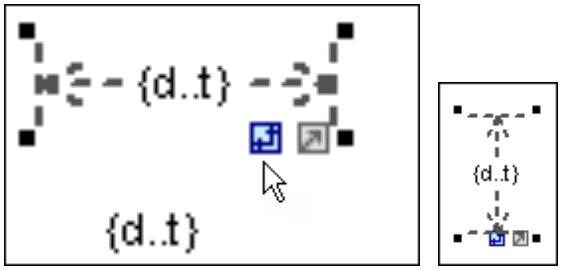
El valor mínimo y máximo predeterminado ("`d..t`") aparece automáticamente. Para editar estos valores haga doble clic en la restricción o edite los valores en la ventana *Propiedades*.

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.

 © 2018 Altova GmbH



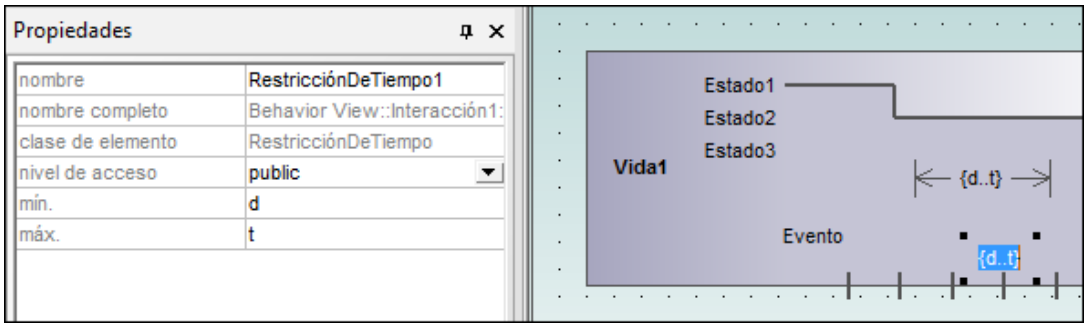
9.1.8.6 Restricción de tiempo

RestricciónDeTiempo

Una restricción de tiempo 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

Mensaje (Llamada) **Mensaje (Respuesta)** **Mensaje asincrónico (Llamada)**

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.

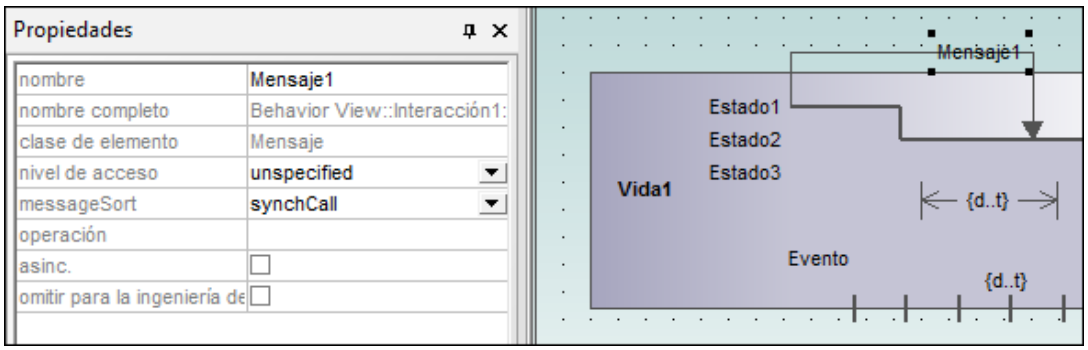
Los mensajes se transmiten entre la escala de tiempo remitente y la escala de tiempo

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.



- 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**.

© 2018 Altova GmbH

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 estructura

- Diagramas de clases
- Diagramas de componentes
- Diagramas de estructura de un compuesto
- 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 forma esférica (Ball and socket)
- Agregar excepciones emitidas a los métodos de una clase
- 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.

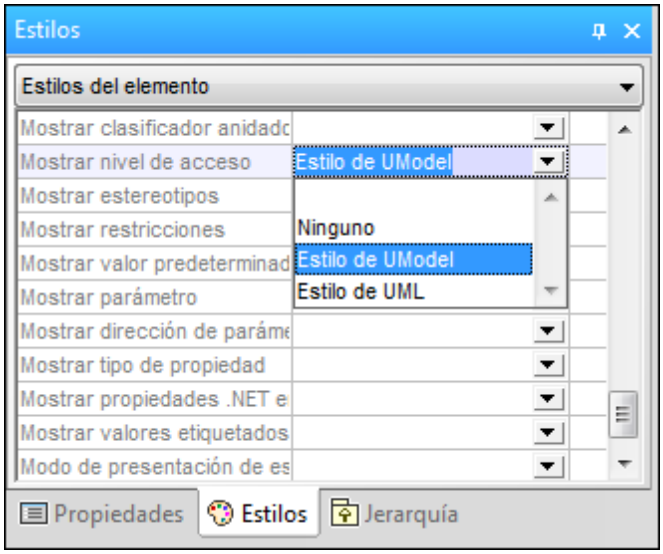
Haga clic en el paquete / en la clase que desea **expandir** y:
- Pulse la tecla ***** para expandir el paquete/la clase actual y todos los subelementos.
- Pulse la tecla **+** para abrir el paquete/la clase actual.

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.

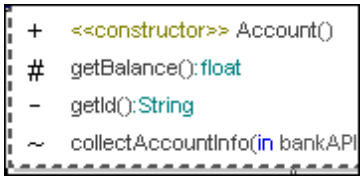
Cambiar el icono de nivel de acceso
Haga clic en el icono de nivel de acceso situado a la izquierda de una operación o propiedad para abrir una lista desplegable en la que puede elegir el nivel de acceso del elemento.

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 la 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 UModel también puede elegir qué atributos y operaciones de una clase aparecen en el diagrama y elegir qué nuevos atributos y operaciones se muestran cuando se añaden.

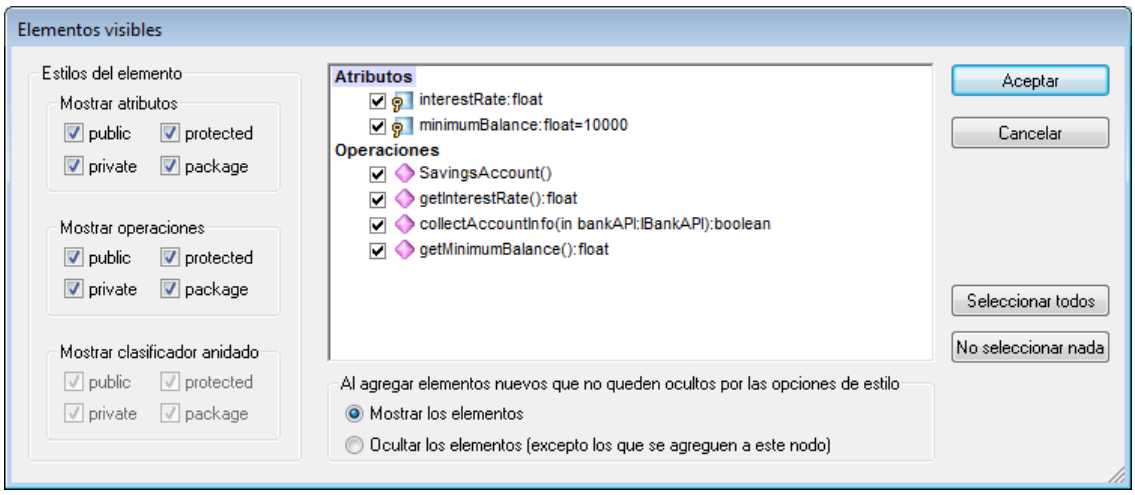
No olvide que los slots de objetos (es decir, EspecificacionesDeInstancia) se pueden mostrar/

 © 2018 Altova GmbH

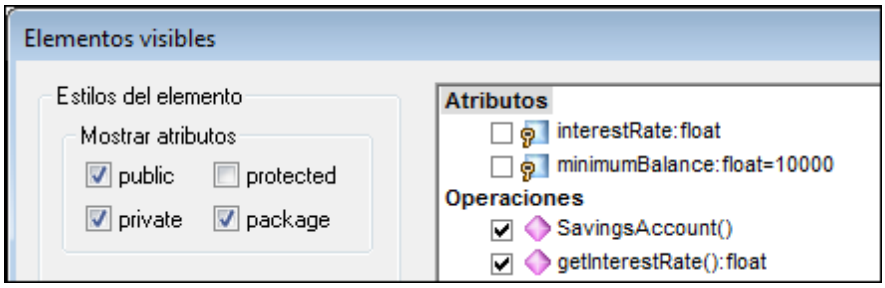
ocultar de la misma manera.

Haga clic con el botón derecho en una clase (p. ej. `SavingsAccount`) y elija el comando **Mostrar u ocultar el contenido del nodo** del menú contextual.

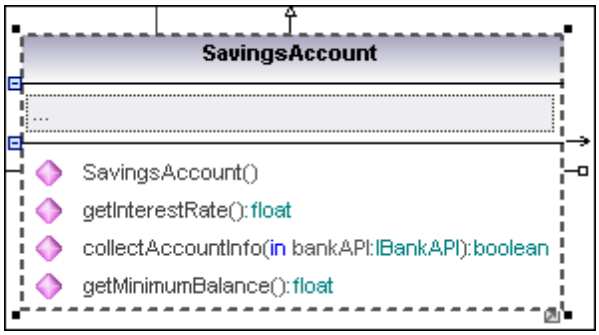
Esto abre el cuadro de diálogo "Elementos visibles".



Si desactiva la casilla *protected* del grupo *Mostrar atributos*, los atributos que tengan el nivel `protected` se desactivan en la vista previa del cuadro de diálogo.



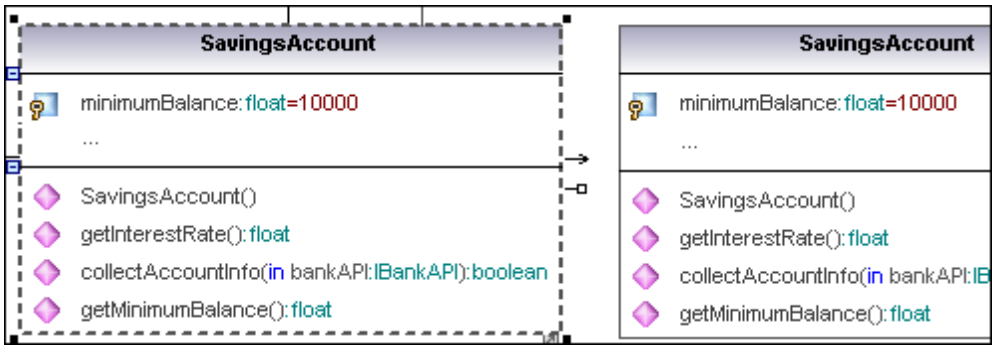
Haga clic en **Aceptar** para cerrar el cuadro de diálogo. Los atributos de la clase que tienen el nivel `protected` se reemplazan con tres puntos (...). Haga doble clic en los tres puntos para abrir otra vez el cuadro de diálogo "Elementos visibles".



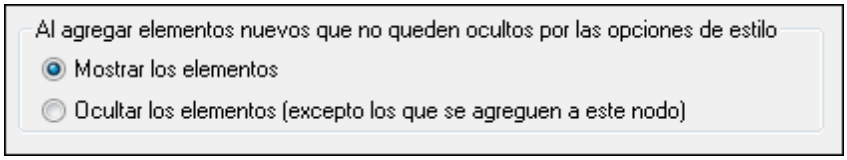
No olvide que en la vista previa del cuadro de diálogo puede desactivar los atributos uno a uno.

Mostrar/ocultar atributos y operaciones (Estilos del elemento)
En UModel puede insertar varias instancias de la misma clase en un el mismo diagrama o

incluso en diagramas distintos. Cada una de las vistas de esta clase puede tener niveles de acceso diferentes. Por ejemplo, la imagen siguiente muestra dos vistas de la misma clase: `SavingsAccount`.

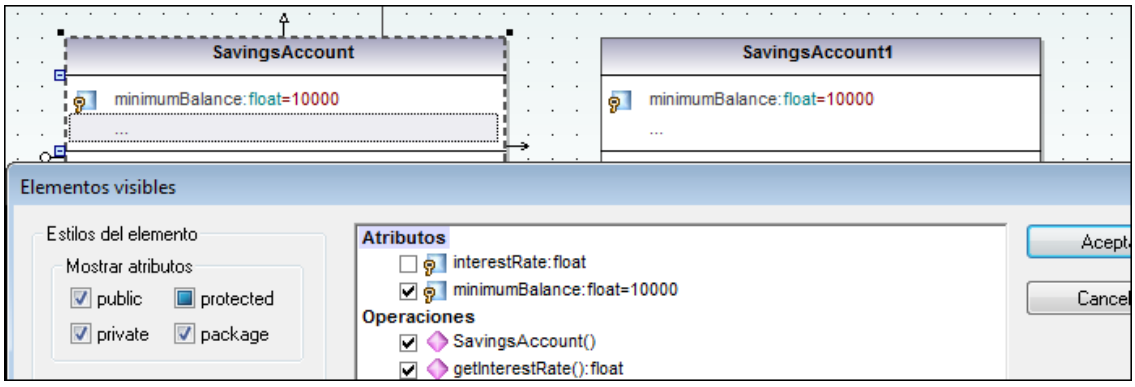


En el cuadro de diálogo "Elementos visibles" las opciones del grupo *Al agregar elementos nuevos que no queden ocultos por las opciones de estilo* sirven para definir qué será visible cuando se añadan elementos nuevos a la clase. Los elementos se pueden añadir a mano en el diagrama o en la *Estructura del modelo* o automáticamente durante el proceso de ingeniería de código.



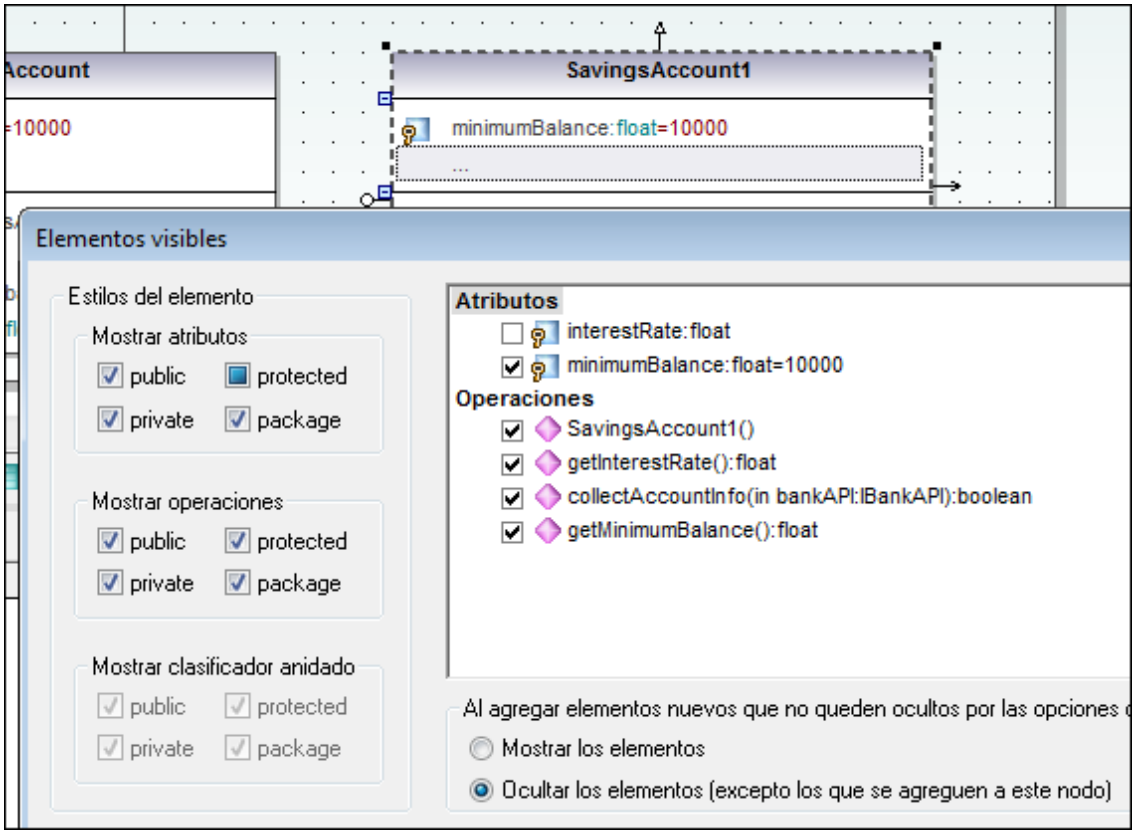
Mostrar los elementos: elija esta opción si quiere que se muestren todos los elementos que se añadan a cualquier vista de la clase.

P. ej. el atributo `interestRate:float` se ocultó en ambas vistas de **`SavingsAccount`** y se dejó visible el atributo `minimumBalance`. El botón de opción *Mostrar los elementos* está activo para la vista izquierda de la clase. Al hacer doble clic en los tres puntos (...) del compartimento del atributo de la vista **izquierda** de la clase, se abre el cuadro de diálogo "Elementos visibles", donde podemos ver que el botón de opción *Mostrar los elementos* está activo.

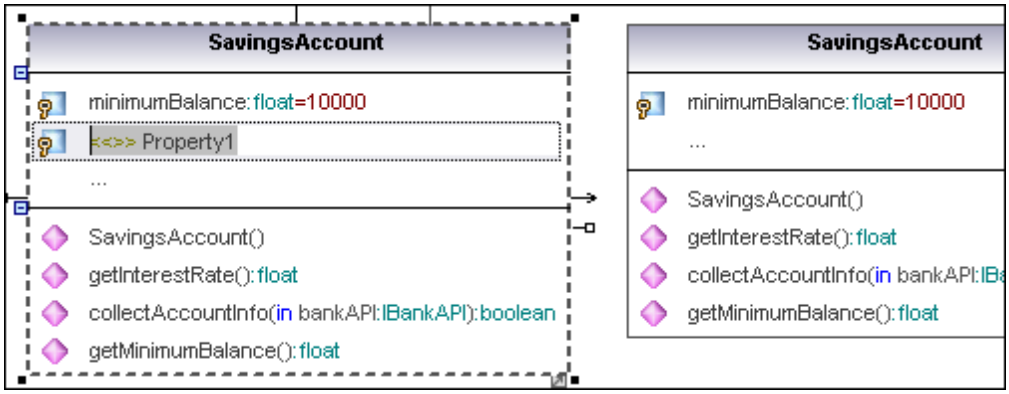


Al hacer doble clic en los tres puntos (...) del compartimento del atributo de la vista **derecha** de la clase, se abre el cuadro de diálogo "Elementos visibles", donde podemos ver que está activo el botón de opción *Ocultar los elementos (excepto los que se agreguen a este nodo)*.

 © 2018 Altova GmbH

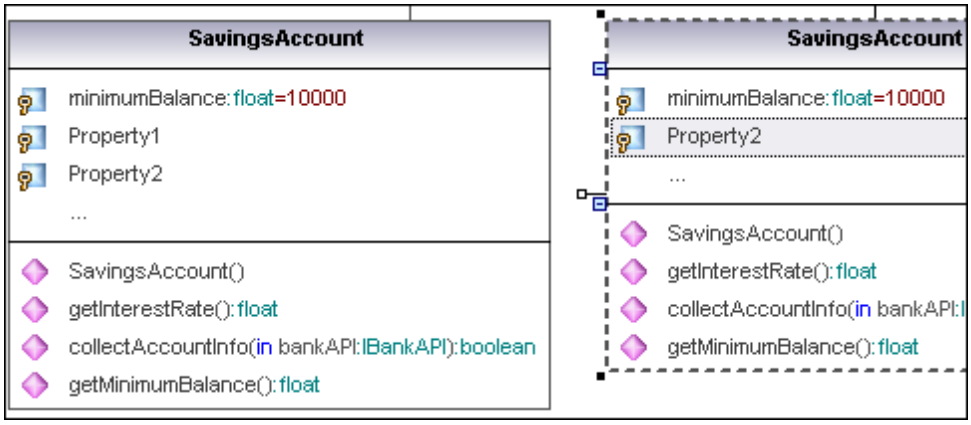


Si hacemos clic en la vista **izquierda** de la clase y pulsamos **F7** (o si hacemos clic en la clase en la *Estructura del modelo* y pulsamos **F7**), en la clase se añade un atributo nuevo (`Propiedad1`).



El nuevo elemento solamente está visible en la vista **izquierda** de la clase (porque en esta vista está activa la opción *Mostrar los elementos*). En la vista **derecha** de la clase el elemento está oculto (porque en esa vista está activa la opción *Ocultar los elementos*).

Si hacemos clic en la vista derecha de la clase y pulsamos F7, en la clase se añade un atributo nuevo (`Property2`). Este atributo nuevo está visible en la vista derecha porque la opción oculta los elementos *excepto los que se agreguen a este nodo*. Aquí *nodo* significa esta clase o elemento de modelado.

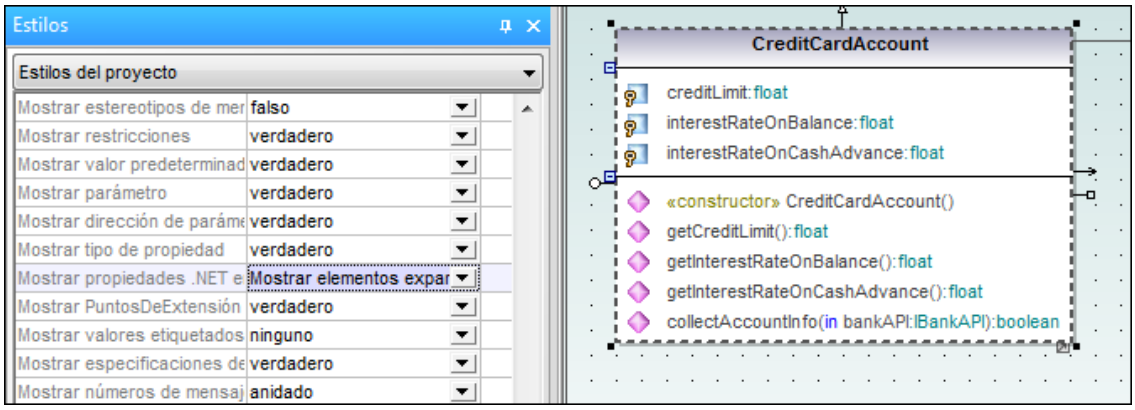


Ahora el atributo `Property2` está visible en la vista izquierda de la clase (porque en esta vista está activa la opción *Mostrar los elementos*).

Nota: los valores etiquetados de los elementos ocultos también se ocultan si se elige la opción *Ocultar los elementos*.

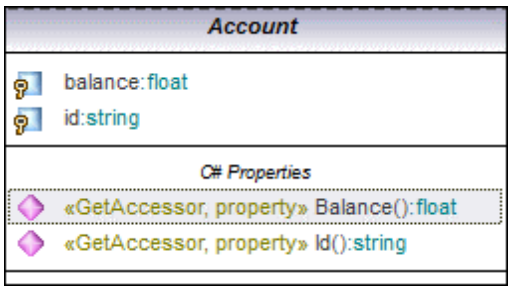
Mostrar/ocultar compartimentos VS .NET

UModel puede mostrar las propiedades .NET en un compartimento separado. Para ello, habilite el estilo `Mostrar propiedades .NET en un compartimento propio` de la ventana *Estilos*.



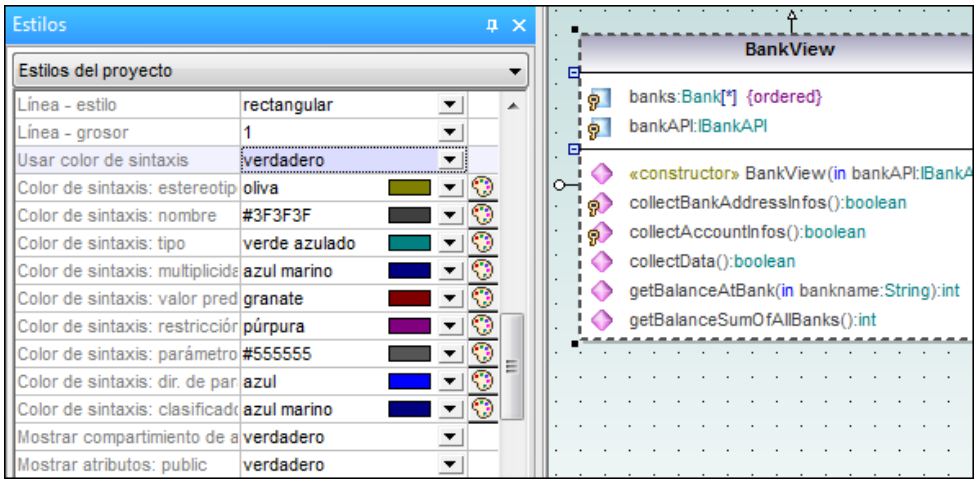
Mostrar las propiedades VS .NET como asociaciones

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.



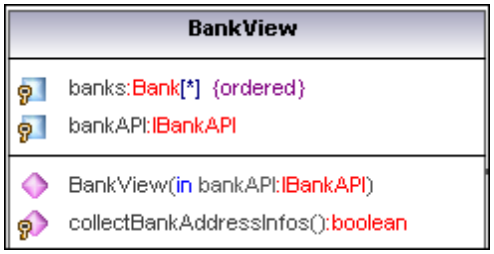
 © 2018 Altova GmbH

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.



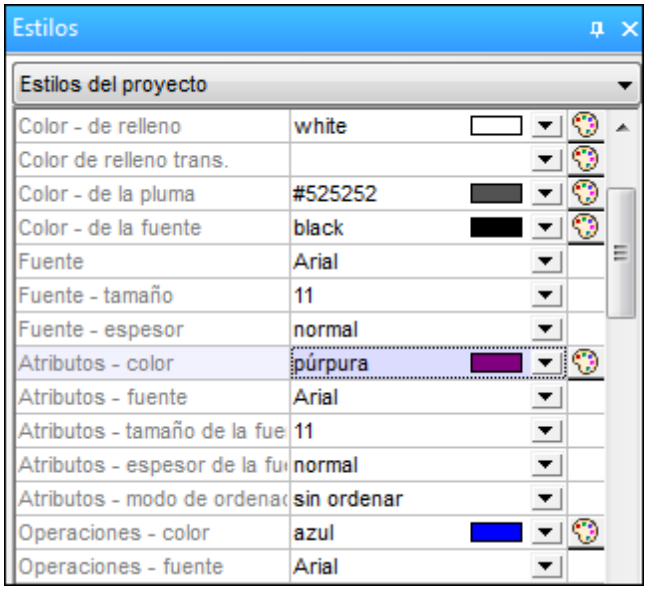
Para cambiar las opciones predeterminadas de color de sintaxis:

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.

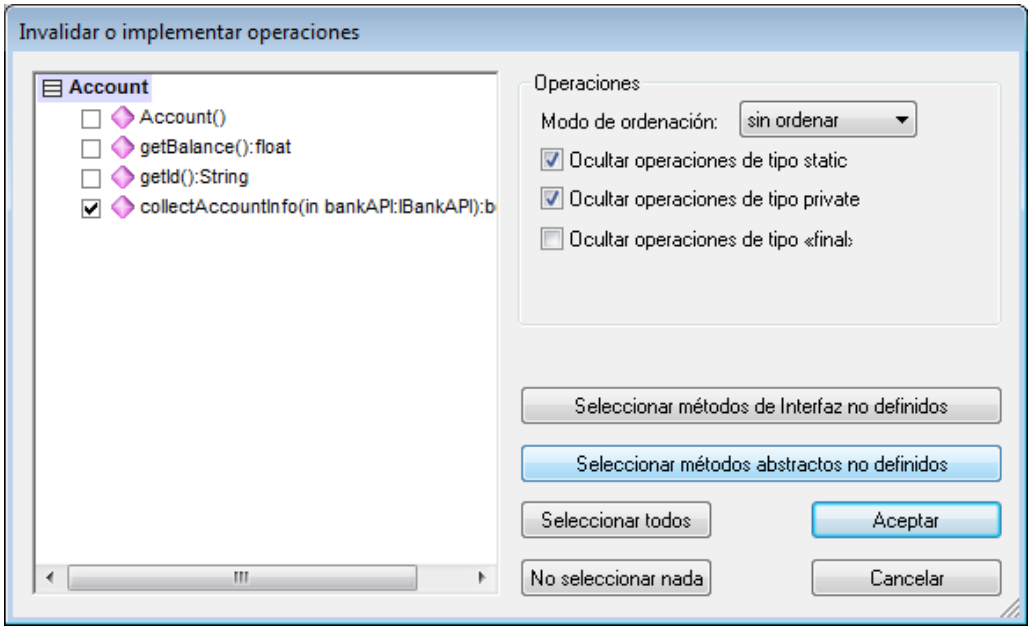


9.2.1.2 Invalidar operaciones de clases base e implementar operaciones de interfaz

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*).

© 2018 Altova GmbH



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.

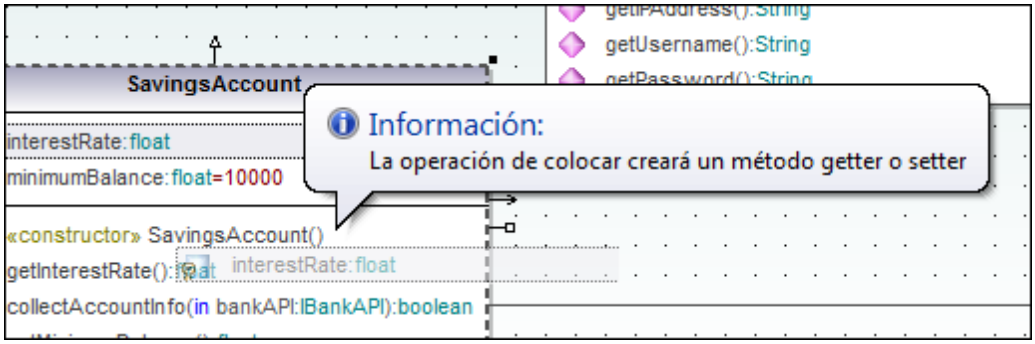
9.2.1.3 Crear métodos getter y setter

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:

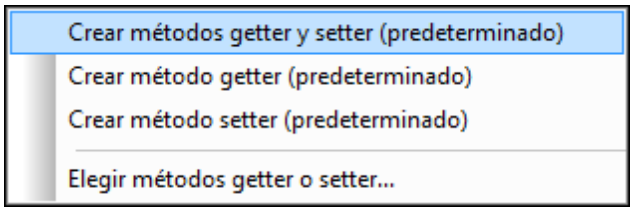
- Arrastrando y colocando un atributo en el compartimento de una operación.
- Usando el menú contextual para abrir un cuadro de diálogo donde puede gestionar los métodos getter/setter.

Para crear métodos getter/setter mediante operaciones arrastrar y colocar:

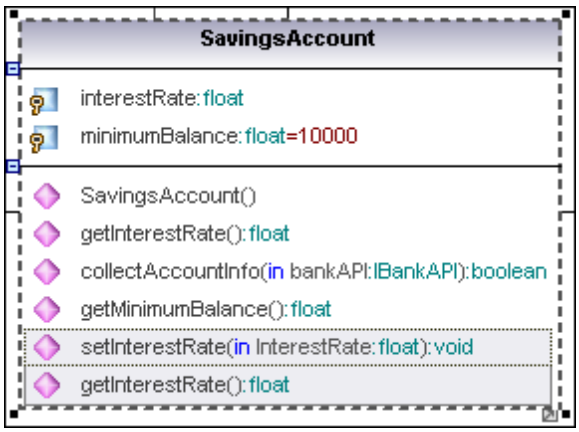
1. Arrastre un atributo desde el compartimento de atributos hasta el compartimento de operaciones.



Aparece un menú contextual donde puede elegir el tipo de método getter/setter que se debe crear.



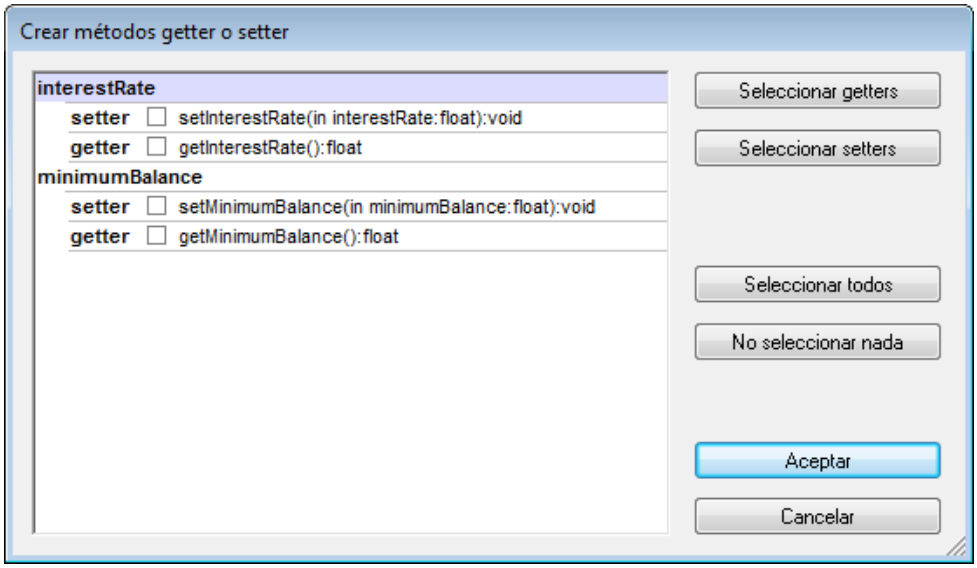
2. Seleccione el primer comando del menú para crear un método getter y setter para interestRate:float.



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.

 © 2018 Altova GmbH

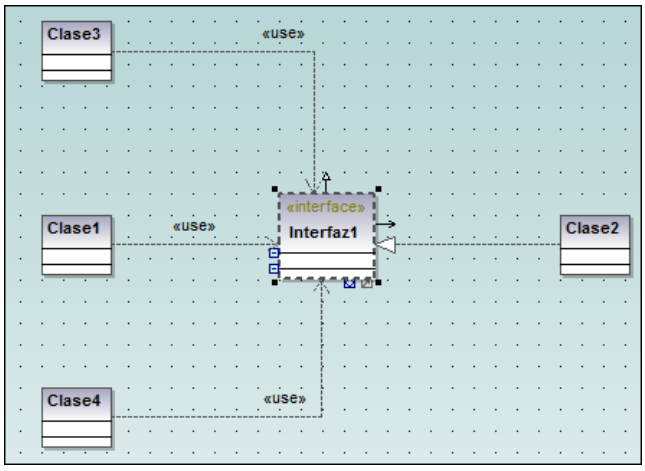


2. Use los botones o marque las casillas para seleccionar los métodos que desea crear.

   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 en forma esférica de UML. Las clases que necesitan una interfaz muestran un círculo y el nombre de la interfaz, mientras que las clases que implementan la interfaz muestran una esfera.

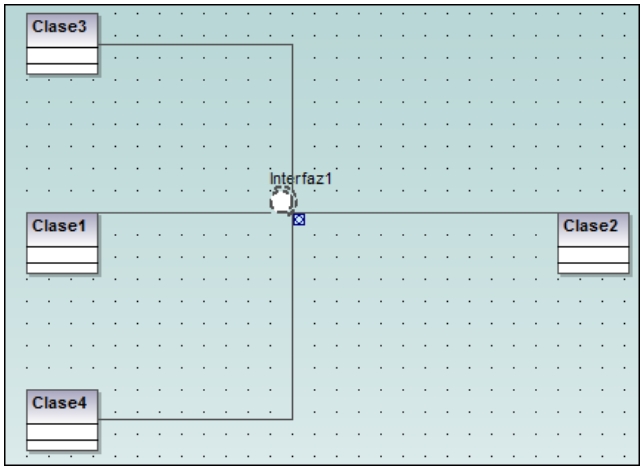


En la imagen anterior, la `Clase2` realiza la `Interfaz1`, que es utilizada por las clases `clase1`, `clase3` y `clase4`. Para crear la relación de utilización entre las clases e interfaz se utilizó el icono **Utilización** de la barra de herramientas Diagrama de clases.

Para cambiar entre la vista estándar y la vista esférica:

- Haga clic en el icono **Alternar estilo de notación de interfaz** situado en la parte

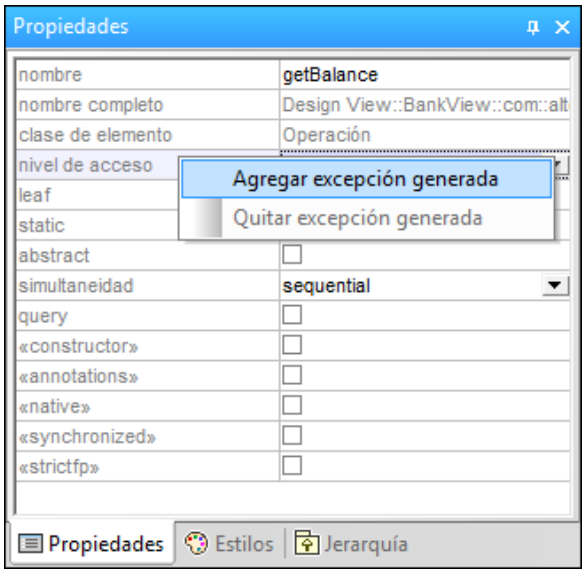
inferior del elemento interfaz.



9.2.1.5 Agregar excepciones emitidas a los métodos de una clase

Para agregar excepciones generadas a los métodos de una clase:

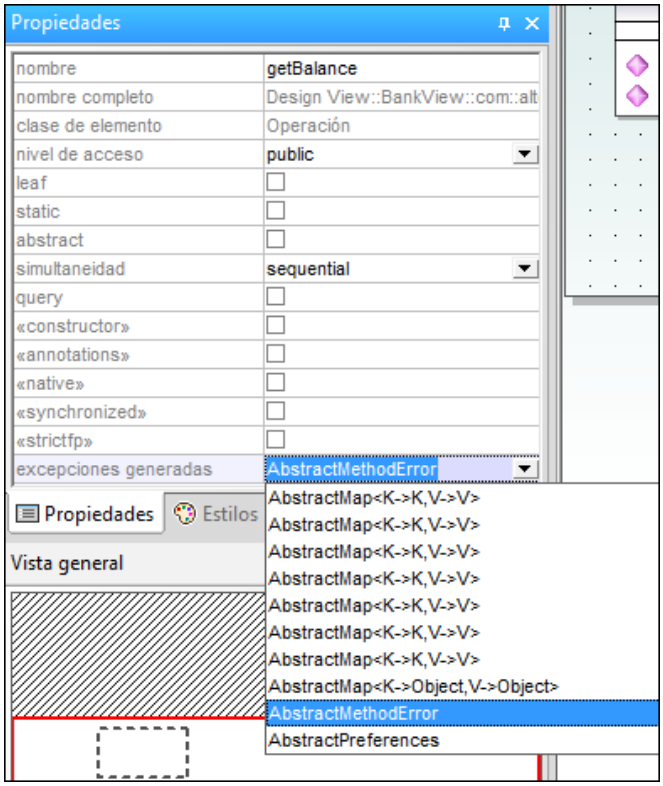
1. En la *Estructura del modelo* haga clic en el método de la clase en la que desea agregar la excepción generada (p. ej. el método **`getBalance`** de la clase **`Account`**).
2. Ahora haga clic con el botón derecho dentro de la ventana *Propiedades* y elija la opción **Agregar excepción generada** 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.

 © 2018 Altova GmbH

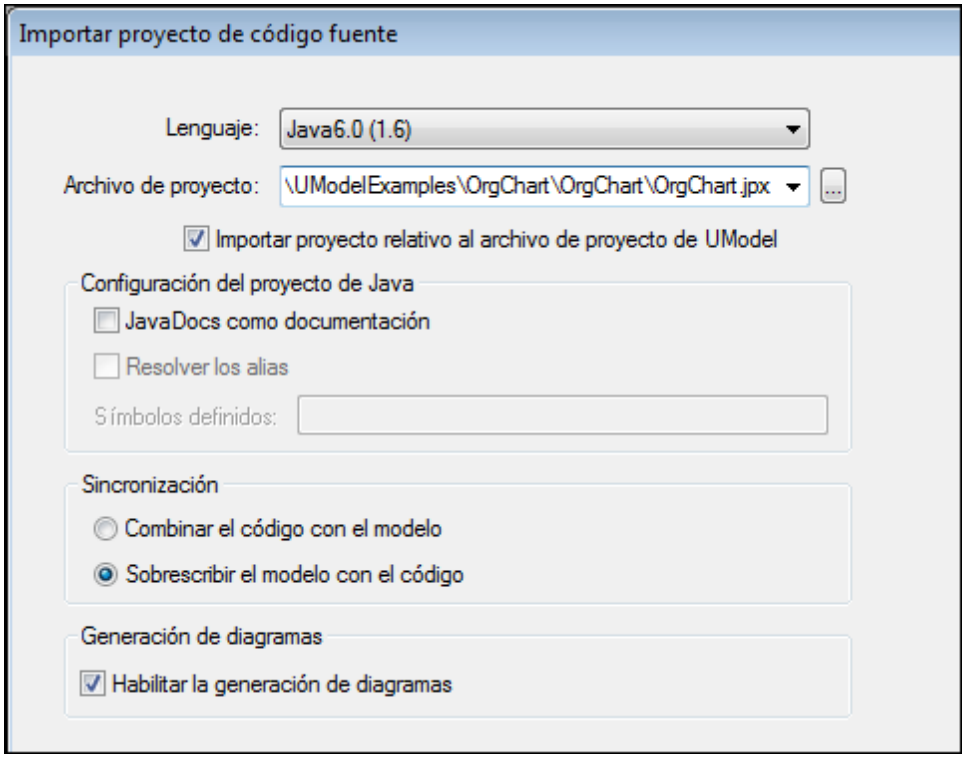


9.2.1.6 Generar diagramas de clases

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 binarios Java, C# y VB).

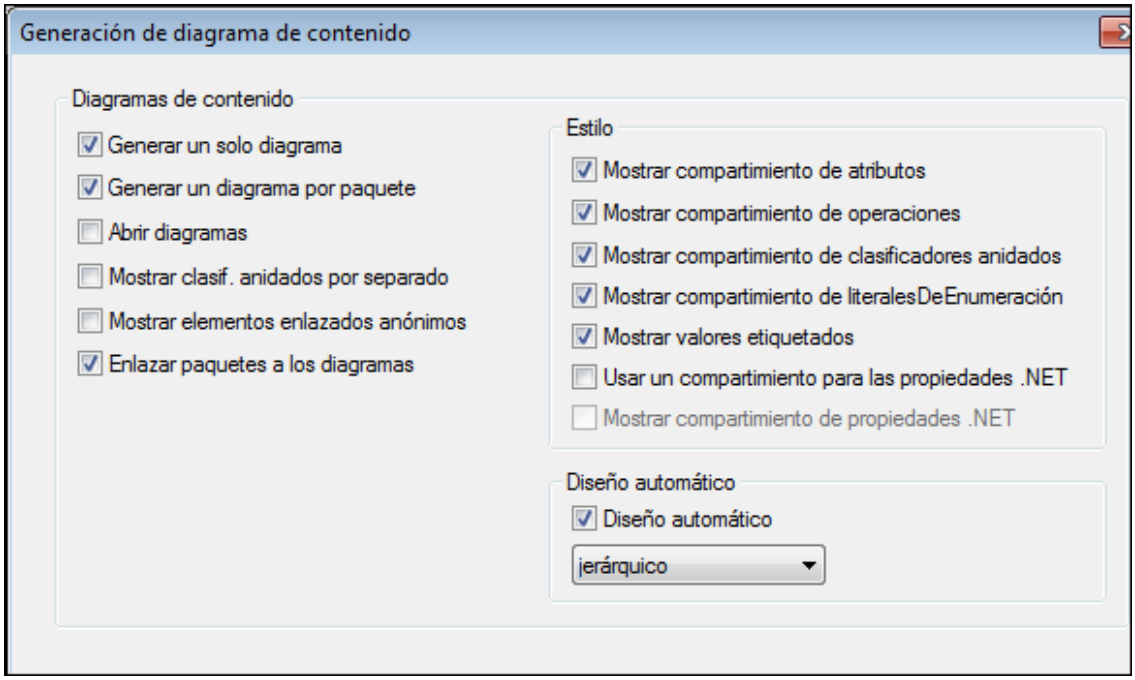
Siga las instrucciones del asistente para la importación y asegúrese de que:

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".

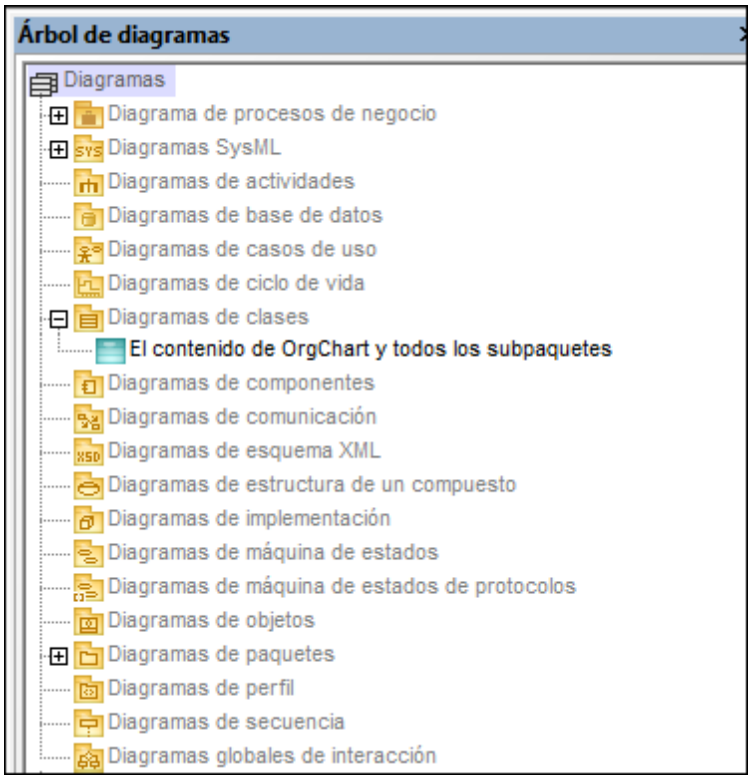


2) Las opciones *Generar un solo diagrama* y *Generar un diagrama por paquete* están marcadas en el cuadro de diálogo "Generación de diagrama de contenido".

 © 2018 Altova GmbH



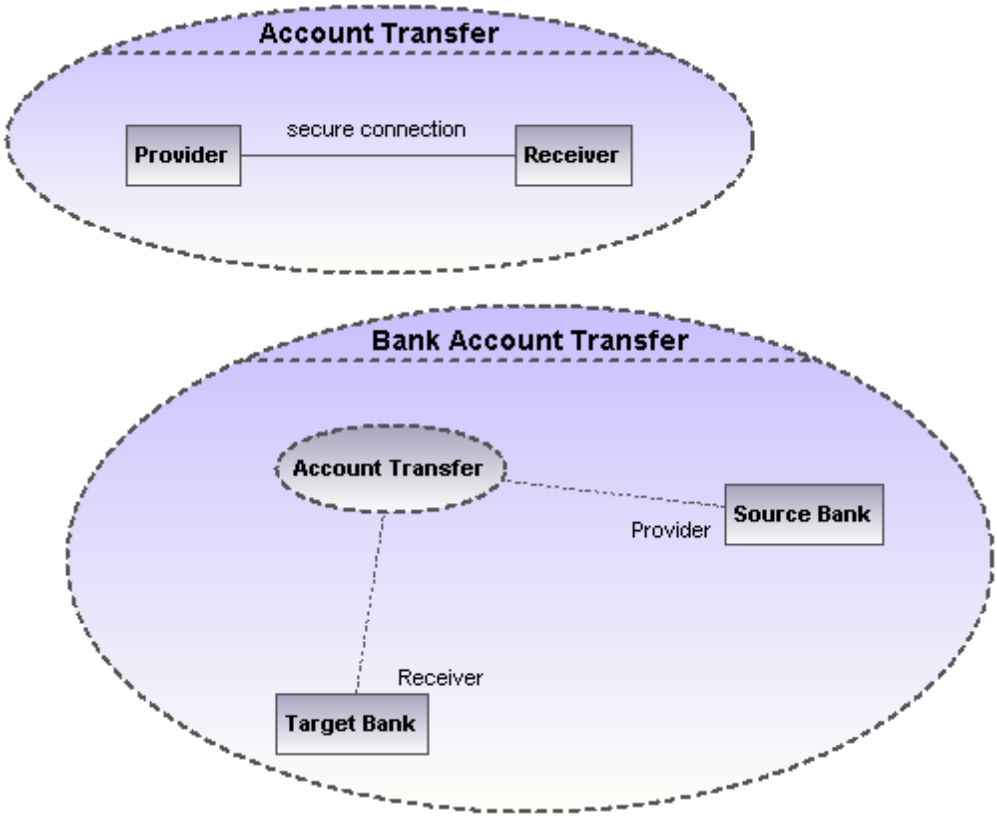
3) Una vez finalizada la operación de importación, los diagramas de clases generados estarán disponibles bajo el nodo `Diagramas de clases` del *Árbol de diagramas*.



9.2.2 Diagrama de estructura de un compuesto

Sitio web de Altova: diagramas de estructura de un compuesto UML

Los diagramas de estructura de un compuesto 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

Para insertar elementos con los iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de estructura de un compuesto.
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.

© 2018 Altova GmbH

Para arrastrar elementos desde la Estructura del modelo hasta el diagrama de estructura de un compuesto:

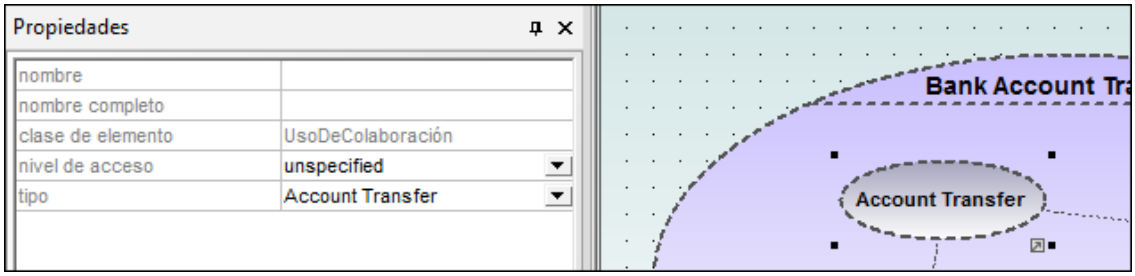
1. En la *Estructura del modelo* busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de estructura de un compuesto.

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

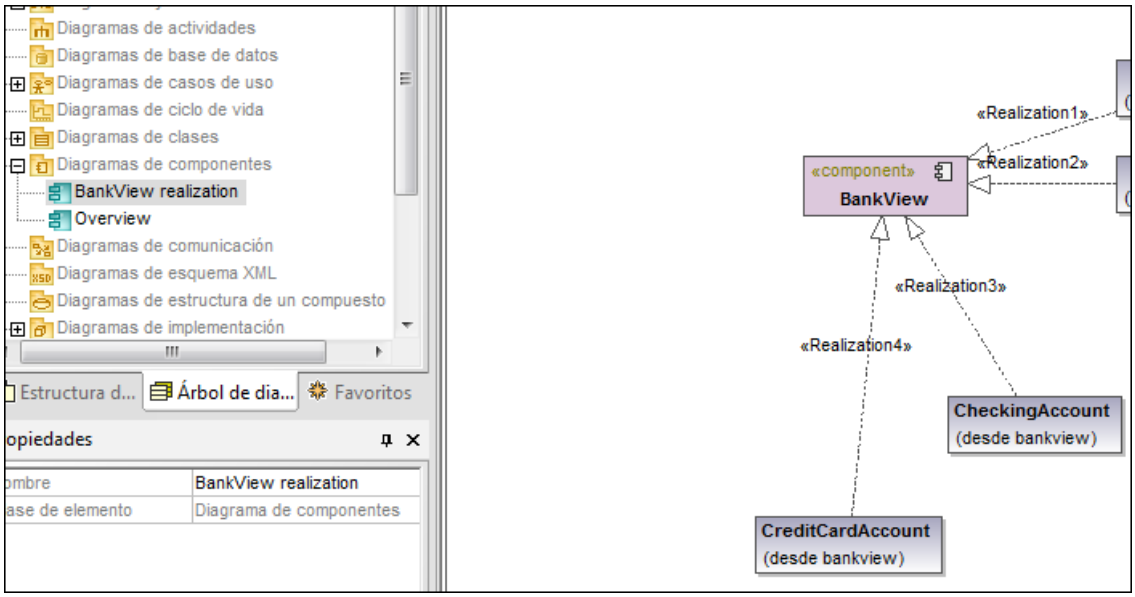
parte o de un puerto. El conector define la relación entre los objetos e identifica la comunicación entre los roles.

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

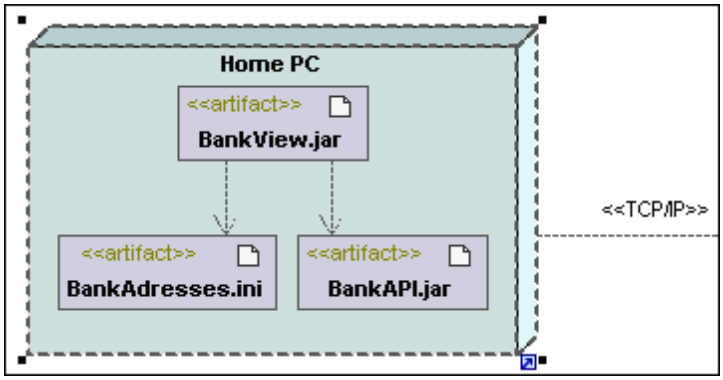
Para más información sobre cómo añadir componentes al diagrama consulte el apartado Diagramas de componentes.



9.2.4 Diagrama de implementación

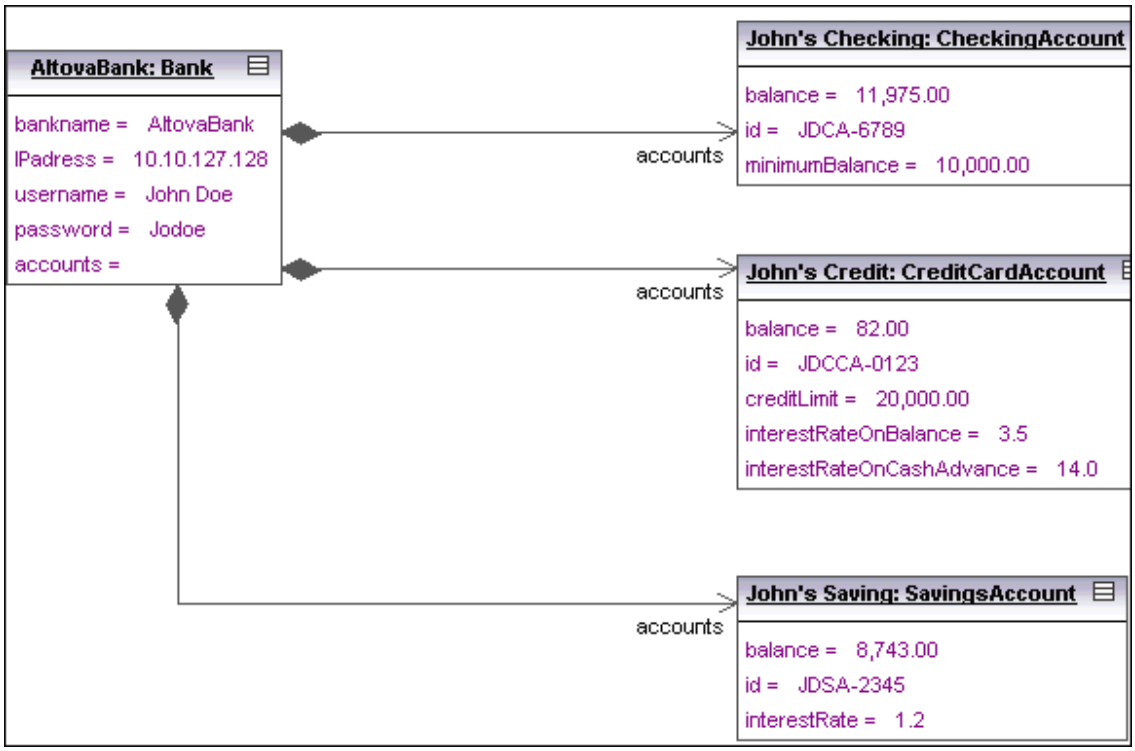
Para más información sobre cómo agregar nodos y artefactos al diagrama consulte el apartado Diagramas de implementación del tutorial.

© 2018 Altova GmbH



9.2.5 Diagrama de objetos

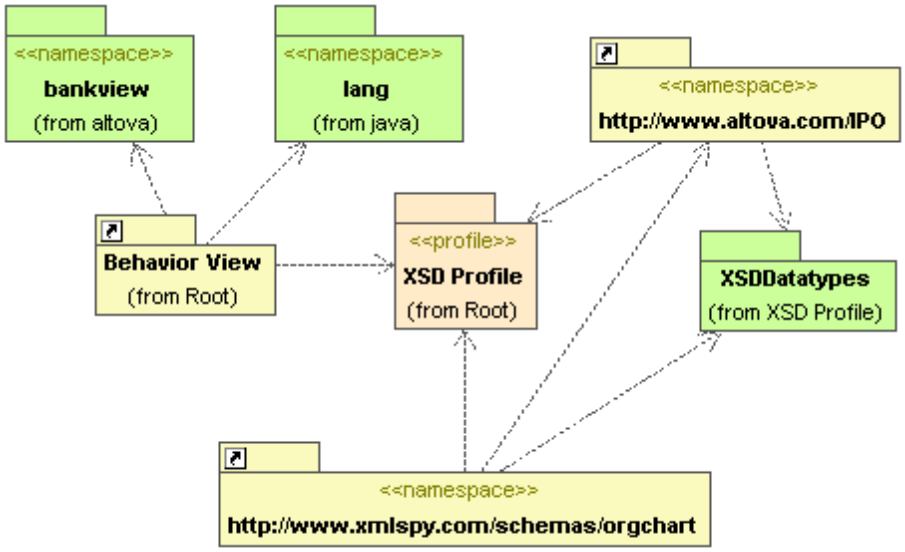
Para más información sobre cómo agregar objetos/instancias nuevos al diagrama consulte el apartado Diagramas de objetos del tutorial.



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

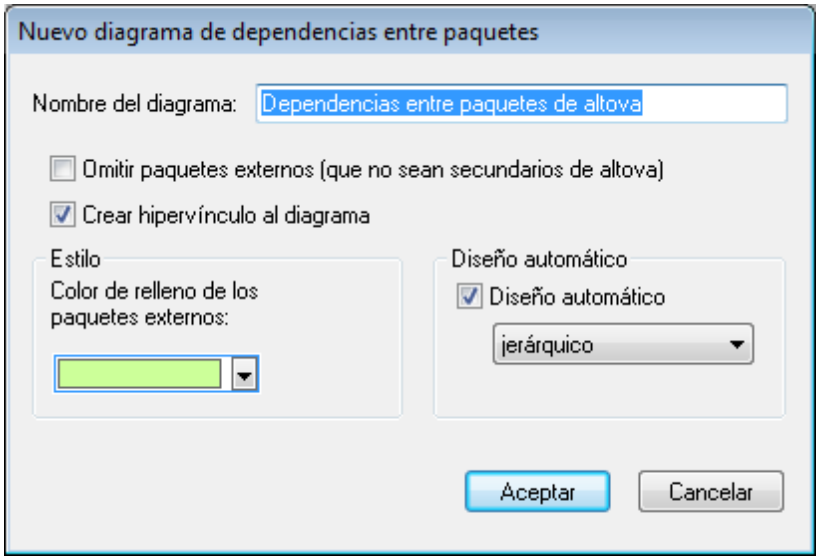
UModel tiene una función para generar diagramas de dependencias entre paquetes a partir de cualquier paquete de la ventana *Estructura del modelo*.

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.

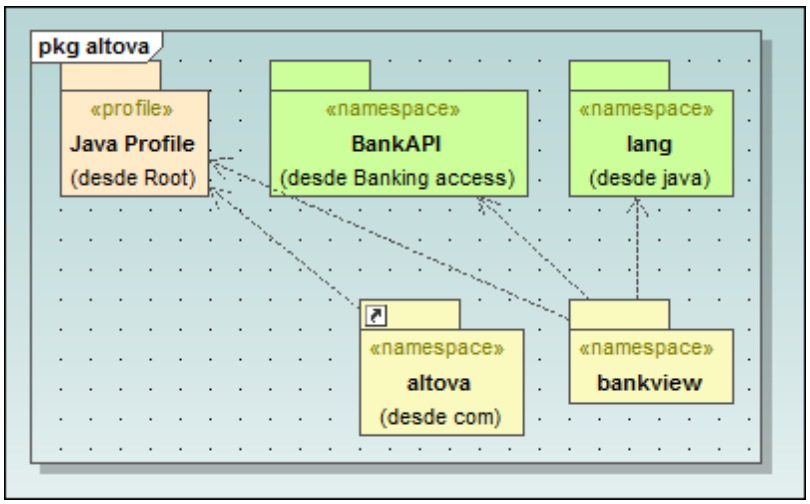
Para generar un diagrama de dependencias entre paquetes:

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".

© 2018 Altova GmbH



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

Para 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.

Para 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

 © 2018 Altova GmbH

disposición del paquete.

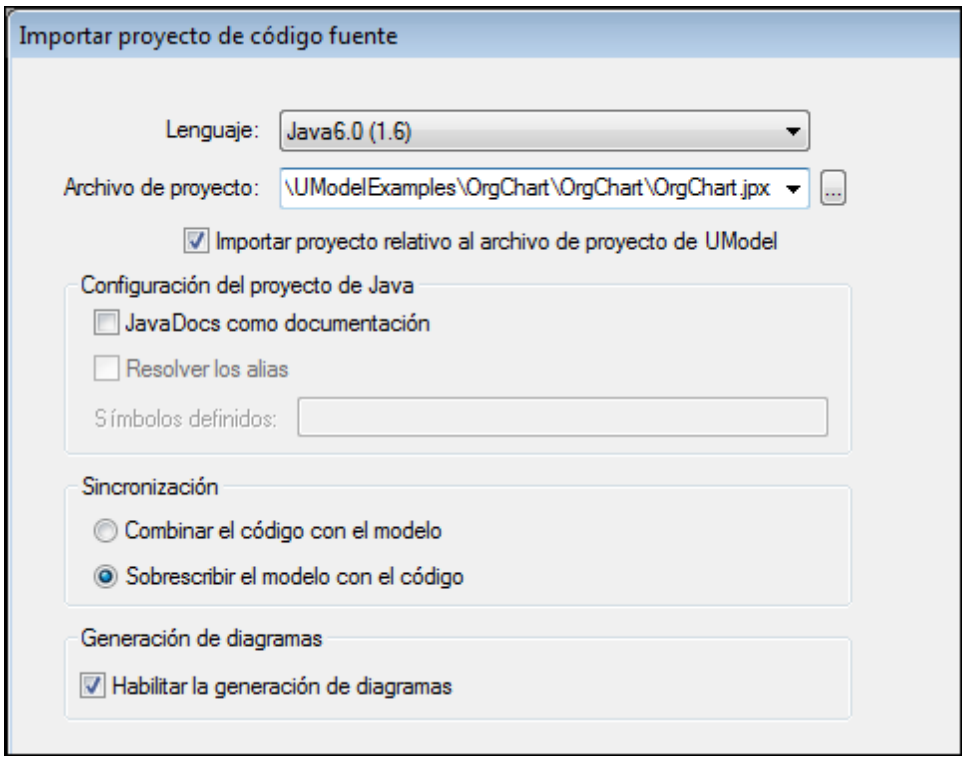
Los nombres de perfil aparecen en forma de líneas discontinuas con puntas de flecha que van del paquete hasta el perfil aplicado y llevan la etiqueta `<<apply>>`.

9.2.6.2 *Generar diagramas de paquetes al importar código o binarios*

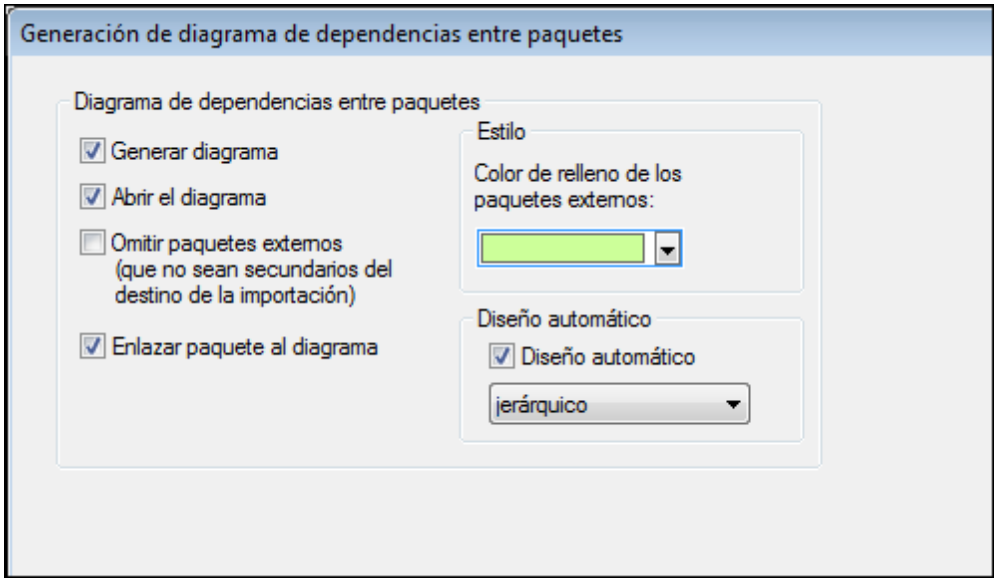
Con UModel puede generar diagramas de paquetes al importar código fuente o binarios en el proyecto de UModel (consulte los apartados Importar código fuente e Importar binarios Java, C# y VB).

Siga las instrucciones del asistente para la importación y asegúrese de que:

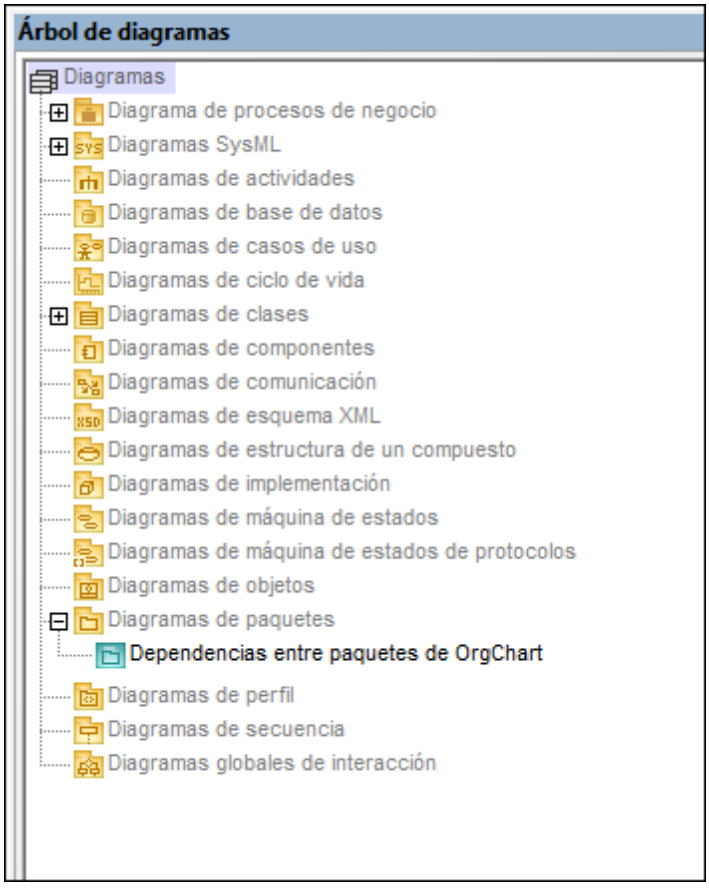
1) La casilla *Habilitar generación de diagramas* está marcada en el cuadro de diálogo "Importar proyecto de código fuente", "Importar tipos binarios" o "Importar directorio de código fuente".



2) La opción *Generar diagrama* está marcada en el cuadro de diálogo "Generación de diagrama de dependencias entre paquetes".



3) Una vez finalizada la operación de importación, los diagramas de clases generados estarán disponibles bajo el nodo `Diagramas de paquetes` del *Árbol de diagramas*.



 © 2018 Altova GmbH

9.2.7 Diagrama de perfil y estereotipos

Sitio web de Altova: Diagramas de perfil UML

Con los diagramas de perfil de UModel puede definir estereotipos, valores etiquetados y restricciones personales en un diagrama especial.

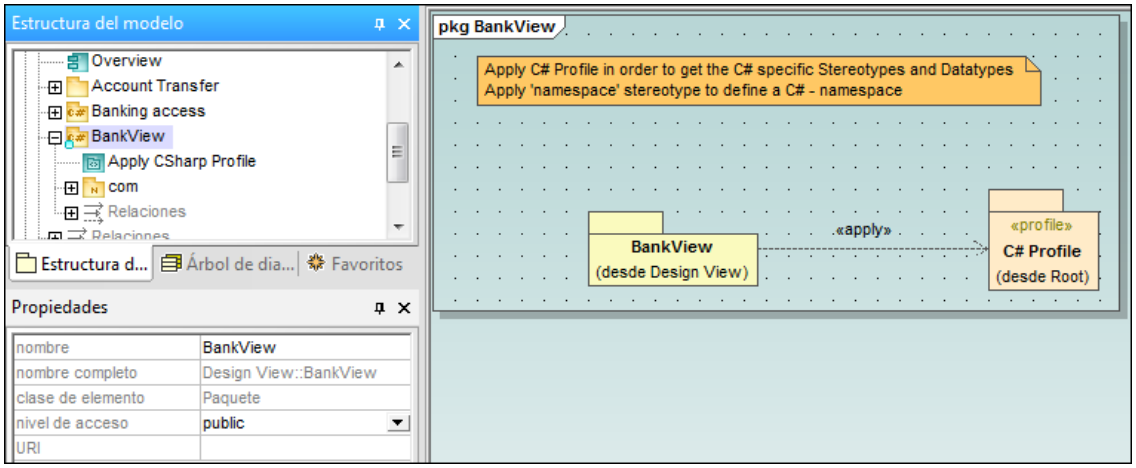
Los perfiles y los estereotipos sirven para extender el metamodelo UML. La construcción de extensión principal es el estereotipo, que forma parte del perfil. Los perfiles siempre deben estar relacionados con un metamodelo de referencia como UML porque no pueden existir por sí solos. UModel ofrece un tipo de diagrama de perfil donde puede definir estereotipos propios.

El archivo de ejemplo `Java Profile.ump` (o `C# Profile.ump` o `VB Profile.ump`) debe aplicarse cuando se creen proyectos nuevos de UModel con el comando de menú **Proyecto | Incluir un subproyecto**. Este perfil aporta los tipos de datos y estereotipos Java y es fundamental para crear código mediante ingeniería de ida y vuelta.

En esta sección explicamos:

- Cómo agregar estereotipos y definir valores etiquetados
- Los estereotipos y las enumeraciones
- Los estilos de estereotipo definidos por el usuario

Para explicar todo ello utilizamos el archivo de ejemplo `Bank_CSharp.ump` (de la carpeta `...\UModelExamples`). El perfil C# se aplicó al paquete **`BankView`**.



- Los perfiles son tipos de paquetes concretos que se aplican a otros paquetes.
- Los estereotipos son metaclases concretas que extienden las clases estándar.
- Los valores etiquetados son los valores de los atributos del estereotipo.

Una `AplicaciónDePerfil` muestra qué perfiles se aplicaron a un paquete y es 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. Si se aplica un perfil a un paquete (con ayuda de **AplicaciónDePerfil**), significa que todos los estereotipos que forman parte del perfil estarán a disposición del paquete.

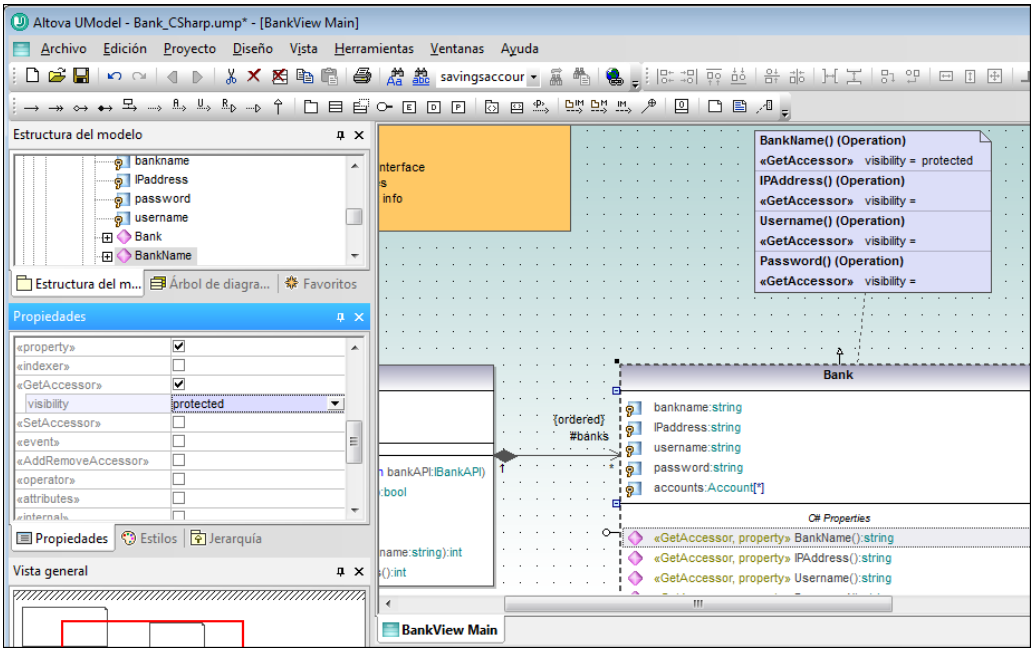
Las `AplicacionesDePerfil` se dibujan como líneas de puntos con punta de flecha que van

desde el paquete hasta el perfil aplicado y tienen la etiqueta `<<apply>>`.

,

Estereotipos:
Un estereotipo define cómo se puede extender una metaclase. Se trata de un tipo de clase que extiende otras clases por medio de extensiones. Los estereotipos solamente se pueden crear en perfiles y se representan en diagramas de clases como clases estándar, pero con la palabra clave `<<stereotype>>` encima del nombre de la clase.

- Los estereotipos pueden tener **propiedades**, llamadas **definiciones de etiquetas**.
- Cuando se aplica el estereotipo a un elemento de modelado, los valores de las propiedades reciben el nombre de **valores etiquetados**.
- Cuando se aplican estereotipos que tienen propiedades, los valores etiquetados aparecen automáticamente en un comentario. Para más información sobre cómo personalizar la vista de valores etiquetados consulte este apartado.
- Los estereotipos tienen su propia familia de estilos
- Si el atributo es de tipo `enumeration`, entonces podrá seleccionar uno de los valores predefinidos. También puede insertar/seleccionar el valor concreto en la ventana *Propiedades* (p. ej. `<<GetAccessor>> nivel de acceso = public`).



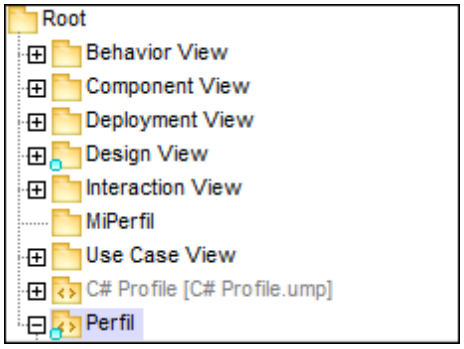
9.2.7.1 *Agregar estereotipos y definir valores etiquetados*

Para los ejemplos que aparecen a continuación utilizamos el archivo `Bank_MultiLanguage.ump` de la carpeta `...\UModelExamples`.

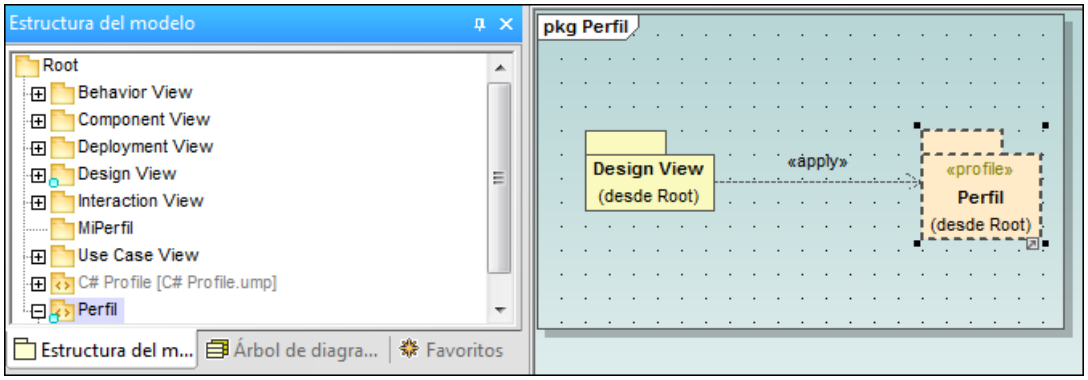
Para crear un diagrama de perfil y un estereotipo:

1. Cree un perfil nuevo desde la ventana *Estructura del modelo*. Por ejemplo: haga clic con el botón derecho en el paquete `Root` y elija **Elemento nuevo | Perfil**. Póngale el nombre `Perfil` al nuevo perfil.

 © 2018 Altova GmbH

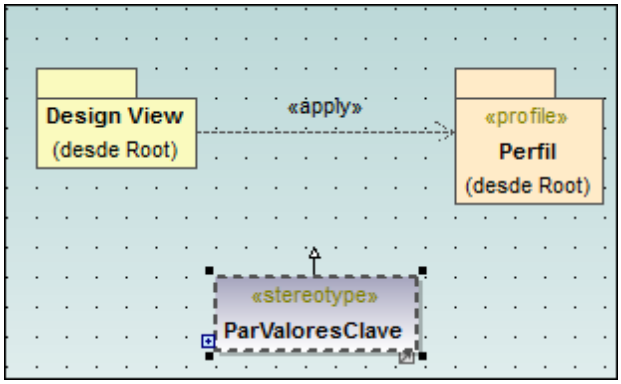


2. Ahora haga clic con el botón derecho en **`Perfil`** y elija **Diagrama nuevo | Diagrama de perfil**.
   Esto añade un diagrama de perfil al paquete seleccionado.
2. Arrastre el perfil recién creado **`Perfil`** desde la *Estructura del modelo* hasta el área de trabajo del diagrama de perfil.
3. Arrastre también el paquete **`DesignView`** hasta el área de trabajo del diagrama de perfil.
4. Ahora haga clic en el icono **AplicaciónDePerfil** de la barra de herramientas. Haga clic en el paquete **`DesignView`** y arrastre el puntero del ratón hasta el paquete **`Perfil`**.



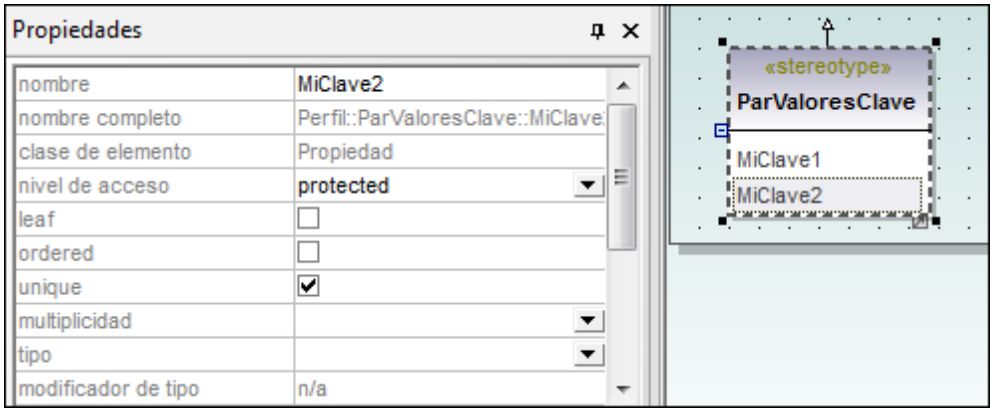
Esto permite usar en el paquete **`DesignView`** y en sus subpaquetes todos los estereotipos que estén definidos en el perfil **`Perfil`**.

5. Haga clic en el icono **Estereotipo** de la barra de herramientas e inserte un estereotipo de clase (p. ej. `ParValoresClave`).



6. Pulse **F7** para agregar un atributo al estereotipo (p. ej. `MiClave1`). Añada otro atributo

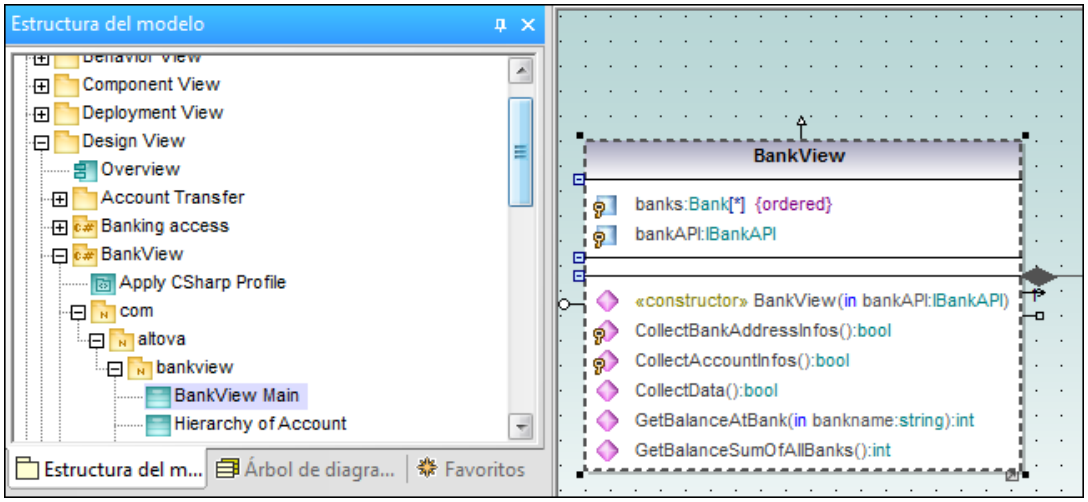
llamado `MiClave2`.



Por ahora hemos terminado de **definir** el estereotipo. En el siguiente paso podemos usar/asignar el estereotipo cuando añadamos atributos a una clase que forme parte del paquete **`BankView`**.

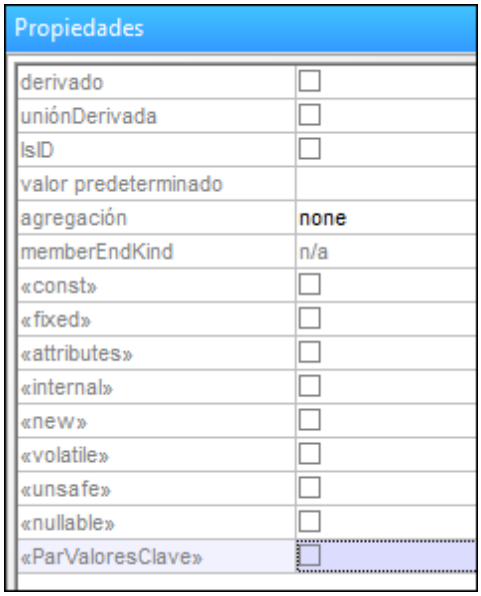
Para usar y asignar estereotipos:

1. En la *Estructura del modelo* haga doble clic en el icono del diagrama de clases **`BankView Main`**.
   Esto abre el diagrama de clases y muestra las asociaciones que existen entre las clases.

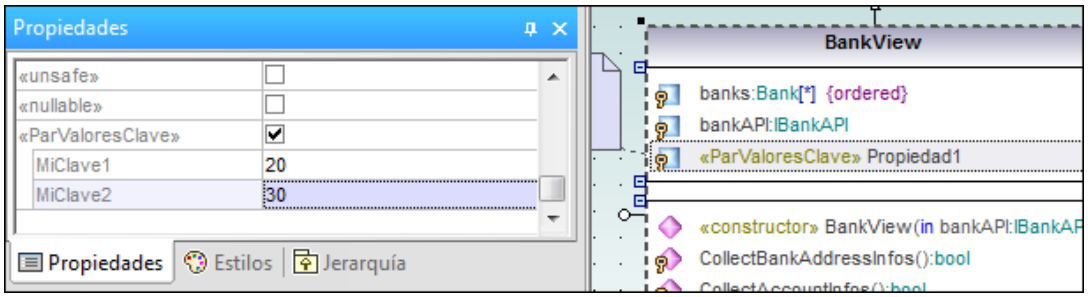


   Ahora vamos a añadir un atributo a la clase **`BankView`** y asignar el estereotipo definido previamente.
2. Haga clic en la clase **`BankView`** y pulse **F7** para añadir un atributo.
3. En la ventana *Propiedades* desplácese hasta el final de la lista. Observe que el estereotipo `ParValoresClave` está disponible en la ventana *Propiedades*.

 © 2018 Altova GmbH

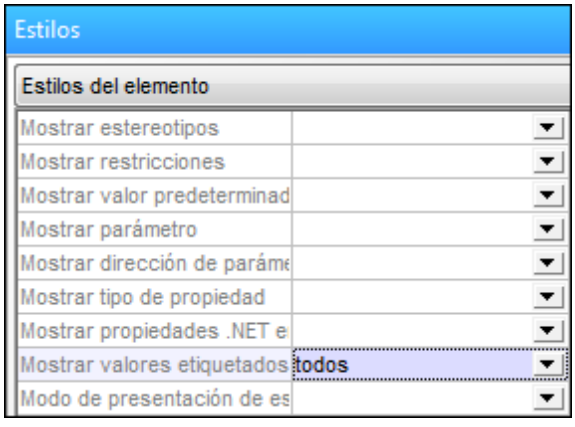


4. Marque la casilla `ParValoresClave` para aplicarlo. Los dos valores etiquetados `MiClave1` y `MiClave2` aparecen ahora bajo el estereotipo en la ventana *Propiedades*.
5. Haga doble clic en el campo de los valores etiquetados e inserte un valor para cada uno.

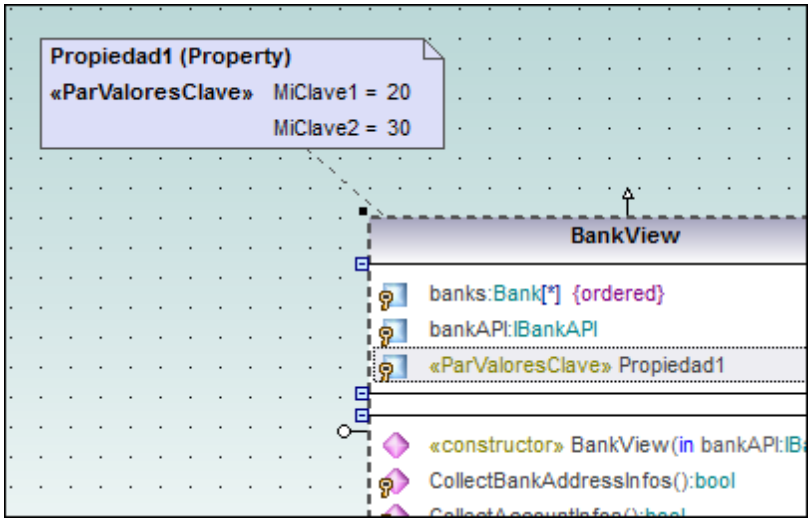


Para mostrar valores etiquetados en un diagrama:

1. Abra la ventana *Estilos* y desplácese hasta la entrada `Mostrar valores etiquetados`. Seleccione el valor `todos`.



Ahora el diagrama muestra los valores etiquetados en el elemento `Nota`. Para editar un valor del elemento `Nota`, haga doble clic.



Mostrar valores etiquetados | en compartimento

Este comando del menú contextual muestra los valores etiquetados en un compartimento separado de la clase.

Nota: cuando oculte atributos y operaciones con el comando **Mostrar u ocultar el contenido del nodo...** del menú contextual, los valores etiquetados también se muestran/ocultan junto con el elemento de modelado.

Los extremos de asociación pueden mostrar estereotipos si asigna el valor `verdadero` a la opción `Mostrar estereotipos de memberEnd` en la ventana *Estilos*.

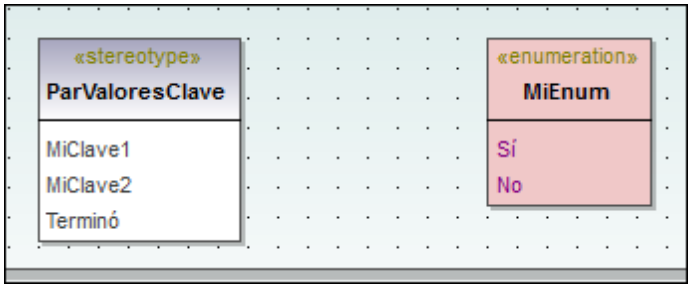
Para más información consulte el apartado Ver valores etiquetados.

9.2.7.2 Estereotipos y enumeraciones

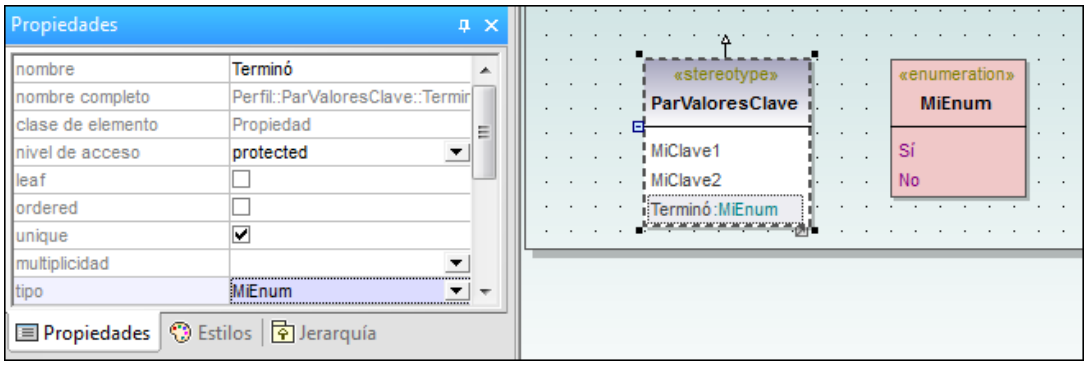
UModel ofrece un método muy eficaz para seleccionar valores enumerados de estereotipos.

Haga clic en la pestaña del diagrama que contiene la definición de estereotipo (es decir, el diagrama de perfil que añadimos en el primer paso):

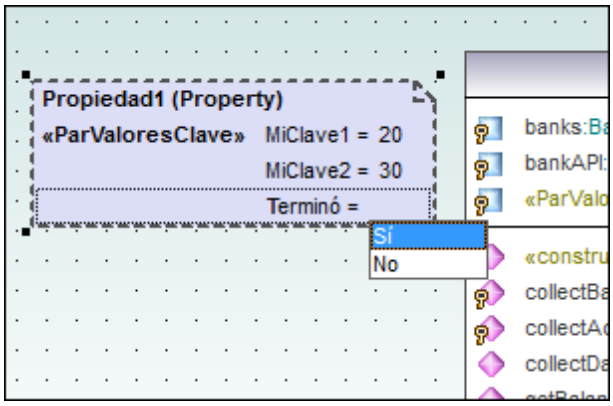
1. Haga clic en el icono **Enumeración** de la barra de herramientas para insertar una enumeración en el diagrama (que contiene el estereotipo previamente definido).
2. Pulse **Mayús+F7** para añadir los LiteralesDeEnumeración `Sí` y `No` a la enumeración. Si lo prefiere puede añadirlos desde el menú contextual.



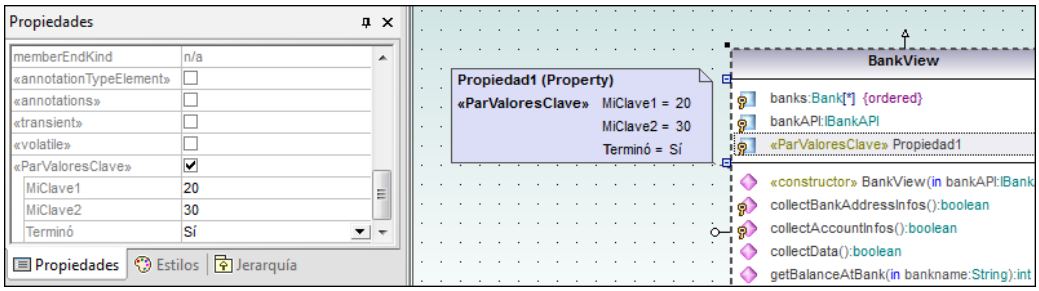
3. Haga clic en la clase estereotipo y pulse **F7** para añadir un atributo/una propiedad nueva (p. ej. `Terminó`).
4. Ahora en la ventana *Propiedades* seleccione el tipo `MiEnum`.



5. Ahora vuelva al diagrama de clases **`BankView Main`**.
6. Observe que la propiedad `Terminó` aparece ahora como valor etiquetado en la nota.

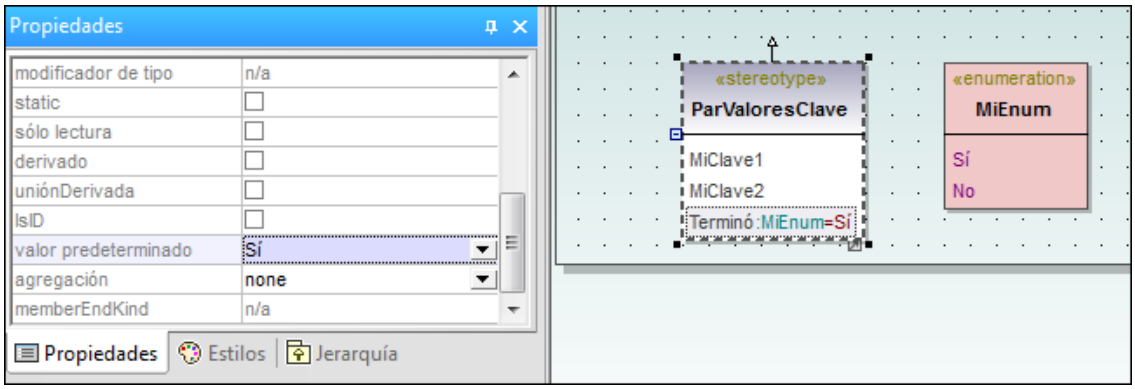


Haga doble clic en el valor etiquetado `Terminó` para ver los valores de enumeración predefinidos. Haga clic en una enumeración para seleccionarla.

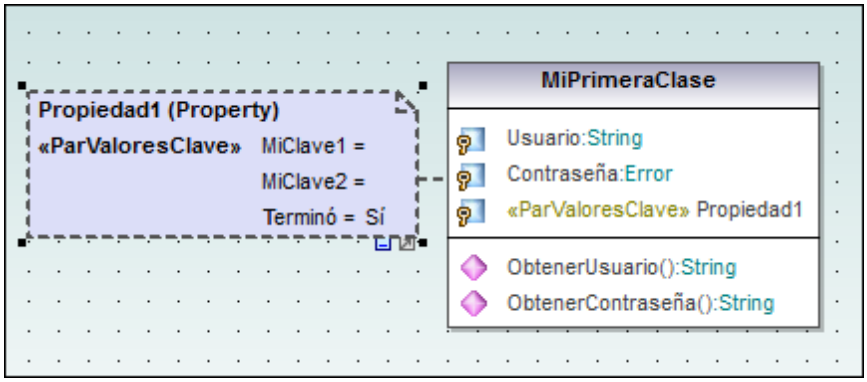


Valores de enumeración predeterminados

UModel también permite definir valores etiquetados **predeterminados**. Cuando añada un atributo al estereotipo, haga doble clic en el campo predeterminado e inserte una de las enumeraciones disponibles como valor predeterminado.



En el ejemplo de la imagen anterior se insertó el valor predeterminado `Sí`.
Cuando añada una propiedad a una clase y esté seleccionado el tipo `MiEnum`, el valor predeterminado se inserta automáticamente como valor etiquetado (es decir, `Terminó = Sí`).



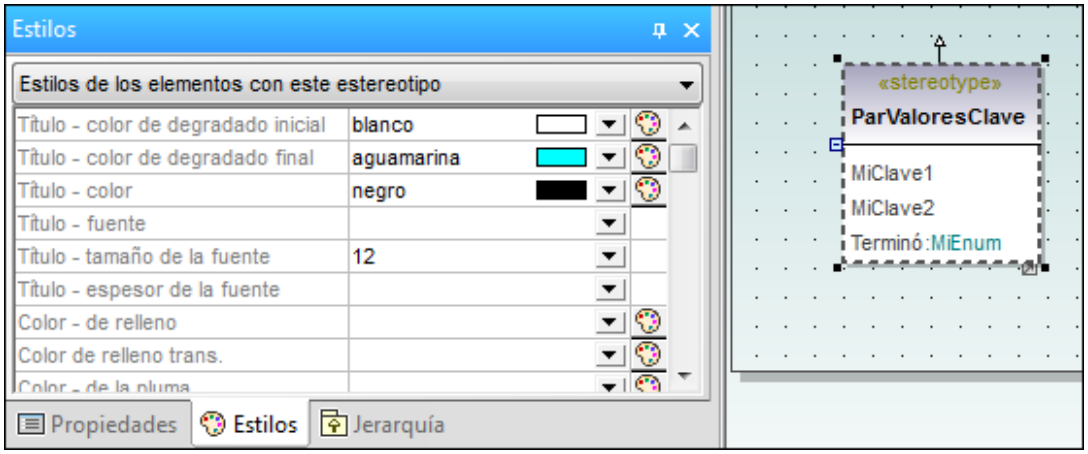
9.2.7.3 *Estilos de estereotipo definidos por el usuario*

En UModel puede crear estilos definidos por el usuario para cada estereotipo. Esto significa que puede tener fuentes y colores diferentes para cada estereotipo y aplicarlos a las clases que sean del tipo de estereotipo.

Para crear estilos de estereotipo definidos por el usuario:

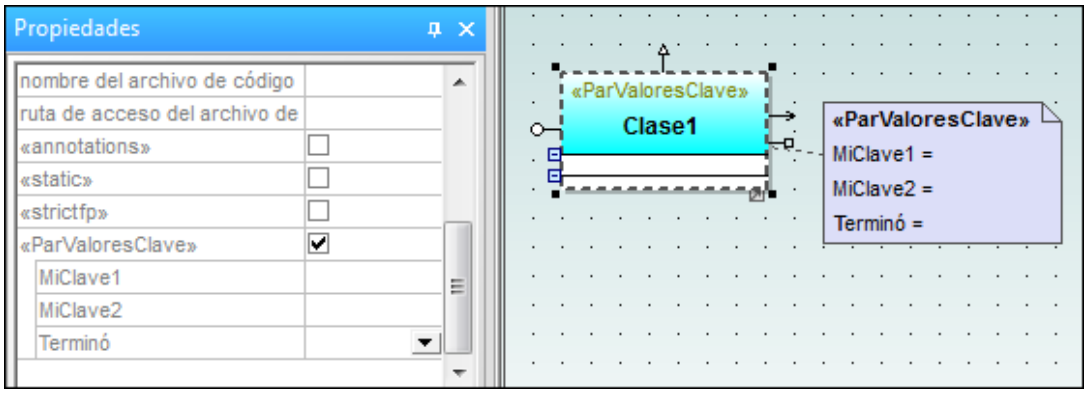
1. Haga clic en un estereotipo (p. ej. `ParValoresClave`).
2. Abra la ventana *Estilos* y, en el cuadro combinado de la parte superior, seleccione la opción **Estilos de elementos con este estereotipo**.
3. Defina los estilos de este estereotipo (p. ej. `Título - color de degradado final = aguamarina`).

© 2018 Altova GmbH



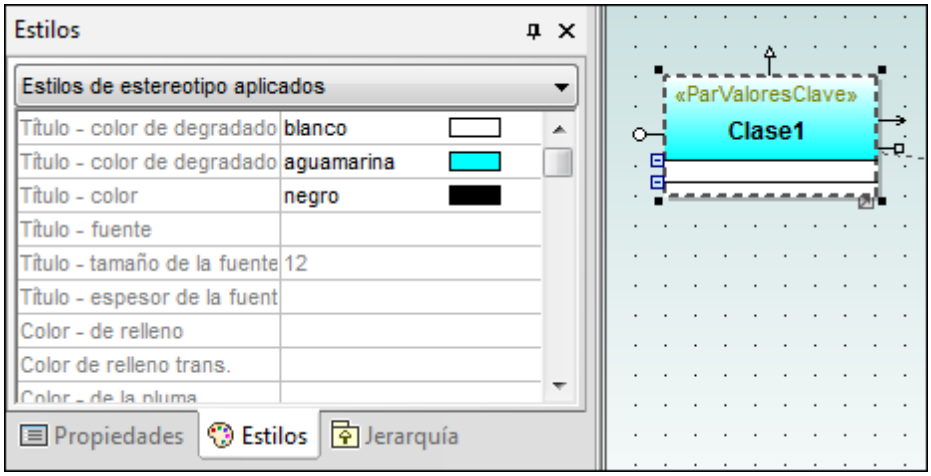
Al hacer clic en la clase del estereotipo, aparecen automáticamente los estilos definidos para el estereotipo en la ventana *Estilos*.

4. Ahora pase al diagrama de clases e inerte una clase nueva.
5. Haga clic en el título o encabezado de la clase y marque la casilla `ParValoresClave` en la ventana *Propiedades*.



La clase nueva tiene los estilos que se asignaron al estereotipo (el color de degradado del título es aguamarina, p. ej.). No olvide que los estilos del estereotipo **no se aplican** si el estereotipo se aplica a una propiedad u operación de la clase.

6. Haga clic en la nueva clase estereotipo del diagrama y después abra la ventana *Estilos*.
7. En el cuadro combinado situado en la parte superior seleccione la opción **Estilos de estereotipo aplicados**.



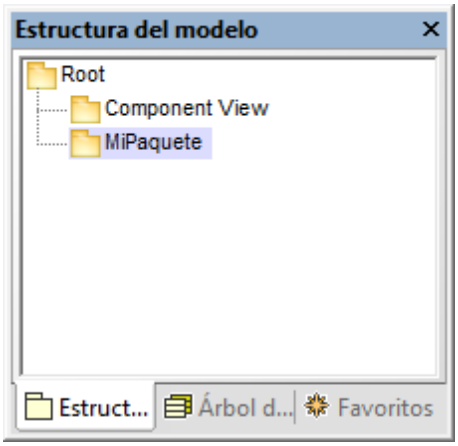
Aquí obtiene una vista previa de las opciones de estilo definidas para este estereotipo, pero no puede cambiar las opciones. Esto debe hacerse desde el diagrama de clases donde se definió el estereotipo.

9.2.7.4 Asignar iconos de estereotipo personalizados

Para terminar, en UModel también puede crear iconos personalizados para sus estereotipos.

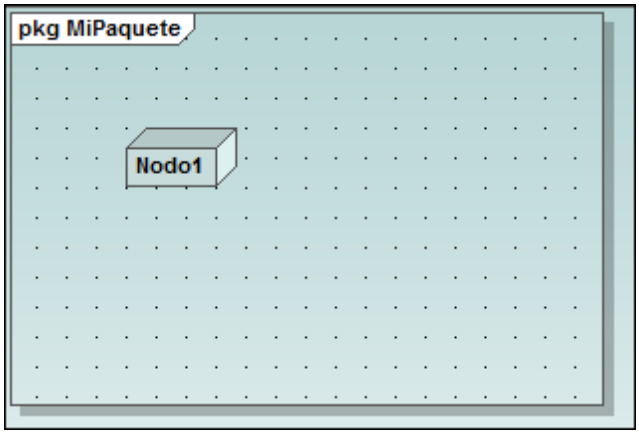
Para crear un icono de estereotipo personal:

1. Haga clic con el botón derecho en el paquete `Root` y elija **Elemento nuevo | Paquete** en el menú contextual.
2. A este paquete le ponemos el nombre `MiPaquete`.



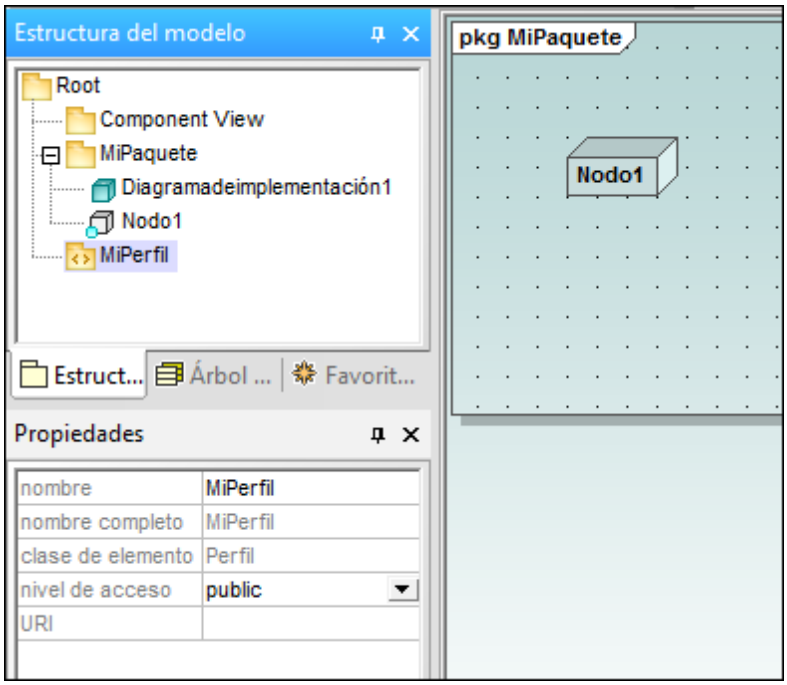
3. Haga clic con el botón derecho en **`MiPaquete`** y elija **Diagrama nuevo | Implementación** en el menú contextual.
   Esto crea un diagrama de implementación en **`MiPaquete`**.
4. Ahora haga clic con el botón derecho en **`MiPaquete`** y seleccione **Nuevo/a | Nodo**.

 © 2018 Altova GmbH



Esto añade el objeto `Nodo1` al diagrama.

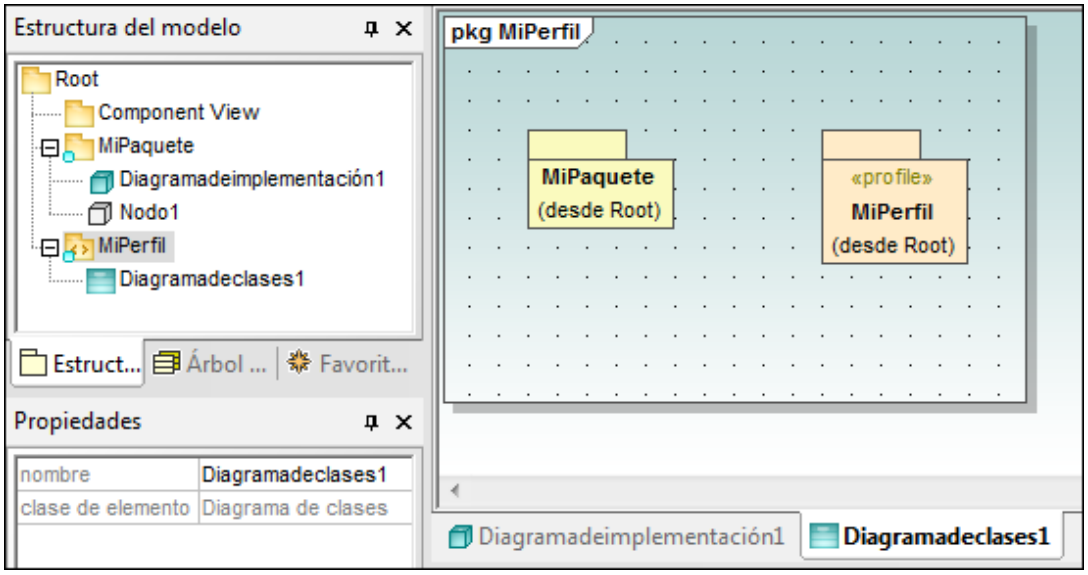
5. En la *Estructura del modelo* haga clic con el botón derecho en el paquete `Root` y después elija **Elemento nuevo | Perfil**.
6. Ponga el nombre `MiPerfil` a este nuevo perfil.



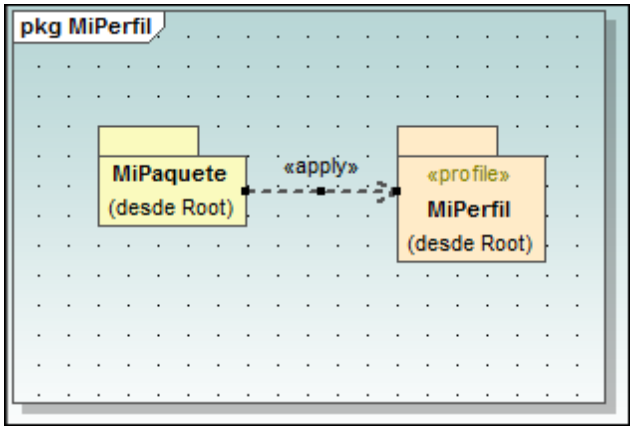
7. Haga clic con el botón derecho en `MiPerfil` y elija **Diagrama nuevo | Diagrama de clases**.

Para seleccionar el icono del estereotipo:

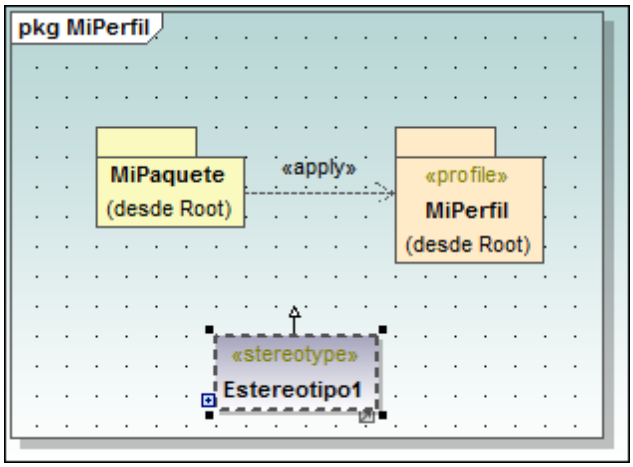
1. Arrastre los paquetes `MiPaquete` y `MiPerfil` desde la *Estructura del modelo* hasta el área de trabajo del diagrama de clases.



2. Haga clic en el icono **AplicaciónDePerfil** de la barra de herramientas. Haga clic en **`MiPaquete`** y arrastre el puntero hasta **`MiPerfil`**.

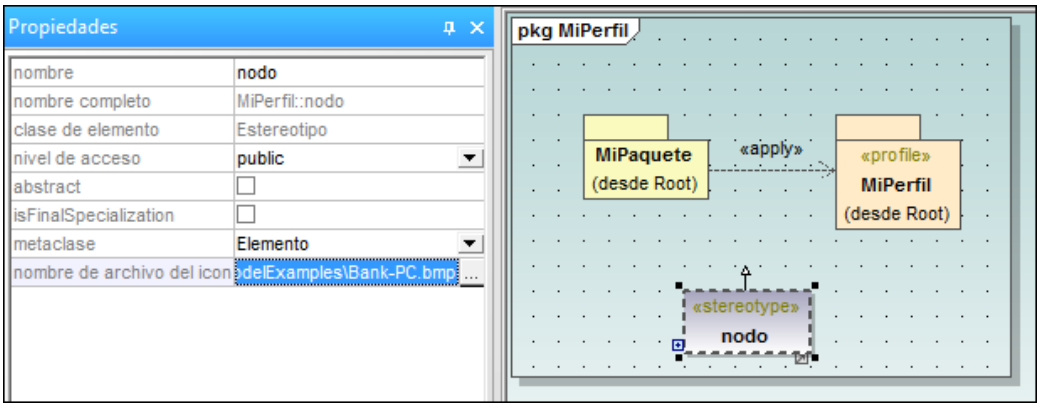


3. Haga clic en el icono **Estereotipo** de la barra de herramientas y haga clic en el paquete **`MiPerfil`** para insertar el estereotipo.

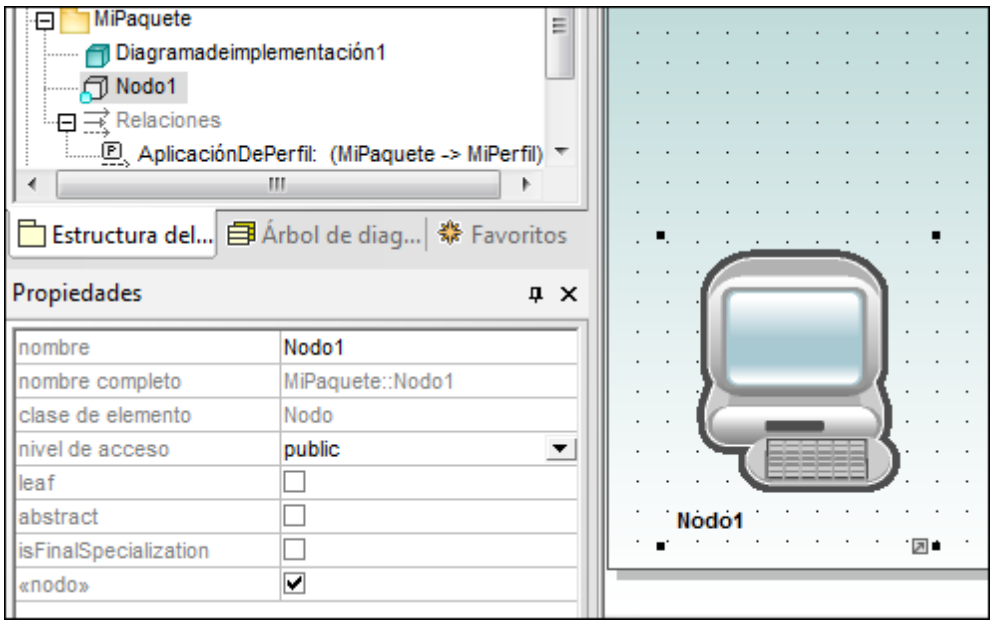


 © 2018 Altova GmbH

4. Haga clic en el estereotipo y ahora, en la ventana *Propiedades*, cambie el nombre del elemento de `Estereotipo1` a `nodo`.
5. Cambie también el valor de la propiedad `metaclase` de `Elemento` a `Nodo`.
6. En el campo `nombre del archivo del icono` inserte la ruta de acceso de la imagen que desea usar como icono del estereotipo o pulse el botón **Examinar** para seleccionar la ruta de acceso (p. ej. **..\Documentos\Altova\UModel2018\UModelExamples\Bank-PC.bmp**).



7. Vuelva al diagrama de implementación (haciendo clic en la pestaña *DiagramaDeImplementación1*).
8. Haga clic en el elemento **Nodo1** del diagrama.
9. En la ventana *Propiedades* marque la casilla `<<nodo>>`.



La representación del icono del estereotipo ahora es la imagen seleccionada.

Nota: con los iconos personalizados se utiliza el color RGB 82,82,82 como color de fondo (es decir, transparente).

9.3 Otros diagramas

UModel ofrece funciones para importar y generar esquemas XML del W3C y puede generar código a partir de ellos mediante ingeniería directa e inversa.

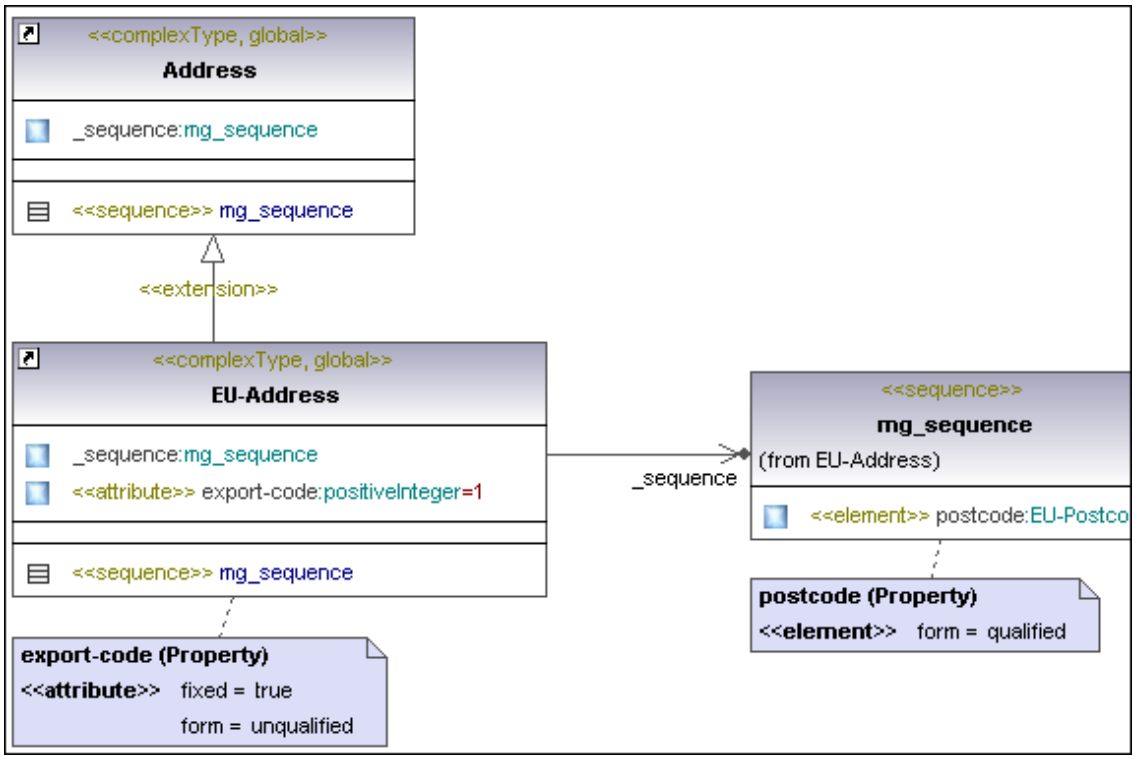
9.3.1 Diagramas de esquema XML

Sitio web de Altova: Esquemas XML en UML

Los diagramas de esquema XML dibujan los componentes de un esquema XML usando la notación UML. Los elementos globales (es decir, los elementos `element`, `simpleType`, `complexType`) se dibujan como clases o tipos de datos y los atributos se representan dentro del compartimento de atributos.

En el compartimento de operaciones no hay operaciones. El elemento de modelado *valor etiquetado* se utiliza para presentar los detalles del esquema.

Para ver las correspondencias entre elementos UML y atributos y elementos de XML Schema consulte el apartado Correspondencias con elementos de XML Schema.



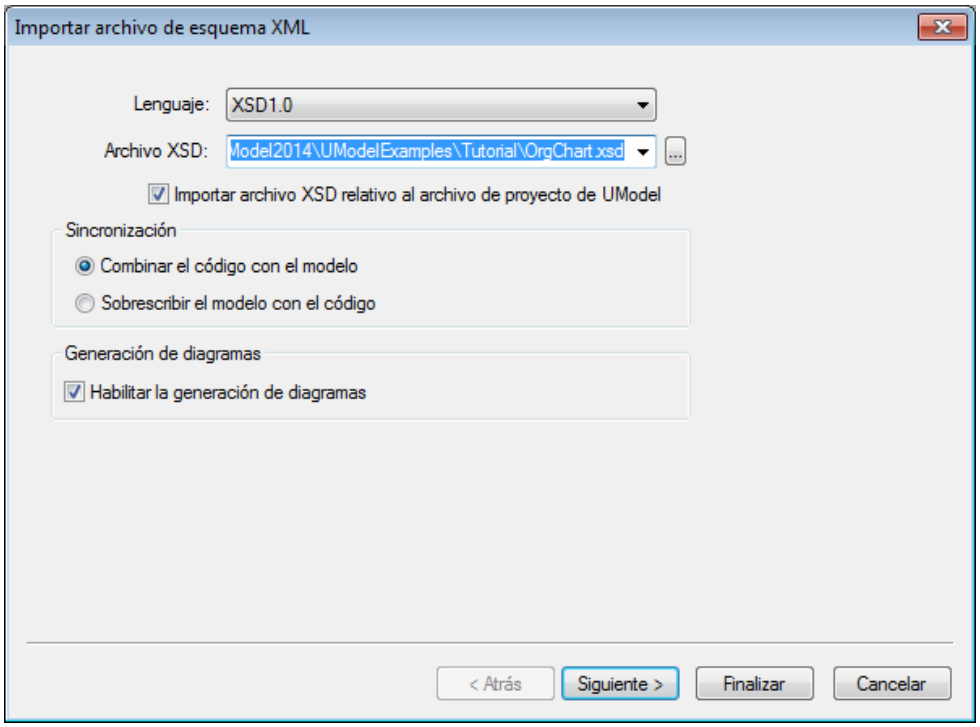
Nota: en UModel no se pueden importar esquemas XML no válidos. Los esquemas XML no se validan durante la importación ni durante su creación en UModel. Los esquemas XML tampoco se revisan durante el proceso de revisión sintáctica. Lo que sí se comprueba durante la importación es si el formato XML del esquema XML es correcto.

© 2018 Altova GmbH

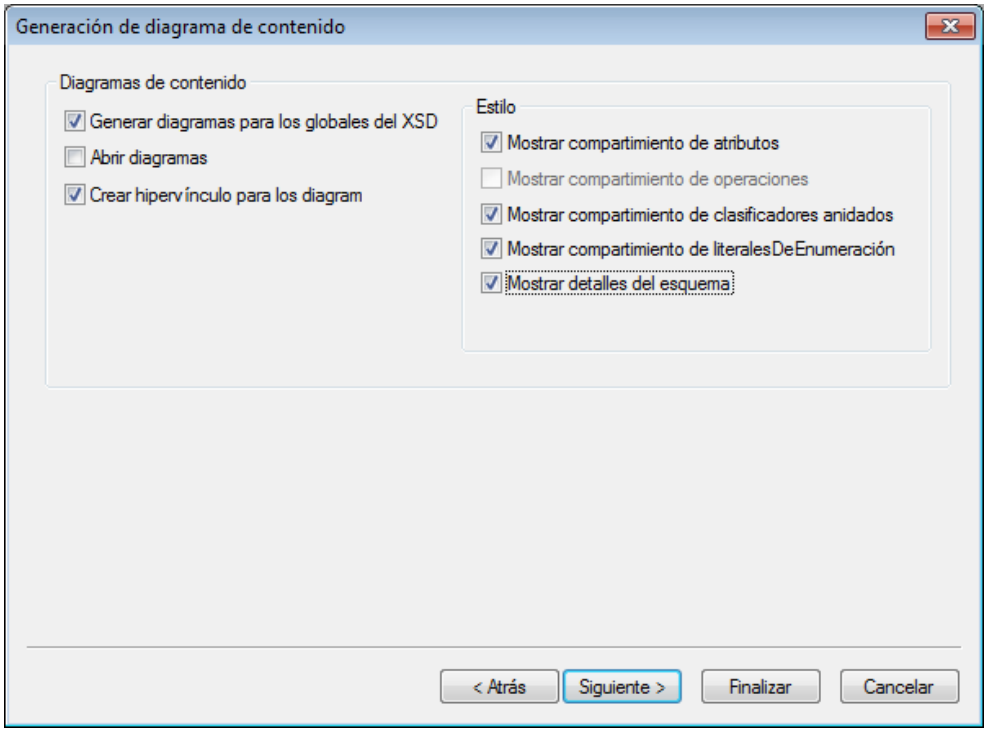
9.3.1.1 *Importar esquemas XML*

Para importar un solo esquema XML:

1. Seleccione el comando de menú **Proyecto | Importar archivo de esquema XML**.

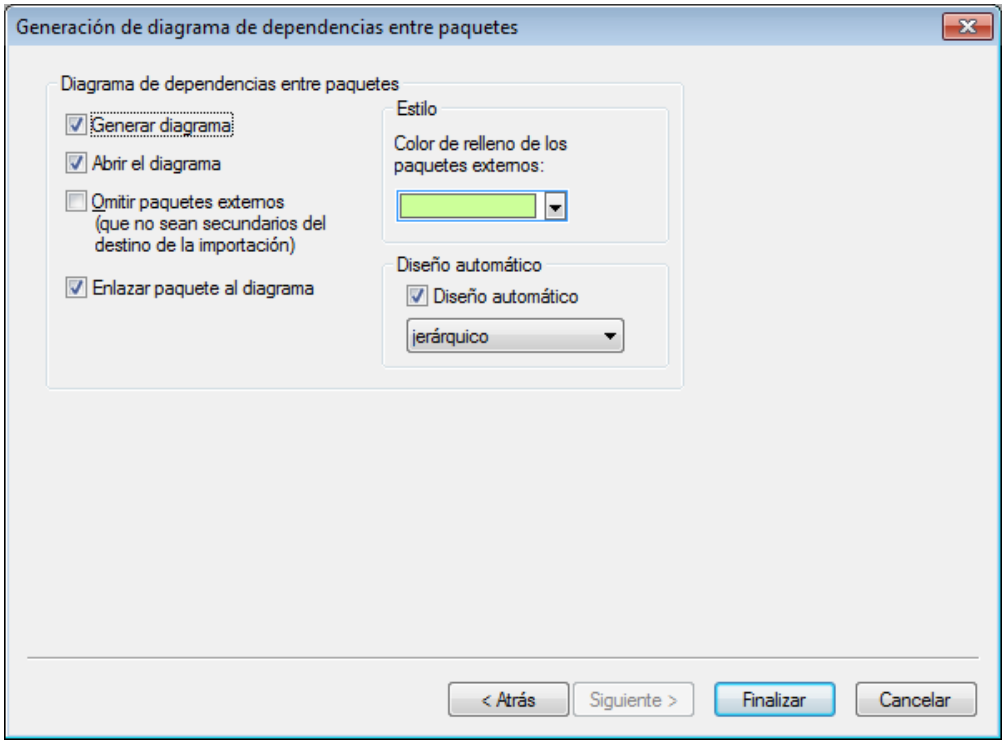


2. Asegúrese de que la casilla *Habilitar la generación de diagramas* esté activa. Haga clic en **Siguiente** para continuar.

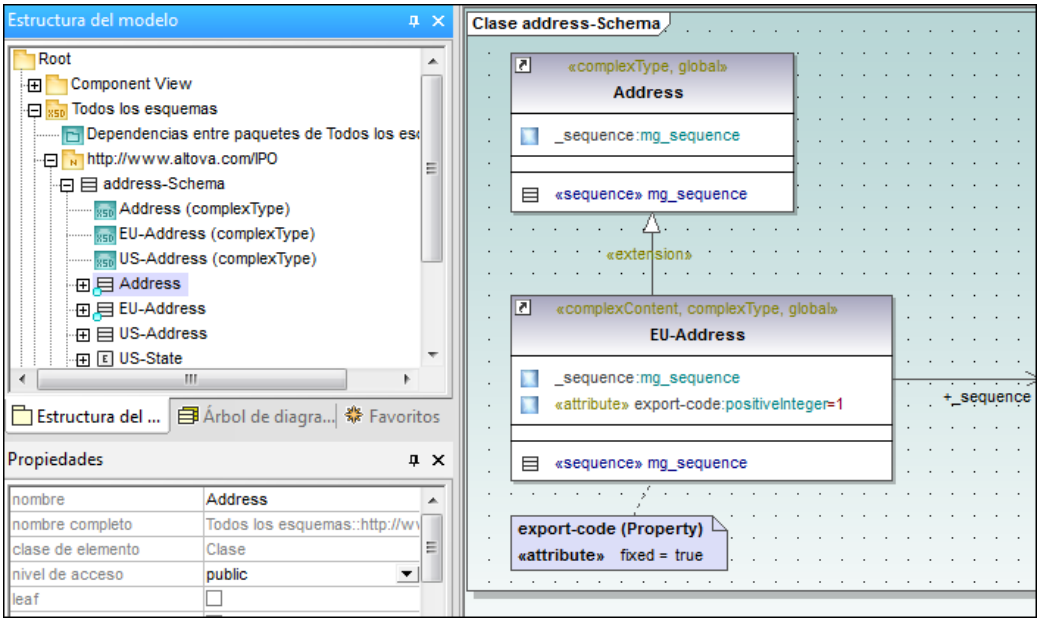


3. En la siguiente pantalla, elija las opciones en el grupo *Diagramas de contenido*. La primera opción crea un diagrama distinto por cada elemento global del esquema.
4. En el grupo de opciones *Estilo* seleccione qué compartimentos deben aparecer en los diagramas de clases. Si marca la casilla *Mostrar detalles del esquema como valores etiquetados*, los detalles del esquema aparecerán en el elemento de modelado `ValorEtiquetado`.
5. Haga clic en **Siguiente** para configurar la generación de diagramas de dependencias entre paquetes.

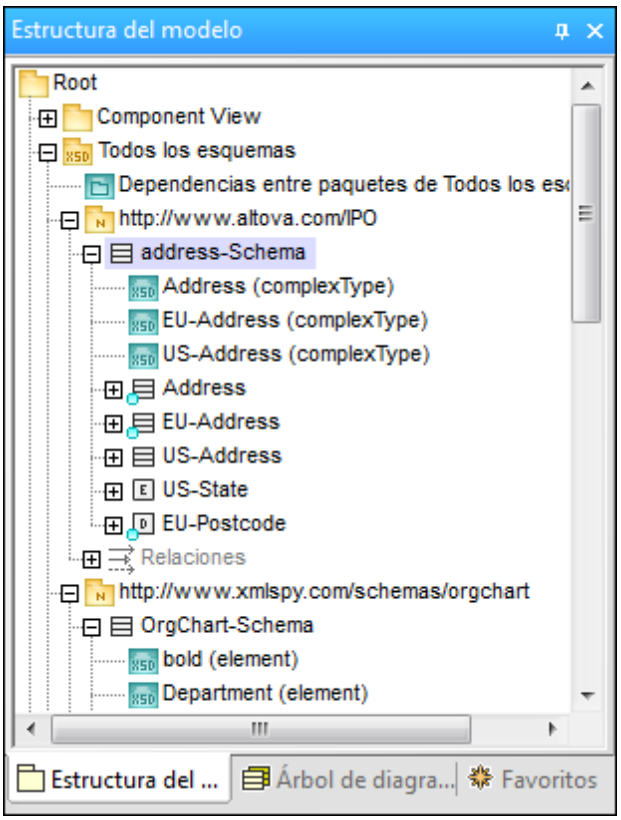
© 2018 Altova GmbH



6. Haga clic en **Finalizar** para empezar a importar el esquema XML.
   El esquema se importa en UModel y los diagramas generados se abren en el área de trabajo. Por ejemplo, en la imagen siguiente puede ver el contenido del diagrama **`EU-Address`** (`complexType`).



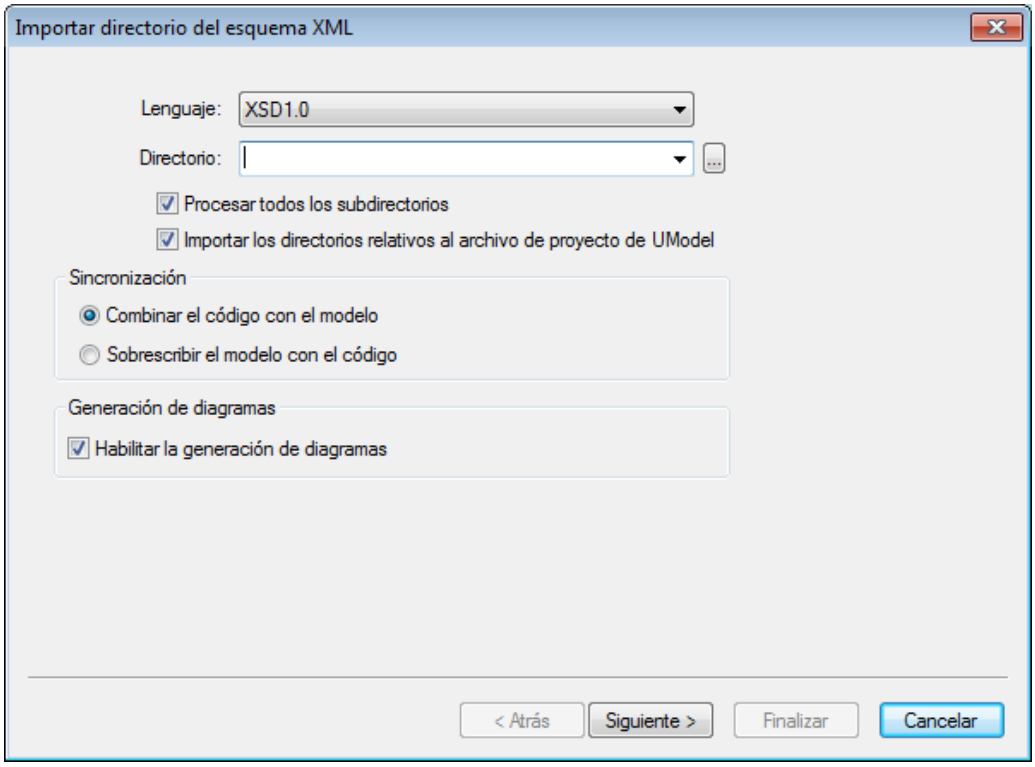
Nota: También se creó un paquete nuevo llamado **`Todos los esquemas`**, que se configuró como raíz del espacio de nombres XSD. Todos los elementos XSD globales generan un diagrama de esquema XML y los diagramas están bajo los paquetes de los correspondientes espacios de nombres.



Para importar varios esquemas XML:

1. Seleccione el comando de menú **Proyecto | Importar directorio del esquema XML**.

© 2018 Altova GmbH

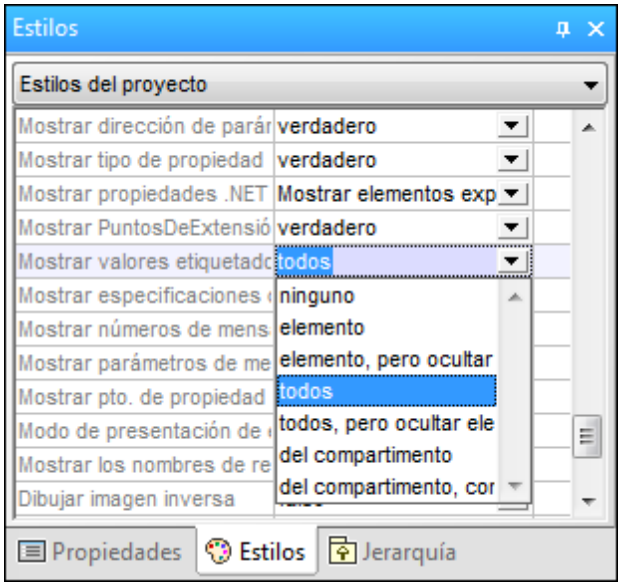


2. Marque la casilla *Procesar todos los subdirectorios* si quiere importar los esquemas de todos los subdirectorios.
   El resto del proceso de importación sigue el mismo orden que cuando se importa un solo esquema XML (ver más arriba).

   Nota: si un esquema XML incluye o importa otros esquemas, estos esquemas se importan automáticamente también.

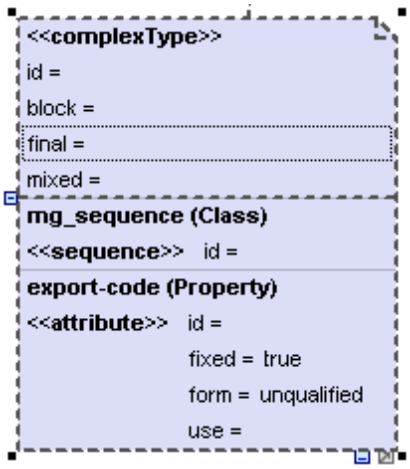
Ver valores etiquetados: detalles del esquema

En UModel tiene la opción de configurar los detalles del esquema que se presentan como valores etiquetados en el elemento `ValorEtiquetado`. Esto se hace en el campo `Mostrar valores etiquetados` de la ventana *Estilos* o haciendo clic en el icono **Activar o desactivar el modo compacto** (situado en la esquina inferior derecha del elemento Valor etiquetado). Con este icono puede alternar entre los diferentes estados del valor etiquetado (p. ej. `todos`, `todos, pero ocultar elementos ocultos`, etc.)



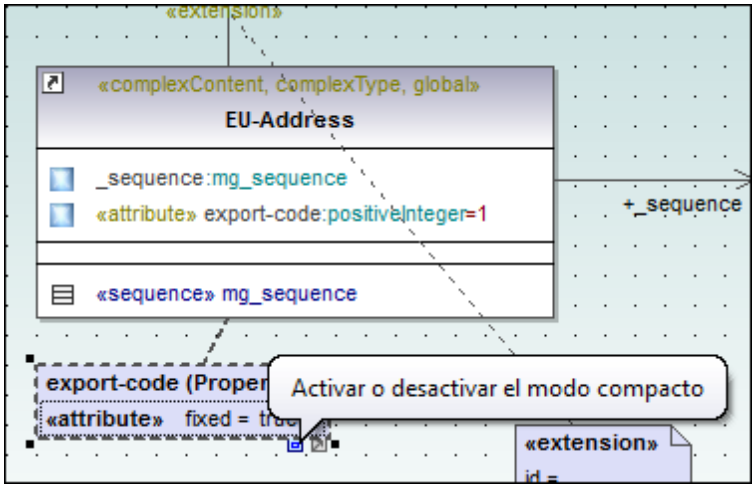
Nota: el estilo seleccionado para los valores etiquetados en la ventana *Estilos* se utiliza para todo el proyecto. En cambio, si hace clic con el botón derecho en una clase y elige el comando **Valores etiquetados | Todos**, por ejemplo, esta opción solamente se aplica a la clase seleccionada.

Mostrar valores etiquetados | todos
Esta opción muestra los valores etiquetados de la clase y de atributos propios y operaciones propias, etc.

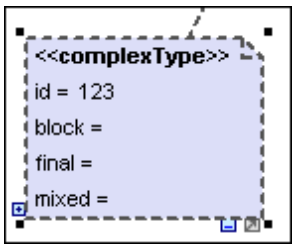


Mostrar valores etiquetados | Todos, pero ocultar elementos vacíos
Esta opción solamente muestra los valores etiquetados que tienen asignado un valor (p. ej. `fixed=true`).

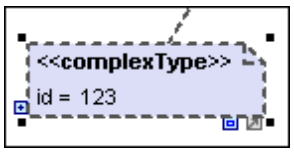
 © 2018 Altova GmbH



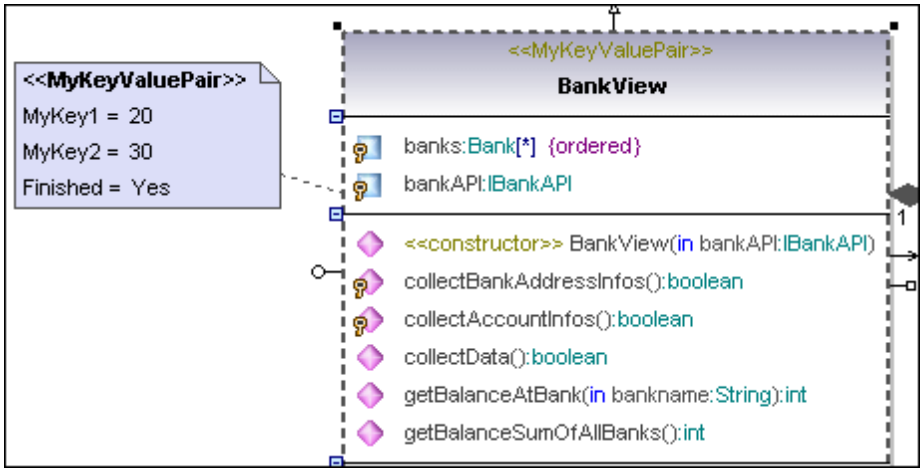
Mostrar valores etiquetados | Todos
Esta opción muestra los valores etiquetados de la clase pero **no** los de los atributos y operaciones propios, etc.



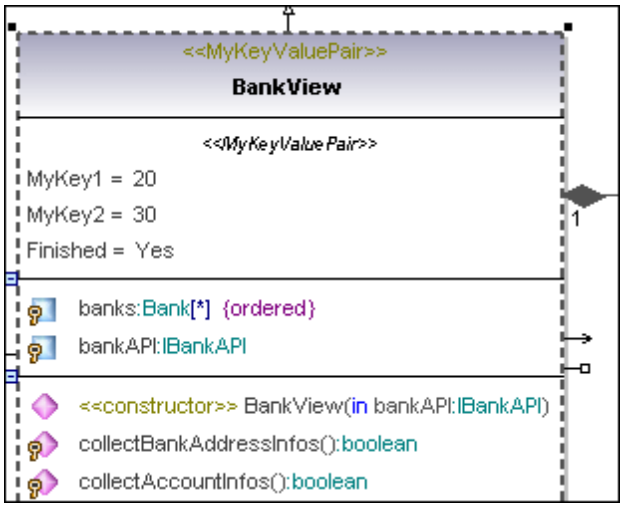
Mostrar valores etiquetados | Elemento, pero ocultar elementos vacíos
Esta opción solamente muestra los valores de elemento etiquetados de una clase, sin los atributos propios, que tengan un valor (p. ej. `id=123`).



Mostrar valores etiquetados | en compartimento



Esta opción muestra los valores etiquetados en un compartimento distinto de la clase.



Mostrar valores etiquetados | En compartimento, pero ocultar elementos vacíos
Esta opción solamente muestra los valores de elemento etiquetados de una clase, sin los atributos propios, que tengan un valor.

Anotación de XML Schema:
Cuando importe esquemas XML, tenga en cuenta que en la ventana *Documentación* solamente aparece la primera anotación de un elemento `complexType` o `simpleType`.

9.3.1.2 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 esquema XML.

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 un diagrama de esquema XML:

La mayoría de los elementos de un diagrama de esquema XML se pueden insertar en otro.

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 esquema XML.

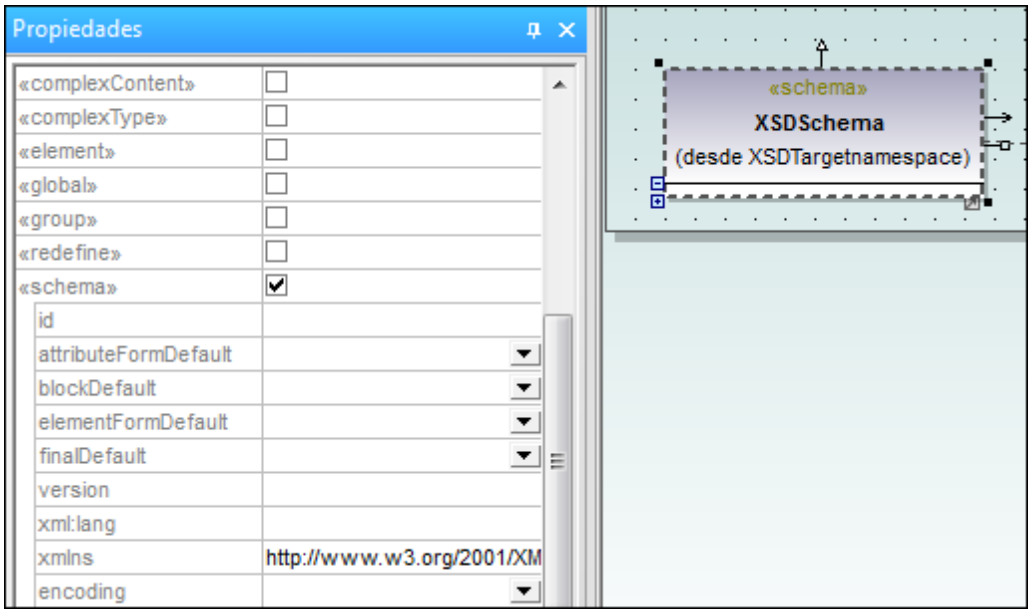
Nota: también puede usar los comandos **Copiar** y **Pegar sólo en el diagrama** para insertar elementos.

targetNamespace XSD

Inserta/define el espacio de nombres de destino para el esquema. Este espacio de nombres de destino debe pertenecer al paquete **`Raíz de espacio de nombres XSD`**.

schema XSD

Inserta/define un esquema XML. El esquema XML debe pertenecer a un paquete del espacio de nombres de destino XSD.



Element (global) XSD

Inserta un elemento global en el diagrama. Tenga en cuenta que también se genera automáticamente una propiedad en el compartimento de atributos.

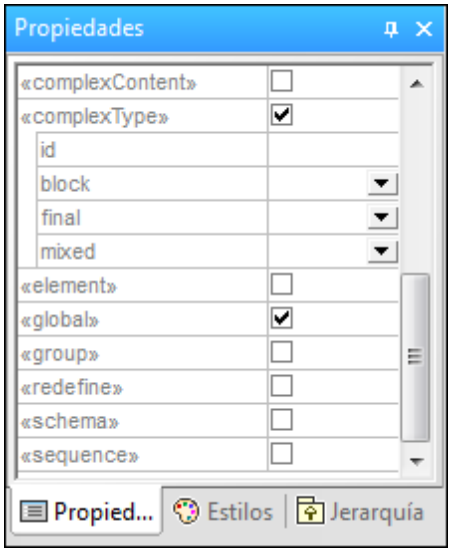
Para definir el tipo de datos de la propiedad:

1. Haga doble clic en la propiedad y ponga el cursor al final de la línea.
2. Inserte dos puntos (:) y seleccione el tipo de datos en el cuadro de diálogo emergente (p. ej. `string`).

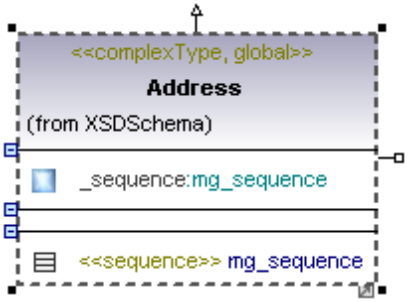
Para crear un modelo de contenido formado por un complexType con elementos obligatorios:

Esto supone insertar un elemento complexType, un elemento `sequence` y tres elementos.

1. Haga clic en el icono **ComplexType XSD** y después haga clic en el diagrama para insertarlo.
2. Haga doble clic en el nombre y ponga el nombre nuevo **`Address`**.

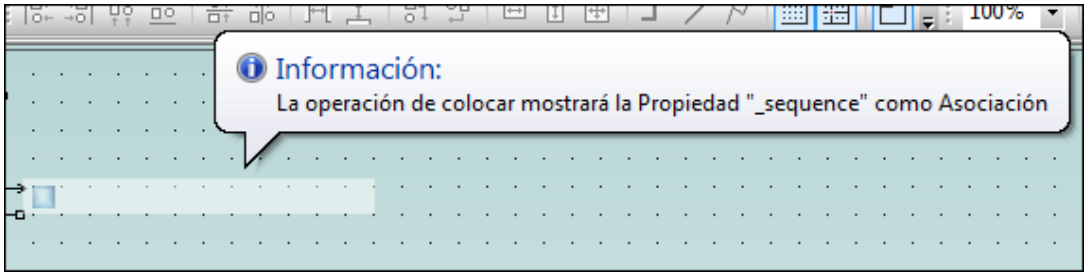


3. Haga clic con el botón derecho en **`Address`** y elija **Nuevo/a | XSD Sequence**.

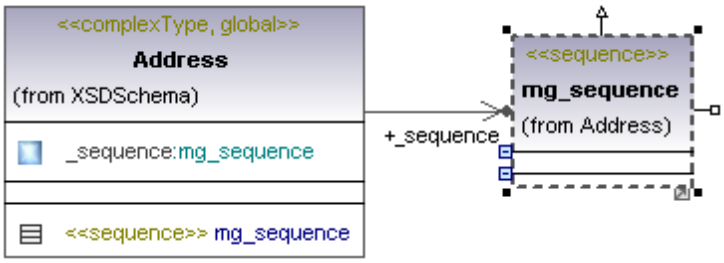


4. Haga clic en el atributo `_sequence:mg_sequence` del compartimento de atributos y arrástrelo fuera del compartimento.

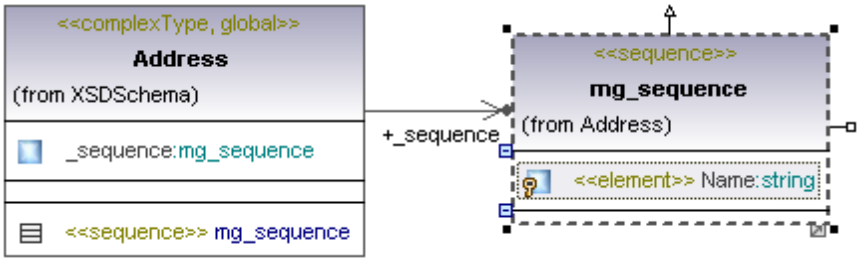
© 2018 Altova GmbH



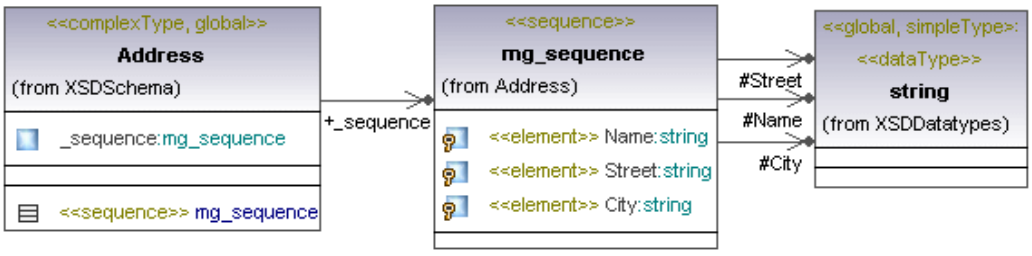
Esto crea una clase `sequence` o `compositor` al soltar el atributo.



5. Haga clic con el botón derecho en la clase **`sequence`** y elija **Nuevo/a | XSD Element (local)** en el menú contextual.
   Esto añade una propiedad nueva.
6. Haga doble clic en la propiedad, inserte el nombre del elemento (p. ej. **`Name`**), añada dos puntos (:) e inserte el tipo de datos `string`.



7. Haga lo mismo con otros dos elementos y llámelos `Street` y `City`.
8. Haga clic en la propiedad `Name` y arrástrela hasta el diagrama.

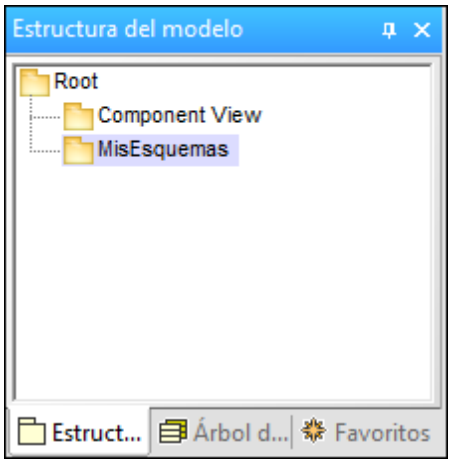


9.3.1.3 Crear y generar un esquema XML

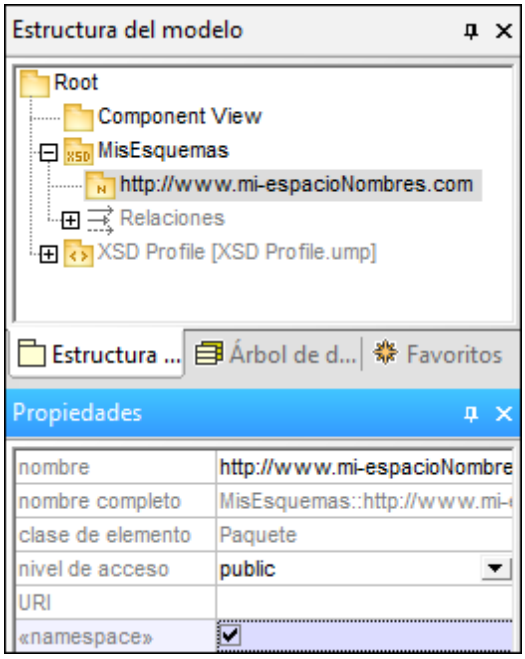
Por lo general, el esquema se importa y se edita en UModel y después se generan los cambios. Sin embargo, en UModel también puede generar un esquema desde cero. A continuación describimos el proceso de forma muy general.

Para crear un esquema nuevo en UModel:

1. En la *Estructura del modelo* cree un paquete nuevo llamado **`MisEsquemas`**.



2. Haga clic con el botón derecho en el paquete nuevo y seleccione la opción **Ingeniería de código | Establecer como raíz de espacio de nombres XSD**. Aparece un aviso preguntando si desea asignar el perfil XSD si esta es la primera raíz de espacio de nombres XSD del proyecto.
3. Haga clic en **Aceptar** para asignar el perfil.
4. Haga clic con el botón derecho en el paquete nuevo y seleccione la opción **Elemento nuevo | Paquete**.
5. En la ventana *Propiedades* haga doble clic en el campo `nombre` del paquete e inserte el espacio de nombres que desea usar (p. ej. `http://www.mi-espacioNombres.com`).
6. Marque la casilla `<<namespace>>` para definir este paquete como espacio de nombres de destino.

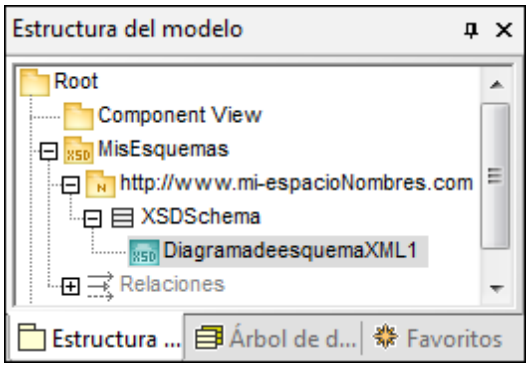


7. Haga clic con el botón derecho en el paquete de espacio de nombres y seleccione la

 © 2018 Altova GmbH

opción **Diagrama nuevo | Diagrama de esquema XML**.
Aparece un aviso preguntando si desea añadir el diagrama en un esquema XML nuevo.

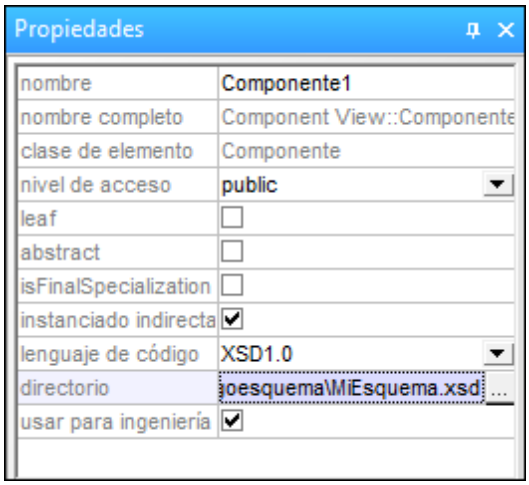
8. Haga clic en **Sí**.



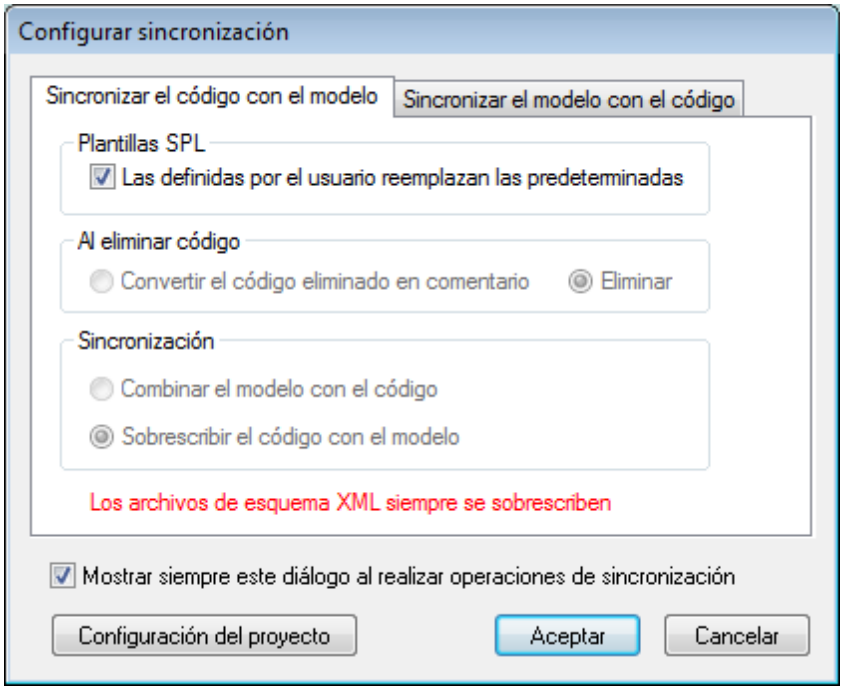
Ahora puede crear su esquema con ayuda de los iconos de la barra de herramientas Diagrama de esquema XML.

Para generar el esquema XML:

1. Arrastre el elemento `XSDSchema` hasta un componente para crear una `RealizaciónDeComponente`.
2. Defina el lenguaje de código XSD 1.0 para el componente y escriba la ruta de acceso donde se debe guardar el esquema generado (p. ej. `C:\códigoesquema\MiEsquema.xsd`).



3. Ahora seleccione el comando de menú **Proyecto | Sobrescribir el código de programa con el proyecto de UModel**. En el cuadro de diálogo haga clic en **Aceptar** para generar el esquema.



© 2018 Altova GmbH

Altova UModel 2018

XMI: intercambio de metadatos XML

10 XMI: intercambio de metadatos XML

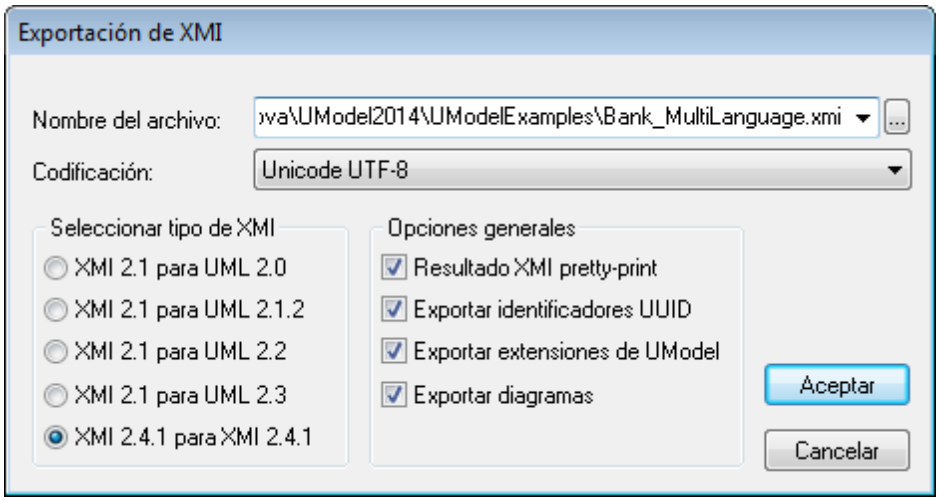
Sitio web de Altova: Intercambio de modelos UML mediante archivos XMI

UModel ofrece una función para exportar e importar datos XMI 2.4 para UML 2.0 / 2.1 / 2.1.1 y 2.1.2, 2.2, 2.3, 2.4. No utilice la función de exportación XMI para archivar proyectos de UModel. Archive los archivos de proyecto `*.ump`.

El comando **Archivo | Exportar a un archivo XMI** genera un archivo XMI a partir del proyecto de UModel. Por su parte, el comando **Archivo | Importar desde un archivo XMI** importa datos desde un archivo XMI generado con anterioridad.

El cuadro de diálogo "Exportación de XMI" sirve para seleccionar el formato XMI en el que se exporta el archivo (p. ej. XMI para UML 2.0/2.1.1). Durante el proceso de exportación también se exportan los archivos incluidos, incluso los que están *incluidos mediante referencia*.

Importante: si tiene pensado reimportar el código XMI al proyecto de UModel, asegúrese de marcar la casilla *Exportar extensiones de UModel*.



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

 © 2018 Altova GmbH

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 marcar también la casilla *Exportar extensiones de UModel.*

Altova UModel 2018

Trabajo en equipo con UModel

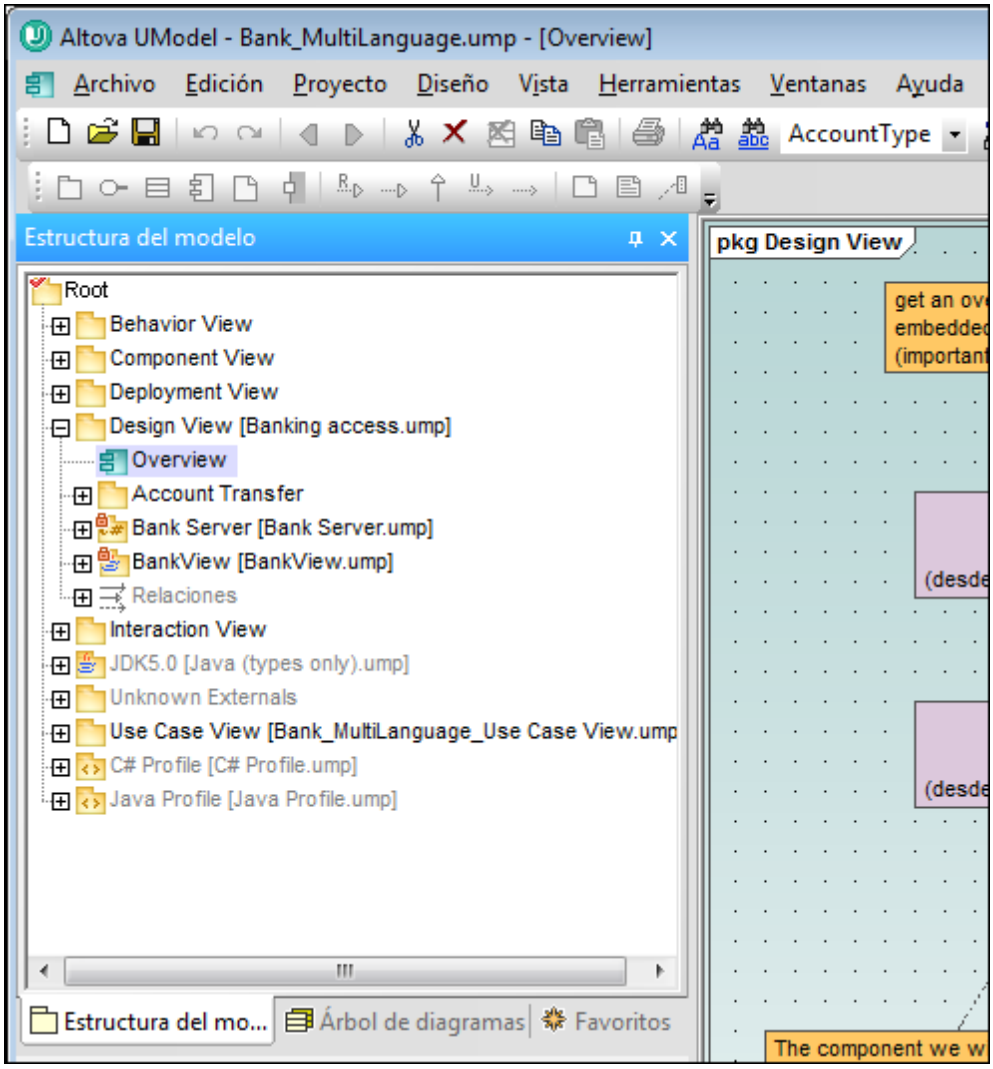
11 Trabajo en equipo con UModel

Los proyectos de UModel se pueden dividir en varios subproyectos para que varios programadores puedan editar partes diferentes del proyecto por separado. Los subproyectos pueden añadirse al sistema de control de versiones utilizado por el equipo.

El proyecto de nivel superior (es decir, el proyecto en el que se pueden incluir los subproyectos) recibe el nombre de *proyecto principal*. Los subproyectos se crean al nivel de los paquetes como archivos de proyecto UModel independientes y tienen la extensión de archivo `*.ump`.

Puede crear e incluir subproyectos de dos maneras distintas:

- un subproyecto puede ser **editable** dentro del proyecto **principal** y a nivel de subproyecto.
- un subproyecto puede ser de **sólo lectura** dentro del proyecto **principal** y editable a nivel de subproyecto.



Los subproyectos pueden estar organizados de forma jerárquica, con una estructura plana o de ambas formas. Esto permite, en teoría, dividir cada paquete de un proyecto principal en varios

 © 2018 Altova GmbH

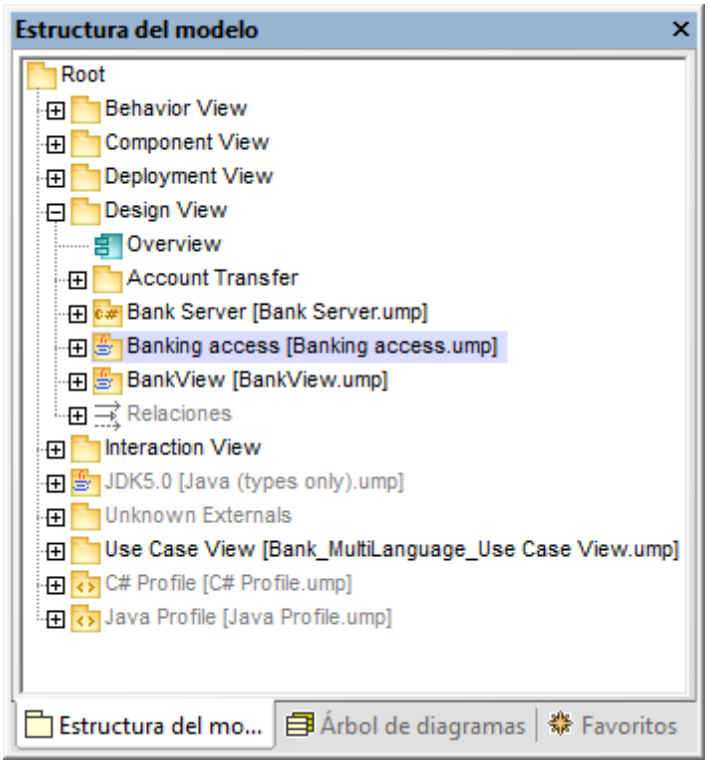
subproyectos editables o de sólo lectura.

Durante el proceso de ingeniería de código todos los componentes subordinados de un subproyecto se tienen en cuenta. Es decir, para la ingeniería de código no hay diferencia alguna entre un archivo de proyecto sencillo y uno formado por varios subproyectos editables.

Esto también afecta a los diagramas UML, que se pueden editar a nivel de proyecto principal o de subproyecto.

11.1 Crear y editar subproyectos

A continuación usamos un ejemplo para explicar cómo se divide un proyecto en varios subproyectos. El ejemplo se sirve del archivo de muestra `Bank_MultiLanguage.ump`.

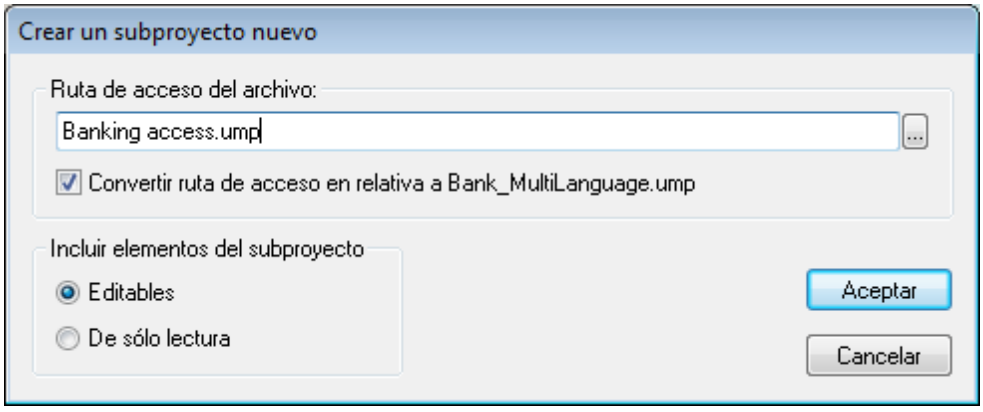


Nota: en UModel puede compartir paquetes y sus diagramas UML con varios proyectos. Los paquetes se pueden incluir en otros proyectos por medio de referencias o como copias. Para más información consulte el apartado Compartir paquetes y diagramas.

Para crear un archivo de subproyecto:

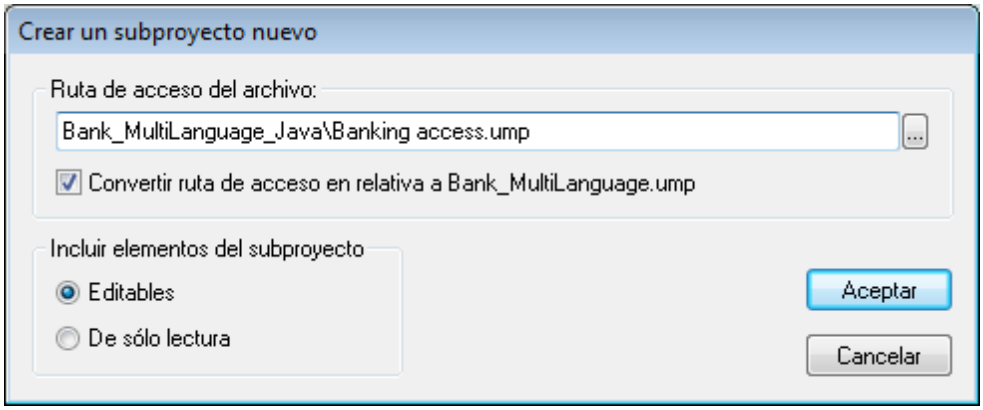
Puede crear subproyectos a partir de un proyecto principal o de otro subproyecto.

1. Haga clic con el botón derecho en un paquete (p. ej. **`Banking access`**) y seleccione **Subproyecto | Crear subproyecto nuevo**.

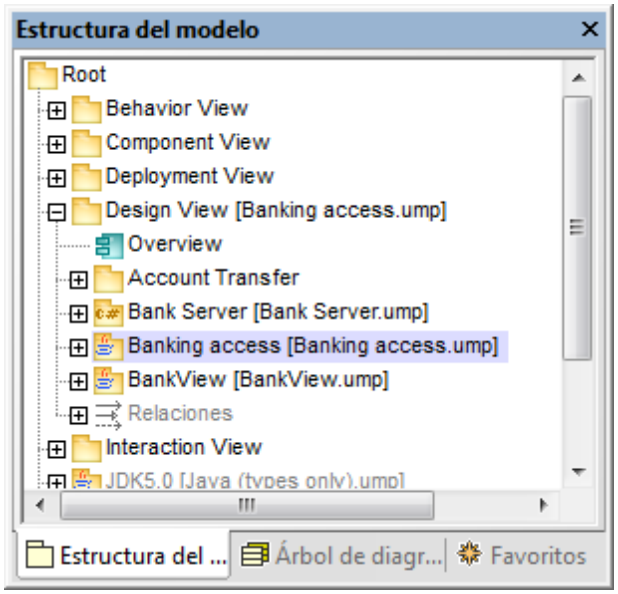


 © 2018 Altova GmbH

2. Haga clic en el botón **Examinar** y seleccione el subdirectorio **\Bank_MultiLanguage_Java**.

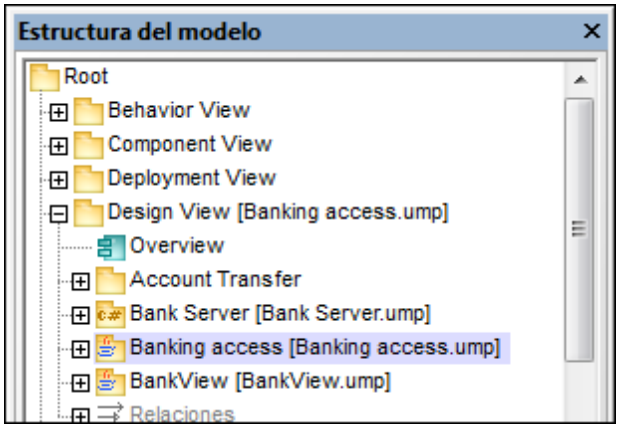


3. Seleccione el botón de opción *Editables* para poder editar el subproyecto desde el proyecto principal. (Si selecciona la opción *De sólo lectura*, el subproyecto no se podrá editar en el proyecto principal). Ahora haga clic en **Aceptar**.



El nombre del subproyecto aparece entre corchetes junto al nombre de paquete y el archivo `Banking access.ump` se guarda en la carpeta `UModelExamples \Bank_MultiLanguage_Java`.

Utilice el mismo método para crear un subproyecto de la carpeta **BankView**. El archivo `BankView.ump` se guarda en la carpeta `...\UModelExamples \Bank_MultiLanguage_Java\`.



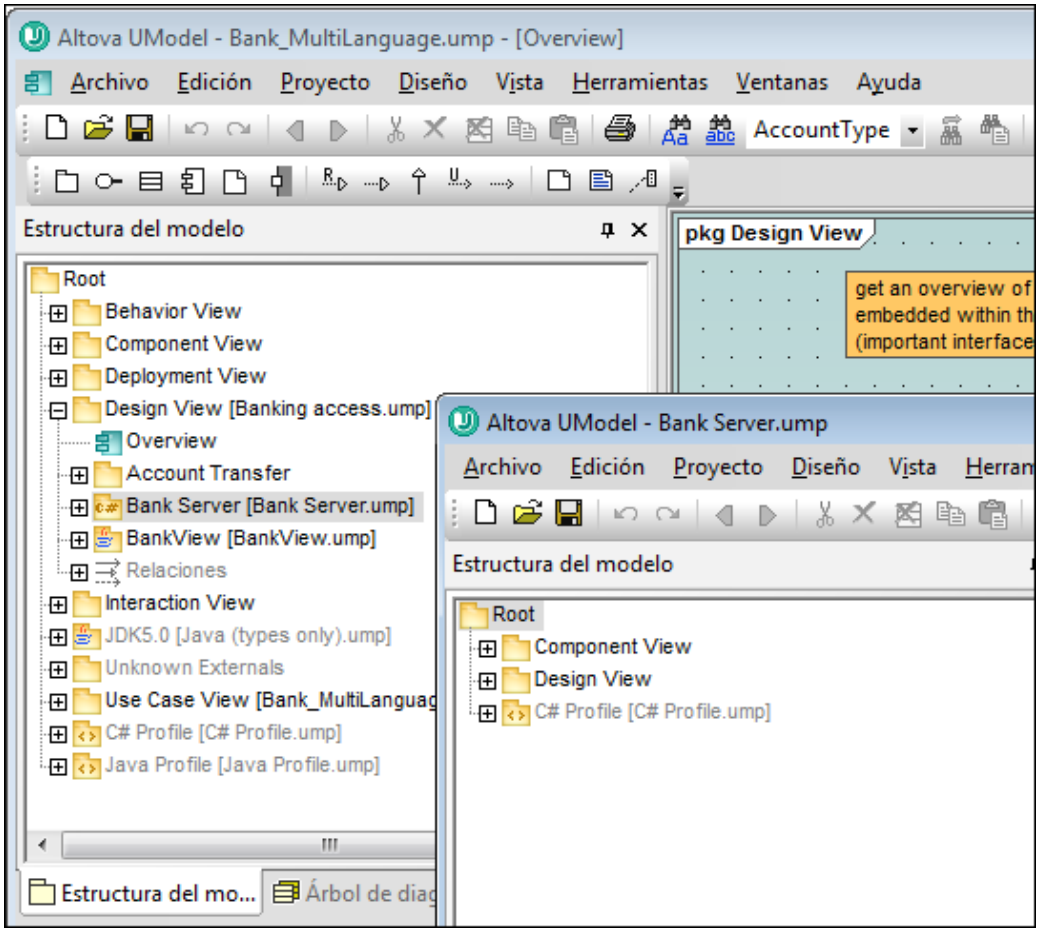
Nota: para cambiar la ruta de acceso del archivo de subproyecto haga clic con el botón derecho en el subproyecto y seleccione **Subproyecto | Editar la ruta de acceso del archivo**.

Para abrir y editar subproyectos:

Una vez creado el subproyecto, el archivo `*.ump` resultante se puede abrir y editar como si fuera un archivo de proyecto principal. Para que esto funcione no puede haber referencias sin resolver a otros elementos. UModel revisa automáticamente el archivo de subproyecto tanto cuando se crea como cuando se guarda.

1. Haga clic con el botón derecho en el paquete de subproyecto (p. ej. `Bank Server.ump`) del proyecto principal y seleccione la opción **Subproyecto | Abrir en forma de proyecto**. Esto inicia otra instancia de UModel y abre el subproyecto como si fuera el proyecto principal. Las referencias no resueltas aparecen en la ventana Mensajes.

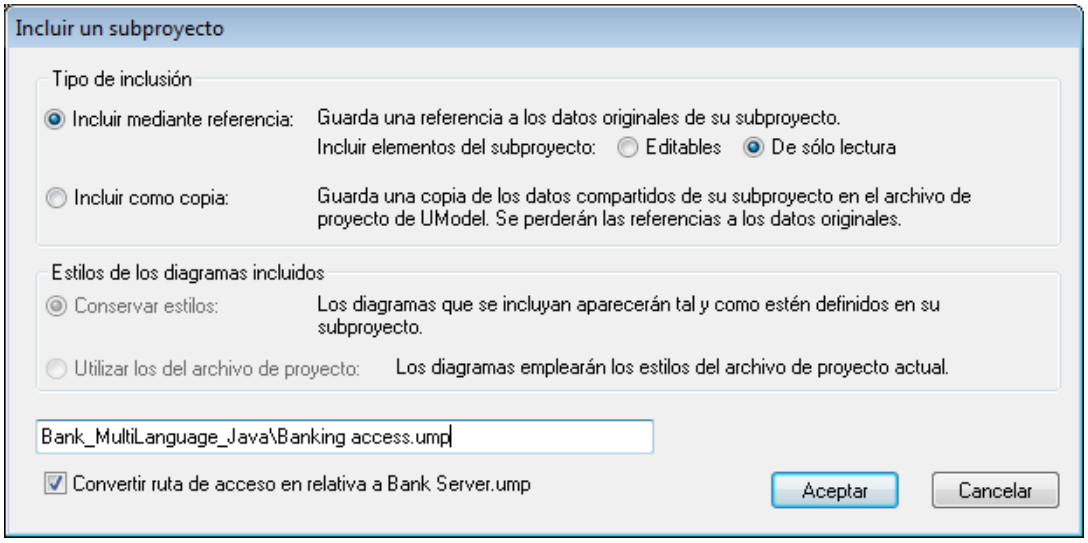
 © 2018 Altova GmbH



Para reutilizar subproyectos en un otros proyectos:

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 un subproyecto**.
2. Haga clic en el botón **Examinar** y seleccione el archivo `*.ump` que desea incluir (p. ej. `Banking access.ump`).



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

Guardar proyectos

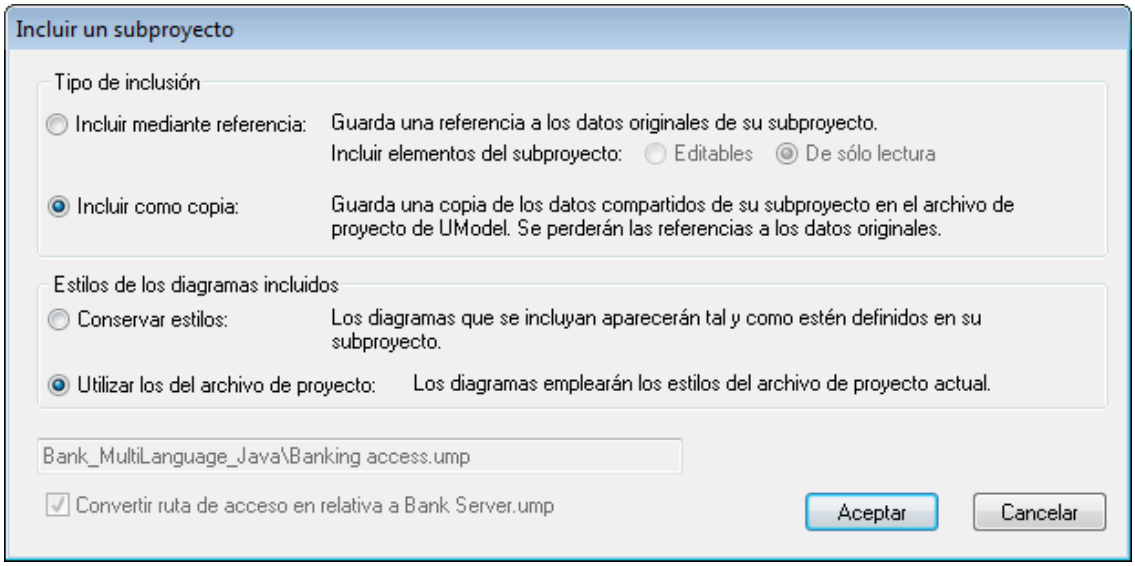
Cuando se guarda el archivo de proyecto principal, también se guardan todos los subproyectos editables (es decir, se guardan todos los datos de los paquetes compartidos de los subproyectos).

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`.

© 2018 Altova GmbH



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:

Los subproyectos se pueden volver a copiar dentro del proyecto principal. Si el subproyecto no contiene diagramas, la reintegración es inmediata. Por el contrario, si existen diagramas, UModel muestra 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**.

Altova UModel 2018

Control de código fuente

12 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.

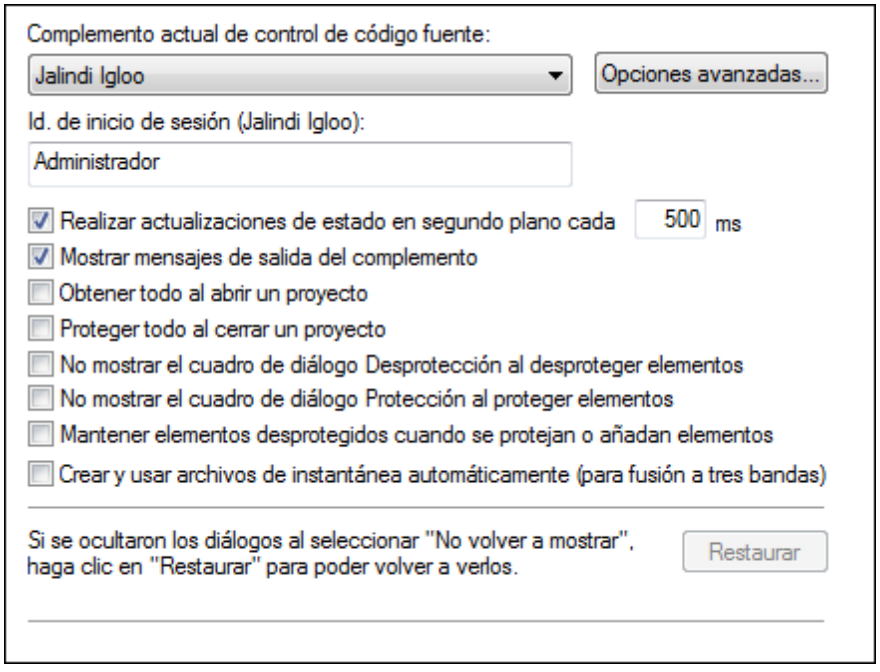
Puede usar cualquier complemento comercial o libre que sea compatible con la API del complemento Microsoft Source Control y puede conectarse a todos los sistemas de control de versiones compatibles (*ver lista de sistemas de control de código fuente compatibles*).

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."
```


© 2018 Altova GmbH

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.

12.1 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

© 2018 Altova GmbH

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

12.2 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\UModel2018\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

12.2.1 Abrir desde el control de código fuente

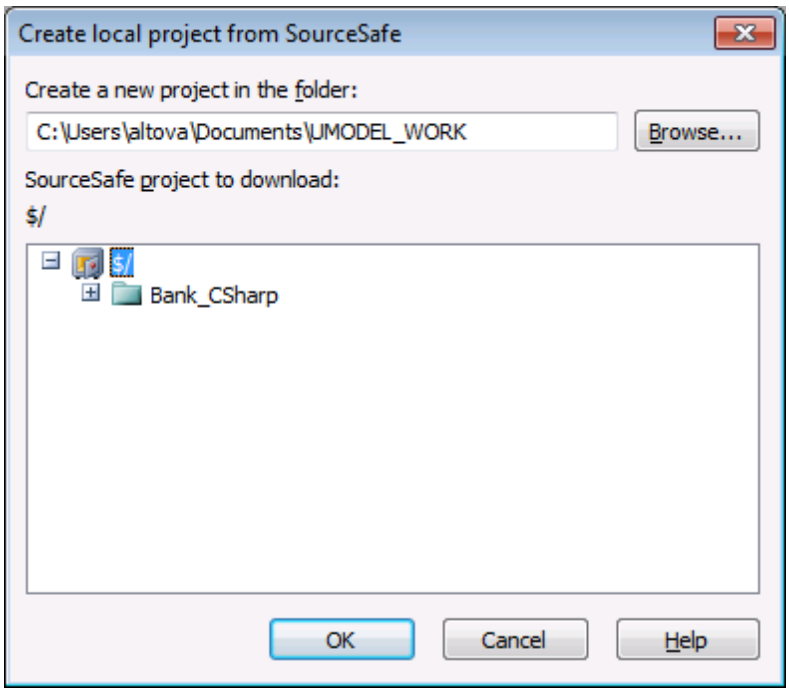
Este comando crea un proyecto local a partir de una BD de control de código fuente ya disponible y lo pone bajo control de código fuente. Para los ejemplos siguientes usamos SourceSafe.

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.

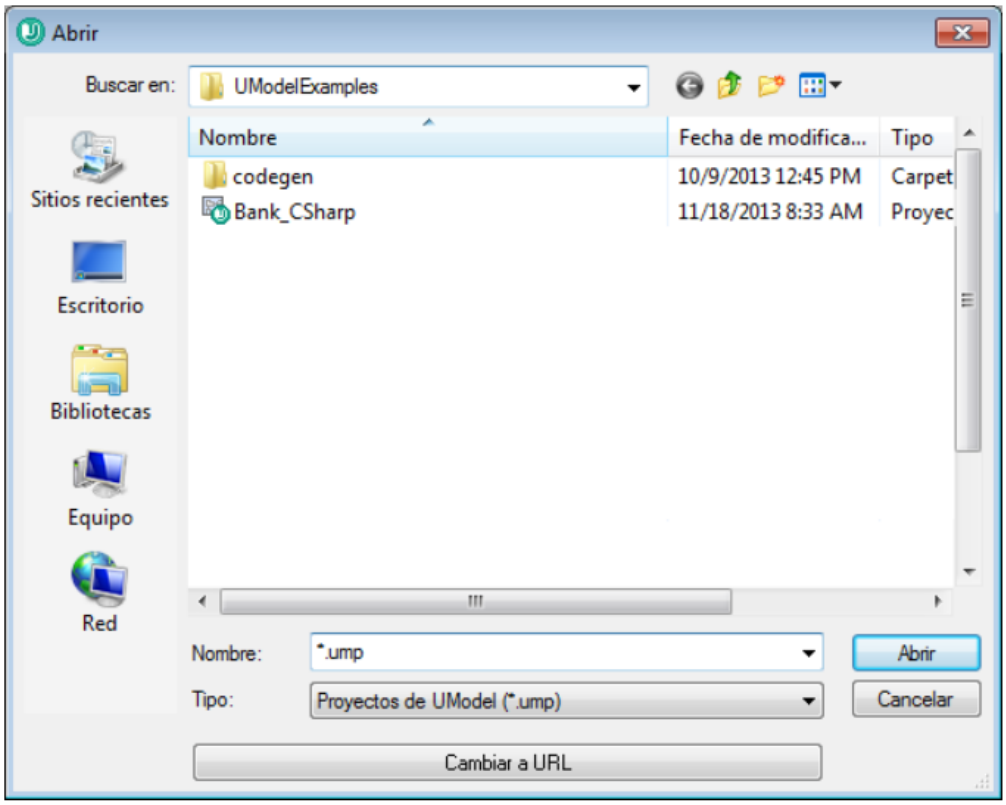
 © 2018 Altova GmbH

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).

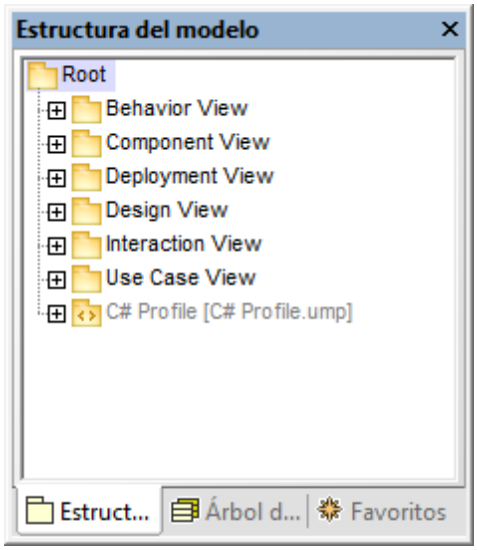


3. Seleccione el proyecto de SourceSafe que desea descargar (p. ej. `Bank_CSharp`).
   Si la carpeta que define aquí no existe en la ubicación, deberá crearla.
4. Haga clic en **Sí** para crear el directorio nuevo. Ahora aparece el cuadro de diálogo "Abrir".



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

 © 2018 Altova GmbH

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

El icono en forma de candado indica que el archivo / la carpeta está bajo control de código fuente pero no está desprotegido.

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.

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.

12.2.2 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.

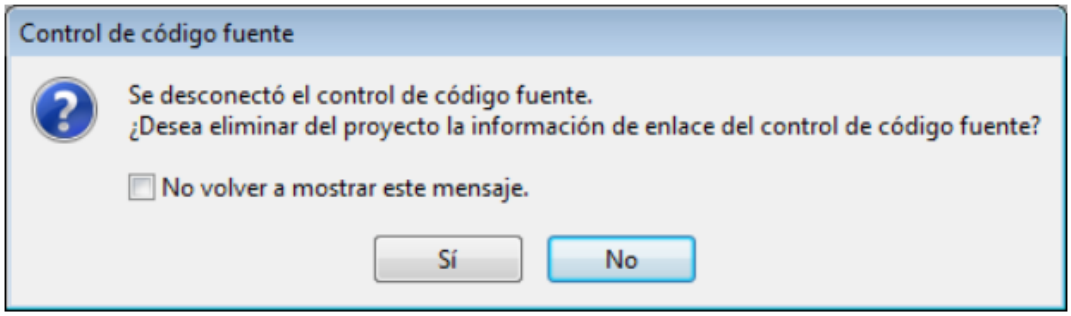
Para habilitar el control de código fuente para un proyecto:

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**.

12.2.3 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**.

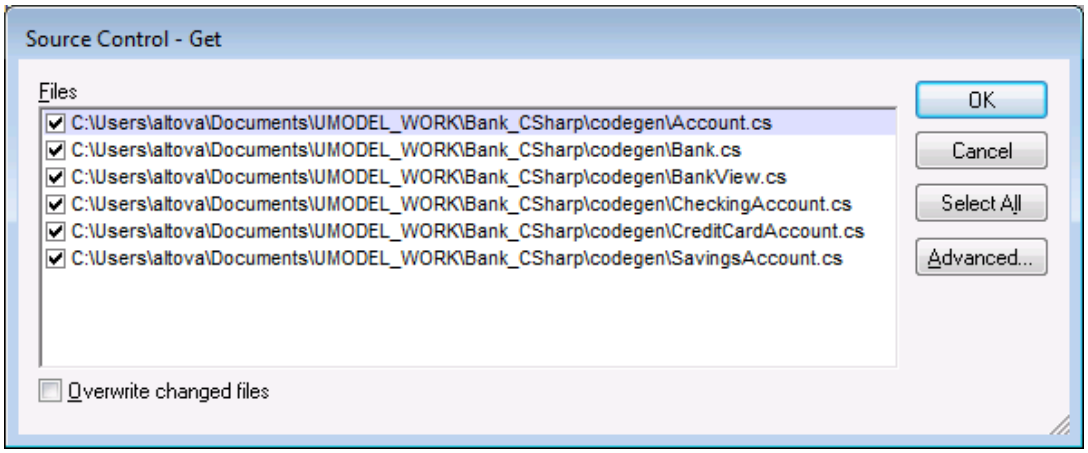
12.2.4 Obtener

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.

© 2018 Altova GmbH

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:

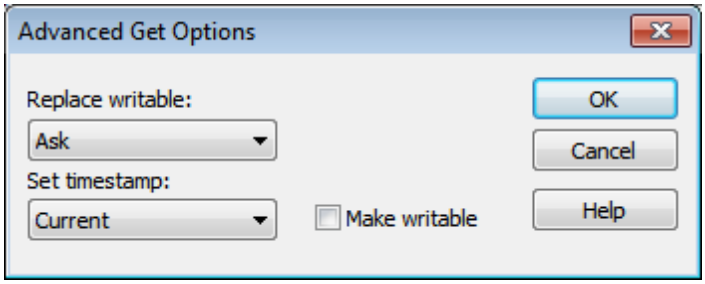


Sobrescribir archivos modificados
Marque esta casilla si quiere sobrescribir los archivos que se modificaron localmente con los archivos de la BD del control de código fuente.

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.

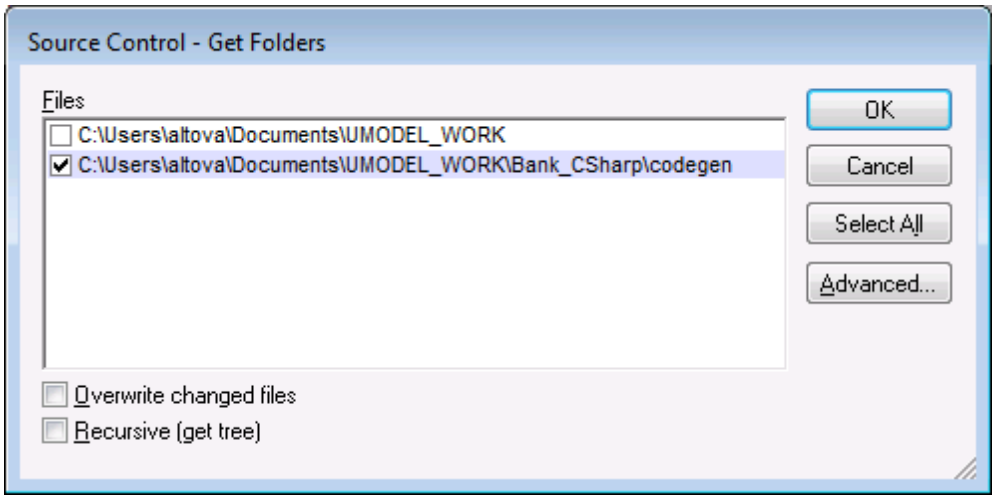
La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

12.2.5 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:



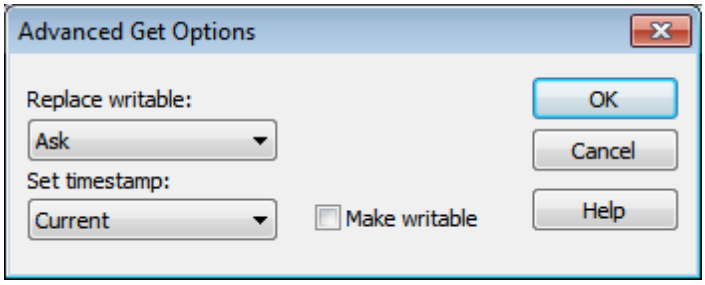
Sobrescribir archivos modificados
Marque esta casilla para sobrescribir los archivos que se modificaron localmente con los archivos de la BD del control de código fuente.

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*).

© 2018 Altova GmbH



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.

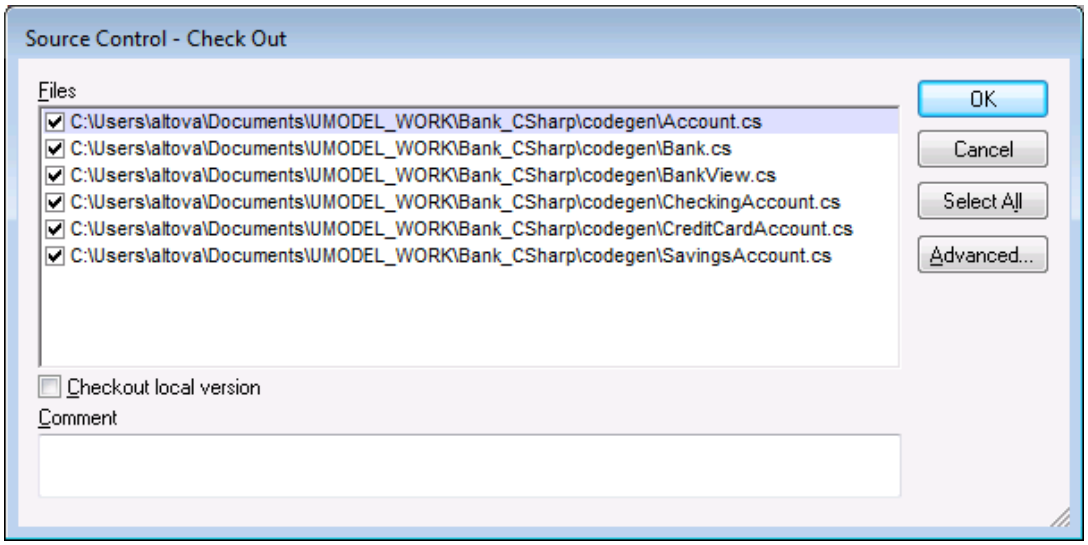
La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

12.2.6 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:

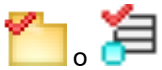
1. En la Estructura del modelo seleccione el archivo o la carpeta que desea desproteger.
2. Haga clic en **Proyecto | Control de código fuente | Desproteger**.
   Aparece un cuadro de diálogo (*imagen siguiente*) donde puede seleccionar qué archivos se desprotegen finalmente (marcando sus casillas). Más abajo se describen las opciones de este cuadro de diálogo.



Desproteger versión local
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:

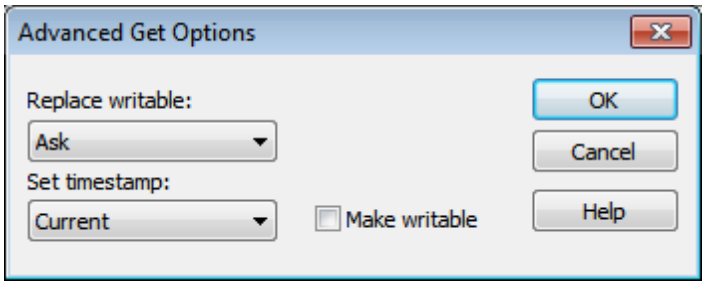
- Archivos (para seleccionar varios, pulse la tecla **Ctrl** mientras hace clic en los archivos en la Estructura del modelo).
- Carpetas (para seleccionar varias, pulse la tecla **Ctrl** mientras hace clic en las carpetas en la Estructura del modelo).


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.

12.2.7 Proteger

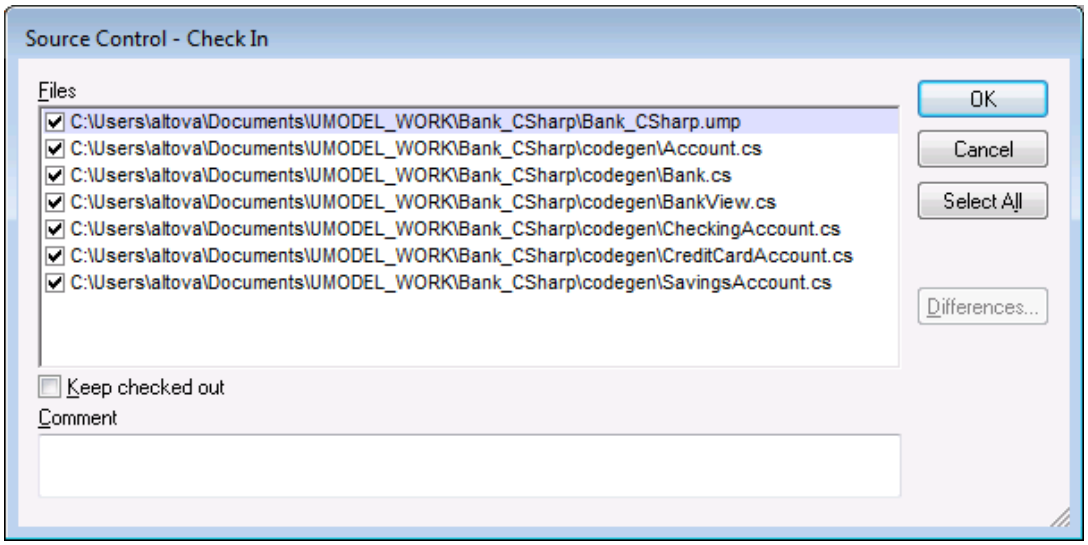
Este comando protege los archivos que estén desprotegidos (es decir, los archivos que se modificaron localmente) y los pone en la BD del control de código fuente.

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.

 © 2018 Altova GmbH



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

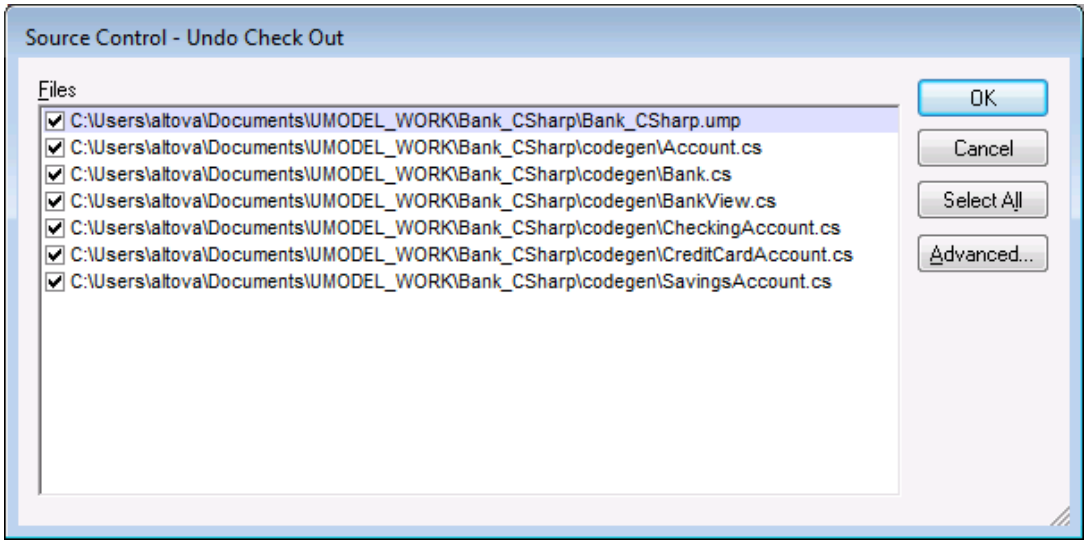
El icono en forma de candado indica que el archivo / la carpeta está **bajo control de código fuente** pero no está desprotegido.

12.2.8 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).

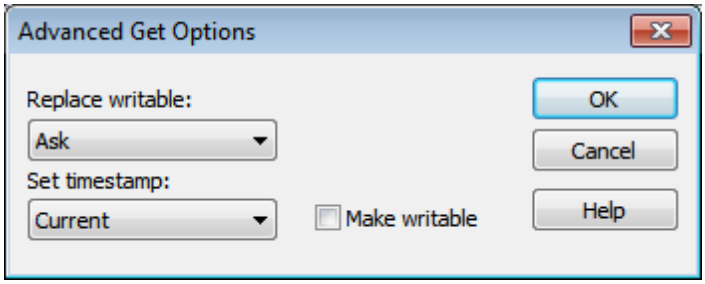


Estos son los elementos cuya desprotección se puede anular:

- 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.

 © 2018 Altova GmbH

12.2.9 Agregar al control de código fuente

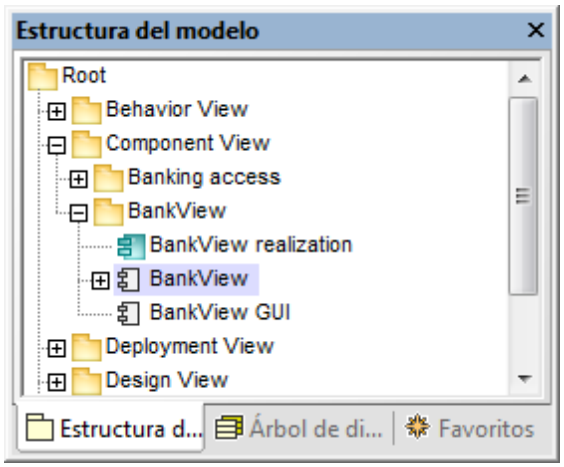
Este comando añade los archivos o las carpetas seleccionados a la BD del control de código fuente y los pone bajo su control. Si añade un proyecto de UModel nuevo, deberá indicar la carpeta del espacio de trabajo y la ubicación donde se debe almacenar el proyecto.

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).

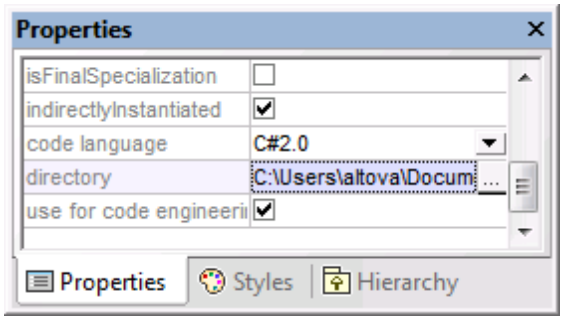
Es importante tener en cuenta que para que esto funcione los archivos de código generados y el proyecto de UModel deben ponerse dentro/bajo el mismo **directorio de trabajo** de SourceSafe. El directorio de trabajo que utilizamos para este ejemplo es `C:\Users\Altova\Documents\UMODEL_WORK\`.

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`**.



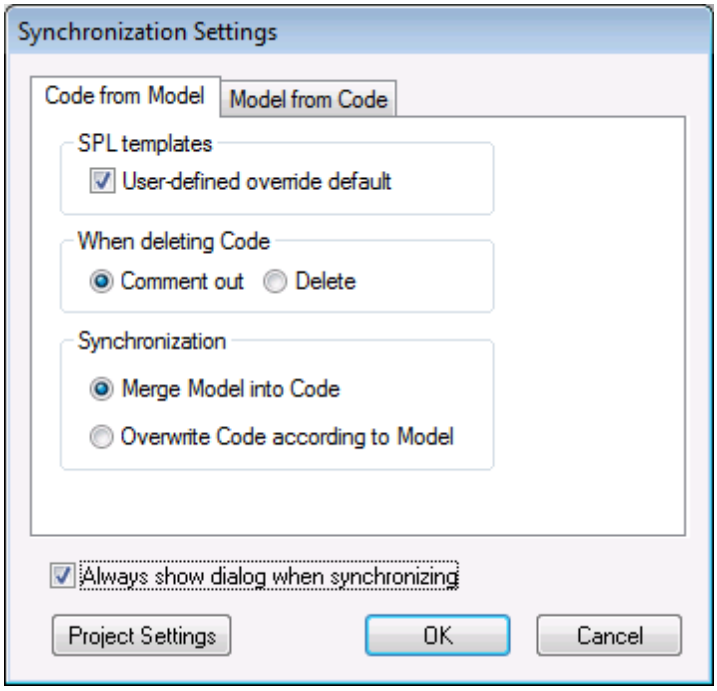
2. Haga clic en el componente **`BankView`**. En la ventana Propiedades haga clic en el icono **Examinar** del campo `directorio`.



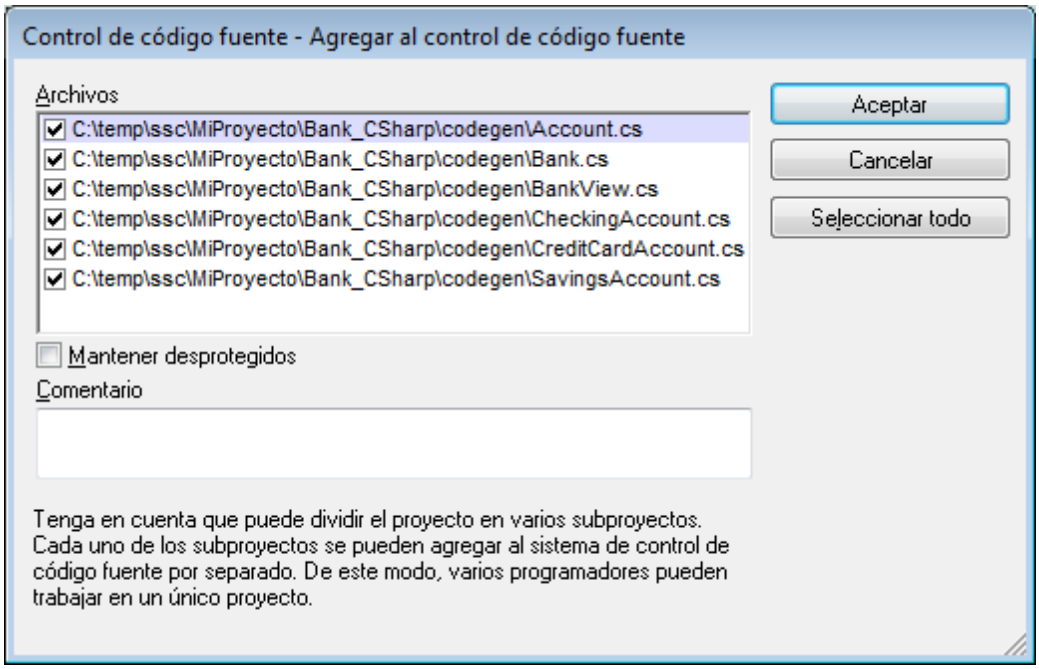
3. Cambie el directorio de ingeniería de código a `C:\Users\Altova\Documents\UMODEL_WORK\codegen`.
4. Ahora haga clic en **Proyecto | Combinar el código de programa con el proyecto de**

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.

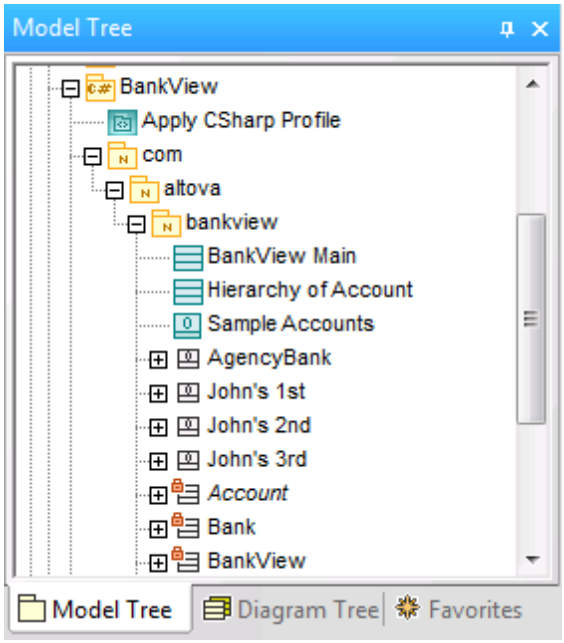


6. Haga clic en **Sí**.
7. Aparece el cuadro de diálogo "Agregar al control de código fuente" donde puede seleccionar qué archivos se ponen bajo control de código fuente.



 © 2018 Altova GmbH

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.

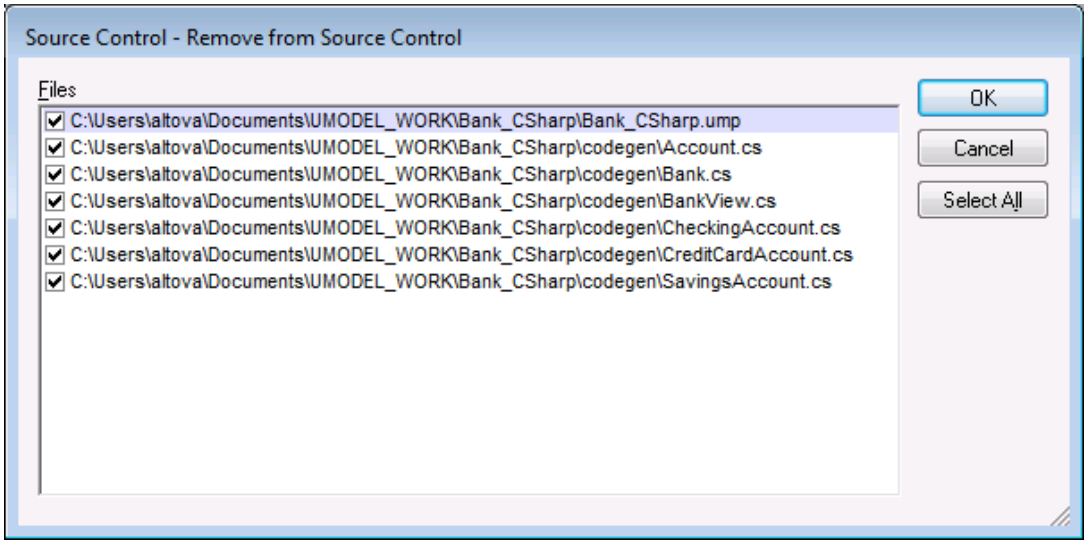


12.2.10 Quitar del control de código fuente

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).

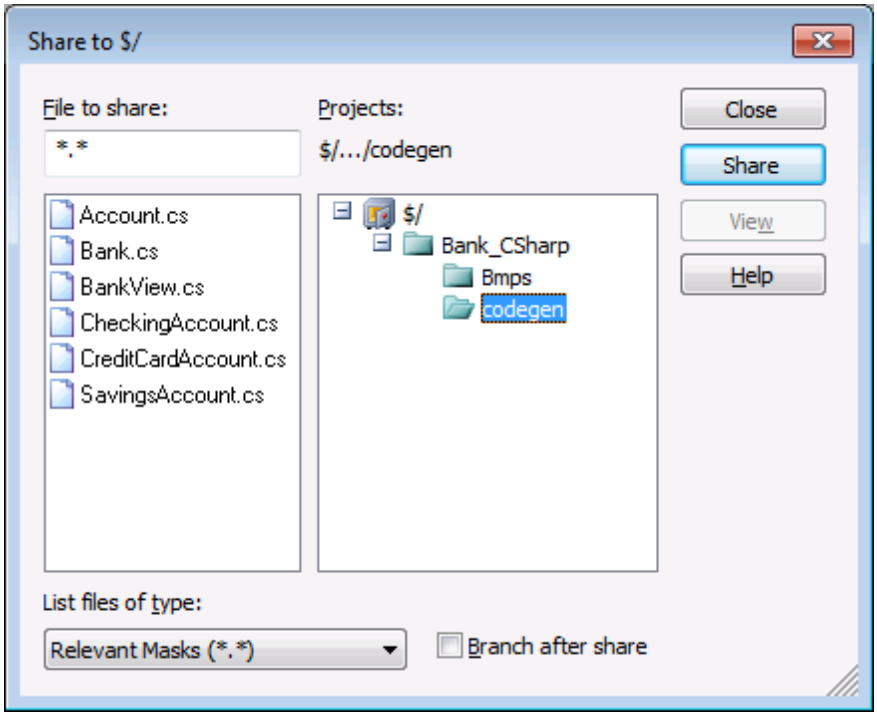
12.2.11 Compartir desde el control de código fuente

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.

 © 2018 Altova GmbH



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.

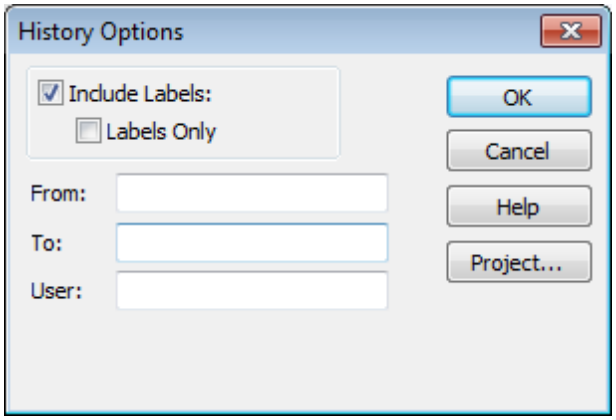
   Nota: marque la casilla *Branch after share* para compartir el archivo y crear una rama nueva para crear una nueva versión.

12.2.12 Mostrar historial

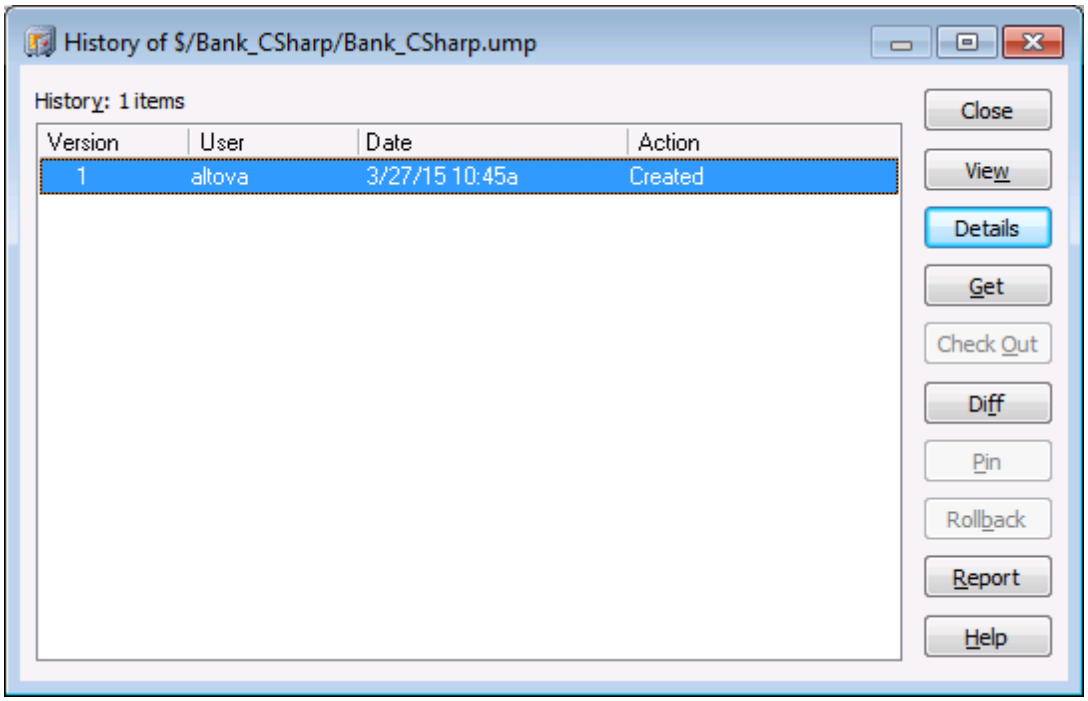
Este comando **muestra el historial** de un archivo que está bajo control de código fuente. El historial permite ver la historia de cambios por la que ha pasado el archivo, ver diferencias entre sus diferentes versiones y recuperar versiones previas.

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.

 © 2018 Altova GmbH

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.

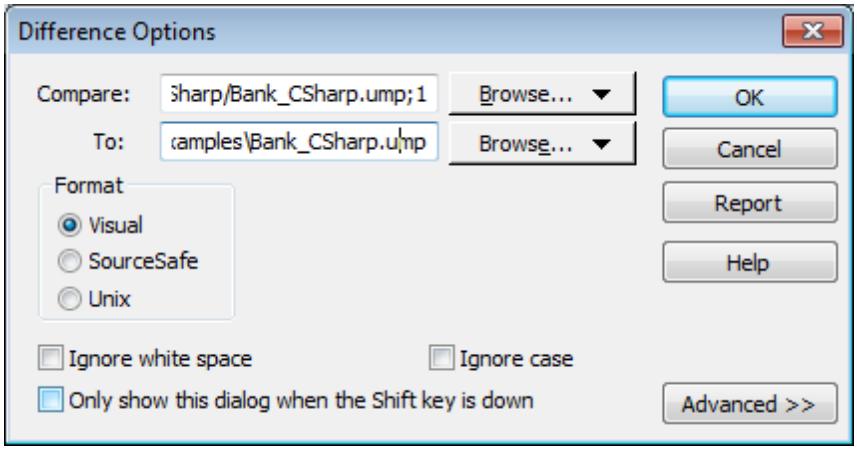
12.2.13 Mostrar diferencias

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.

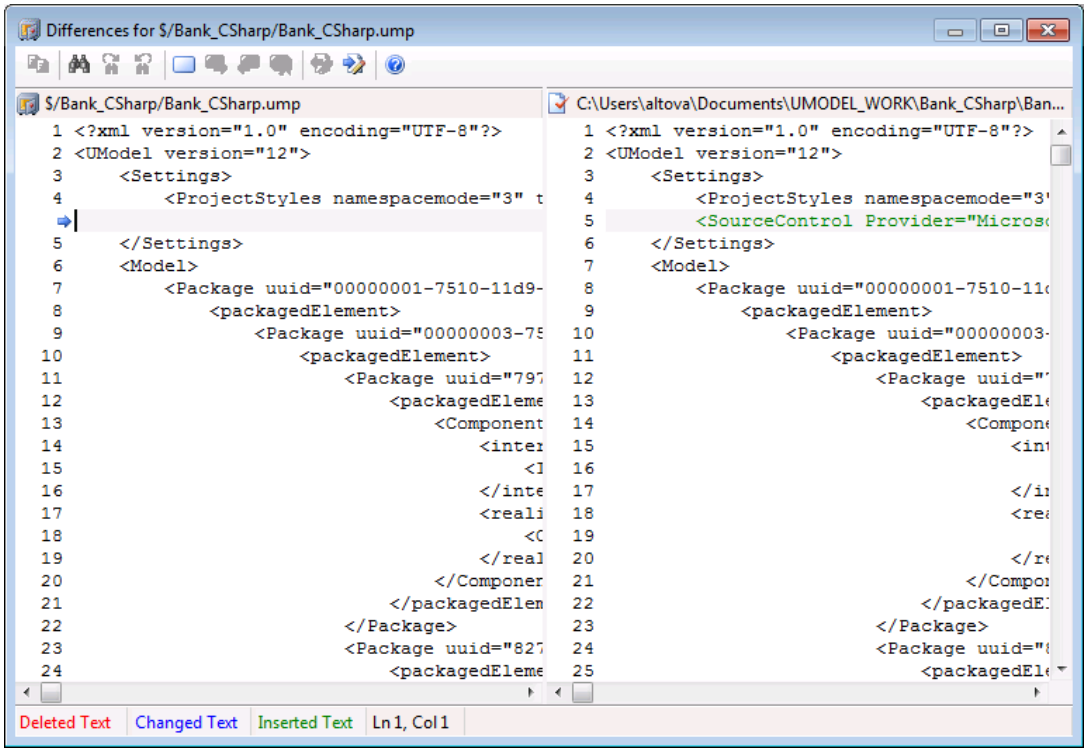
Si ancló uno de los archivos en el cuadro de diálogo del historial, el archivo anclado se inserta automáticamente en el campo de texto *Compare*. Con los botones **Browse** puede buscar los archivos que desea comparar.

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.

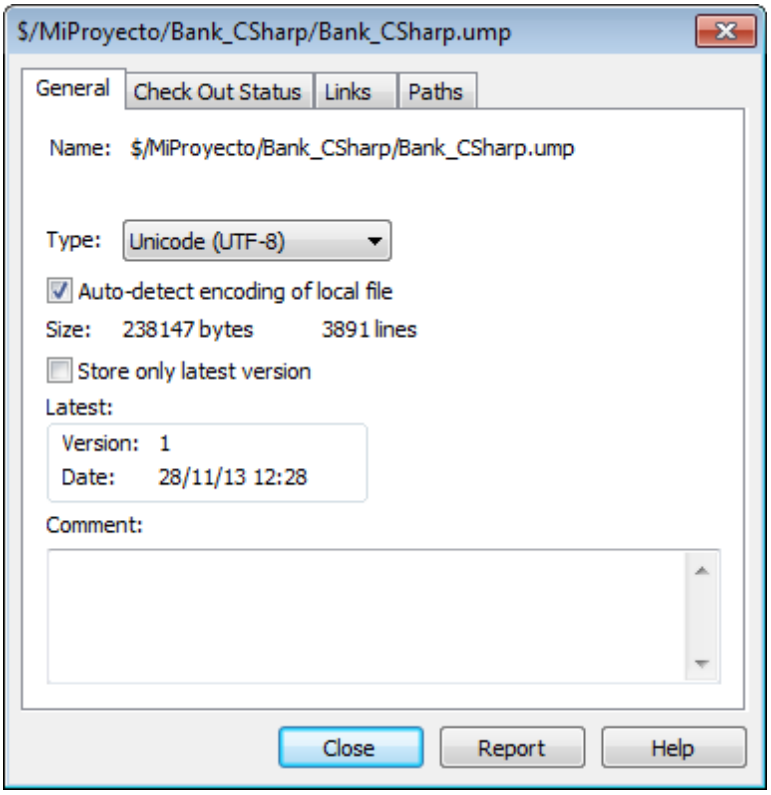
12.2.14 Mostrar propiedades

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.



12.2.15 Actualizar estado

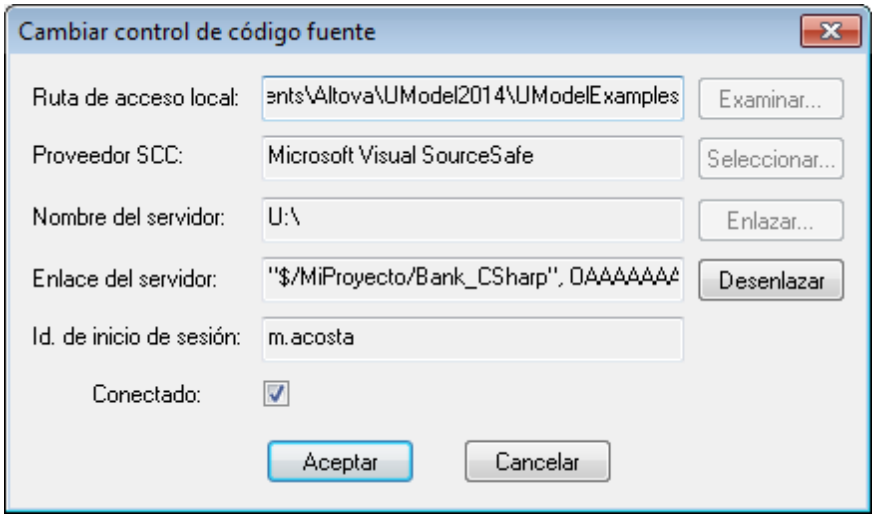
Este comando **actualiza** el estado de todos los archivos de proyecto, independientemente de cuál sea su estado actual.

12.2.16 Administrador del control de código fuente

Este comando **inicia** el software de control de código fuente, con su interfaz de usuario nativa.

12.2.17 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.



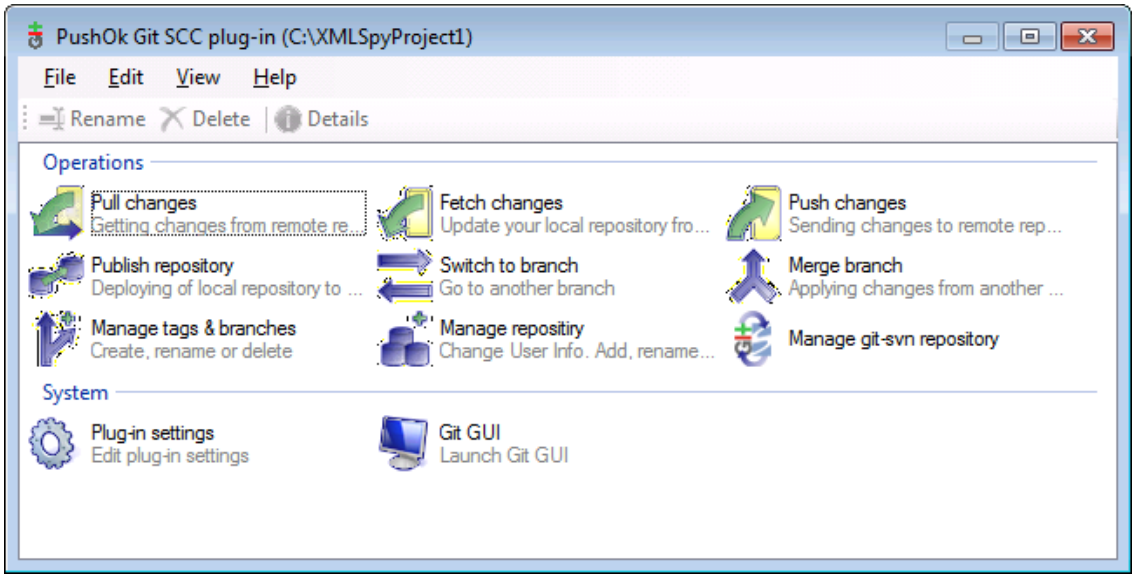
© 2018 Altova GmbH

12.3 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.

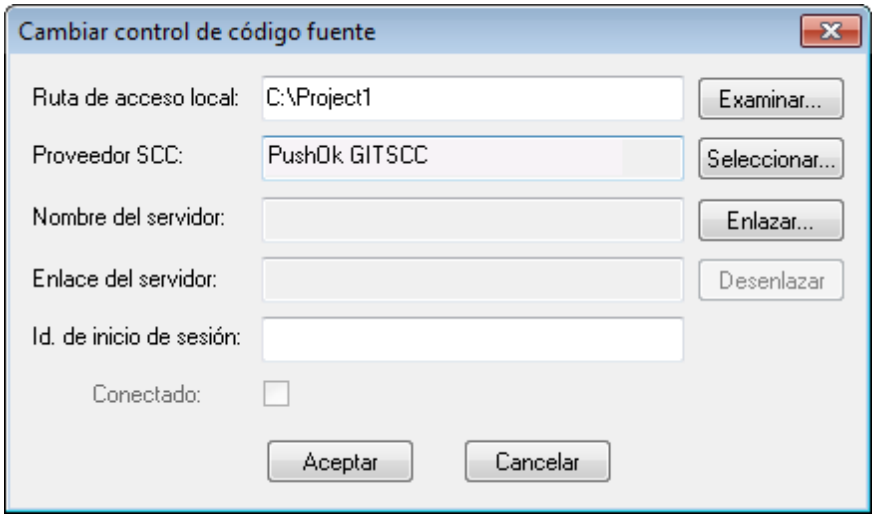
Los diferentes apartados de esta sección describen la configuración inicial del complemento y el flujo de trabajo básico:

- 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

12.3.1 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.



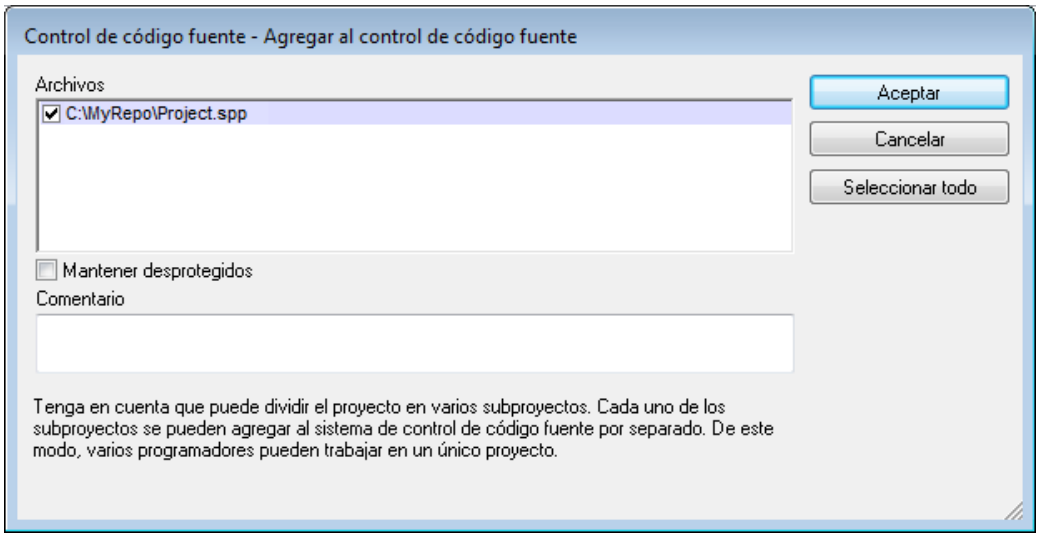
3. Para terminar debe registrar el complemento haciendo clic en **Registration**. Siga los pasos del asistente para terminar de registrar el complemento.

12.3.2 Agregar un proyecto al control de código fuente de Git

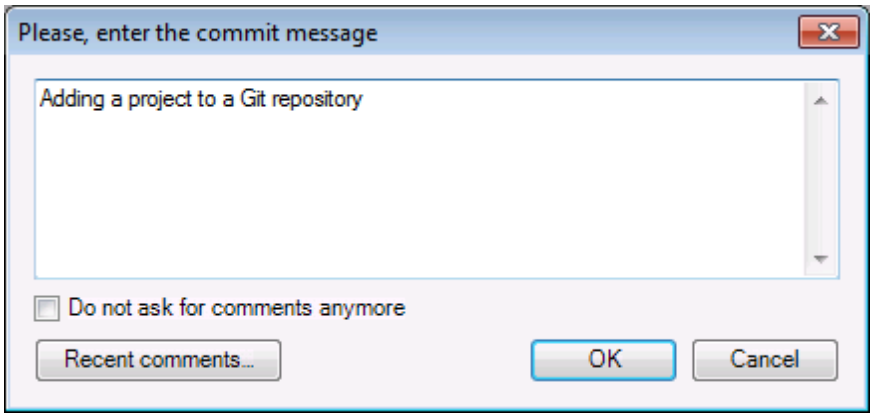
Puede guardar proyectos de UModel como repositorios de Git. La estructura de los archivos o carpetas que añada al proyecto se corresponderán con la estructura del repositorio Git.

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).

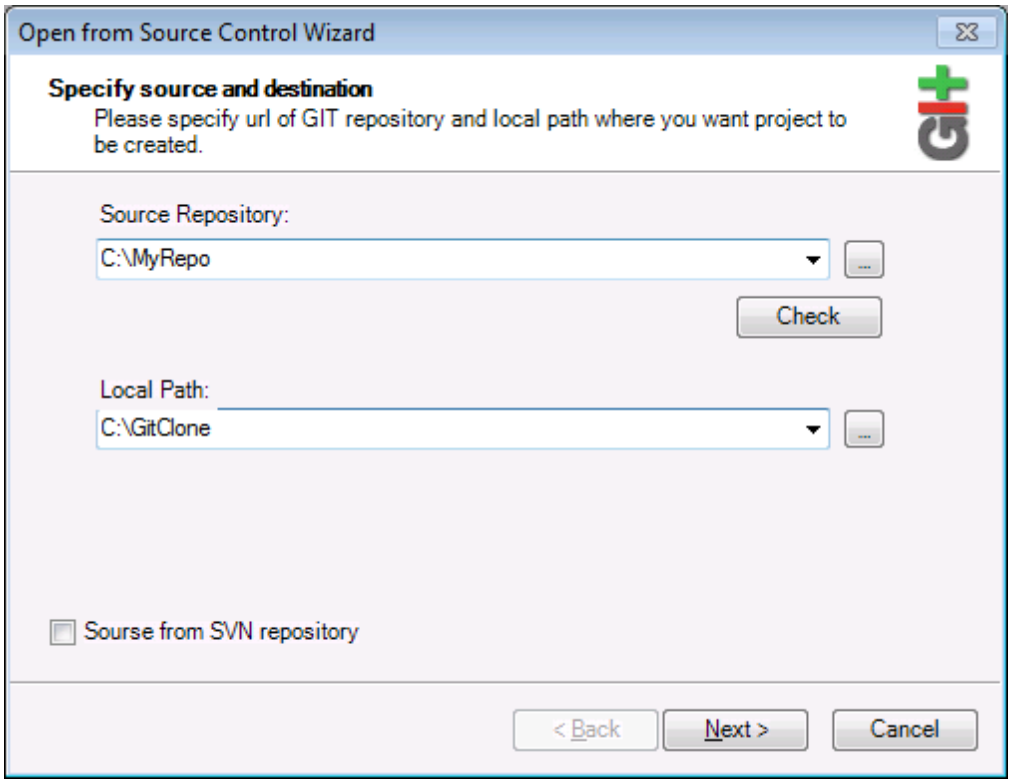
12.3.3 Clonar un proyecto desde el control de código fuente de Git

Los proyectos que ya estén en el control de código fuente de Git (*ver el apartado anterior*) se pueden abrir desde el repositorio Git:

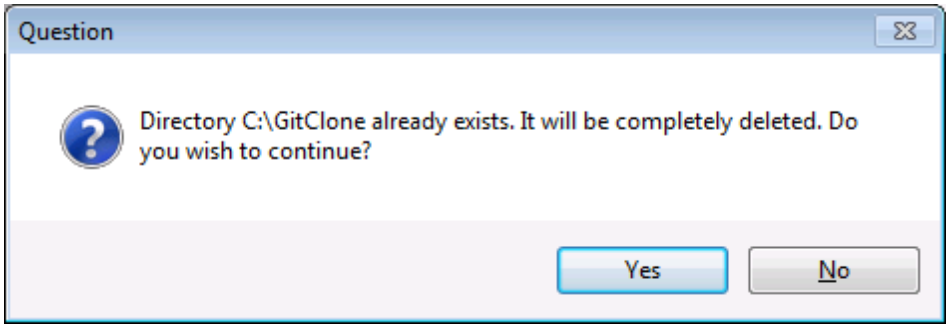
1. Compruebe que el proveedor de control de código fuente seleccionado es **PushOK GIT**

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.

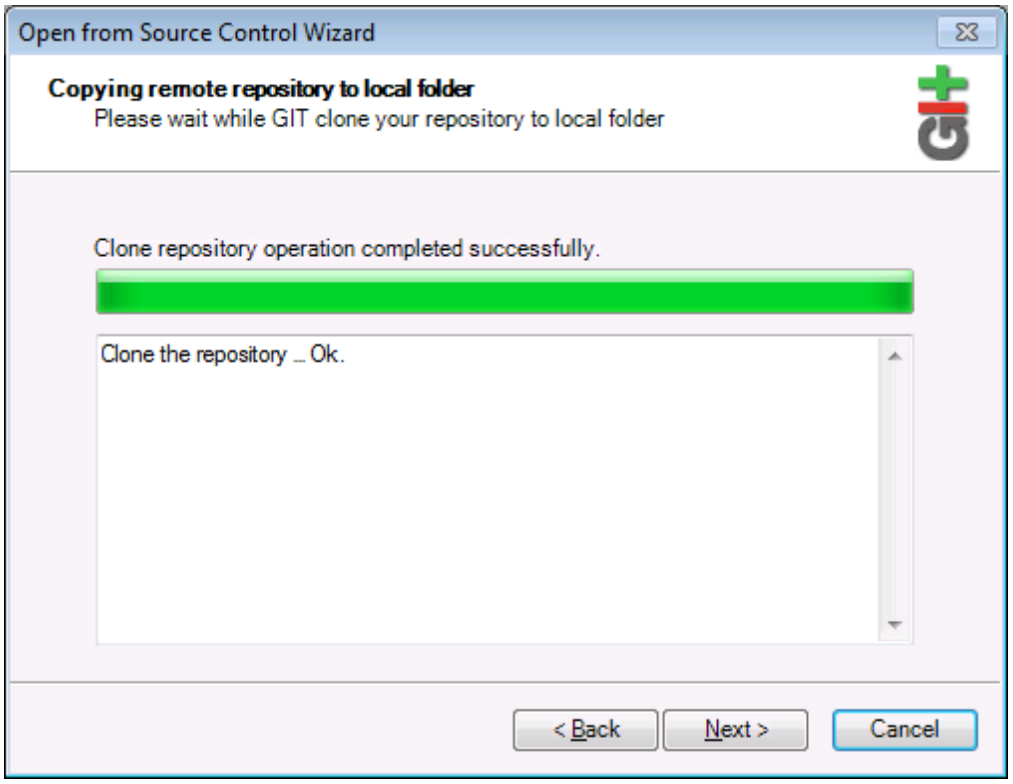


4. En el campo *Local Path* escriba la ruta de acceso de la carpeta local donde desea crear el proyecto y haga clic en **Next** para continuar. Si la carpeta local ya existe (aunque esté vacía), aparece este cuadro de diálogo preguntando si desea borrar totalmente la carpeta:



5. Haga clic en **Yes** para confirmar y después en **Next** para continuar.

© 2018 Altova GmbH



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.

Altova UModel 2018

Iconos en los diagramas de UModel

13 Iconos en los diagramas de UModel

En UModel cada tipo de diagrama tiene una barra de herramientas distinta. Cada barra de herramientas ofrece 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.

© 2018 Altova GmbH

13.1 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

© 2018 Altova GmbH

13.2 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

13.3 Diagramas de comunicación

Agregar:
LíneaDeVida
Mensaje (Llamada)
Mensaje (Respuesta)
Mensaje (Creación)
Mensaje (Destrucción)

Nota
Enlace de nota

© 2018 Altova GmbH

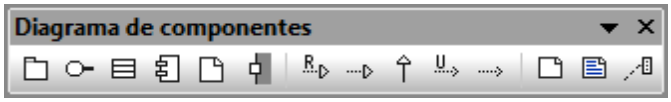
13.4 Diagramas de estructura de un compuesto

Agregar:
Colaboración
UsoDeColaboración
Parte (Propiedad)
Clase
Interfaz
Puerto

Relaciones:
Conector
Dependencia (Enlace de roles)
RealizaciónDeInterfaz
Utilización

Nota
Enlace de nota

13.5 Diagramas de componentes



Agregar:
Paquete
Interfaz
Clase
Componente
Artefacto

Relaciones:
Realización
RealizaciónDeInterfaz
Utilización
Dependencia

Nota
Enlace de nota

© 2018 Altova GmbH

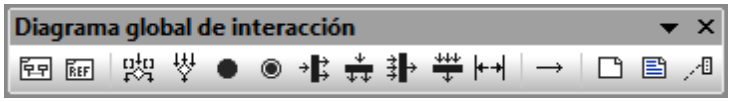
13.6 Diagramas de implementación

Agregar:
Paquete
Componente
Artefacto
Nodo
Dispositivo
EntornoDeEjecución

Relaciones:
Manifestación
Implementación
Asociación
Generalización
Dependencia

Nota
Enlace de nota

13.7 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

© 2018 Altova GmbH

13.8 Diagramas de objetos

Agregar:
Paquete
Clase
Interfaz
Enumeración
TipoDeDatos
TipoPrimitivo
EspecificaciónDeInstancia

Relaciones:
Asociación
ClaseDeAsociación
Dependencia
Utilización
RealizaciónDeInterfaz
Generalización

Nota
Enlace de nota

13.9 Diagramas de paquetes

Agregar:
Paquete
Perfil

Relaciones:
Dependencia
ImportaciónDePaquete
CombinaciónDePaquete
AplicaciónDePerfil

Nota
Enlace de nota

© 2018 Altova GmbH

13.10 Diagramas de perfil

Agregar:
Perfil
Estereotipo

Relaciones:
Generalización
AplicaciónDePerfil
ImportaciónDePaquete
ImportaciónDeElemento

Nota
Enlace de nota

13.11 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

© 2018 Altova GmbH

13.12 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

13.13 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

© 2018 Altova GmbH

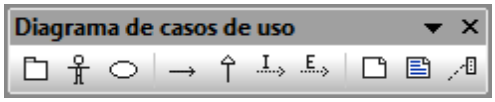
13.14 Diagramas de ciclo de vida

Agregar
LíneaDeVida (Estado o Condición)
LíneaDeVida (Valor general)
MarcaDeGraduación
Evento/estímulo
RestricciónDeDuración
RestricciónDeTiempo

Mensaje (Llamada)
Mensaje (Respuesta)
Mensaje asíncrono (Llamada)

Nota
Enlace de nota

13.15 Diagramas de casos de uso



Agregar:
Paquete
Actor
CasoDeUso

Relaciones:
Asociación
Generalización
Inclusión
Extensión

Nota

© 2018 Altova GmbH

13.16 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

Altova UModel 2018

Referencia del usuario

14 Referencia del usuario

Esta sección repasa uno a uno todos los menús y comandos de menú de UModel, dando una breve descripción de ellos.

© 2018 Altova GmbH

14.1 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 o .EMF (metarchivo mejorado). 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 o .EMF (metarchivo mejorado).

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.

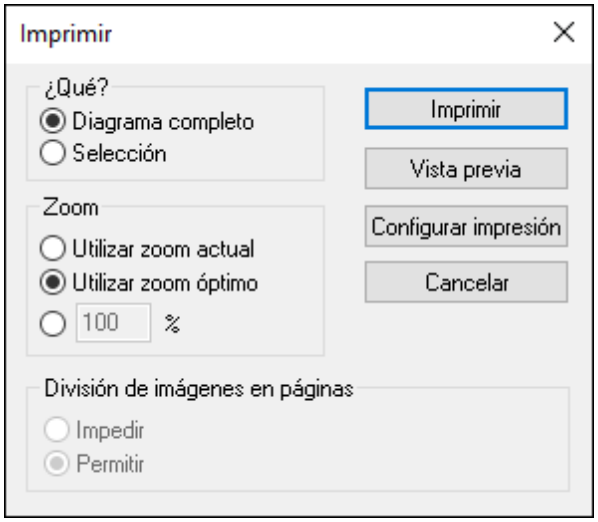
Exportar a un archivo XMI
Este comando exporta el modelo como archivo XMI (*imagen siguiente*). Puede seleccionar la versión de UML y los identificadores que desea exportar. Para más información consulte el apartado XMI: intercambio de metadatos XML.

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 definir la impresora que desea usar y seleccionar la configuración del papel.

© 2018 Altova GmbH

14.2 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 ir adelante y atrás en 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 sólo 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 sólo 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.

© 2018 Altova GmbH

14.3 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 la sección 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 binarios Java, C# y VB.NET.

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*).

Generar diagramas de secuencia a partir del código
Abre el cuadro de diálogo "Seleccione al menos una operación", donde puede seleccionar las operaciones que deben utilizarse como base del diagrama de secuencia generado (*véanse Generar diagramas de secuencia a partir de código fuente y Generar varios diagramas de secuencia a partir de propiedades*).

Combinar el código de programa con el proyecto de UModel / 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 el proyecto de UModel con el código de programa / 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*).

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, sin embargo, se pueden dejar para más tarde. En UModel los errores y advertencias los genera 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*).

 © 2018 Altova GmbH

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.

14.4 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, dependiendo de la opción elegida.

Espaciar uniformemente

Este grupo de comandos sirve para espaciar los elementos seleccionados uniformemente, en horizontal o en vertical.

Igualar tamaño

Este grupo de comandos sirve para ajustar el ancho y el alto de los elementos seleccionados al ancho/alto del elemento activo.

Poner en fila

Este grupo de comandos sirve para poner en fila vertical u horizontal los elementos seleccionados.

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 hasta alcanzar un tamaño óptimo.

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**.

© 2018 Altova GmbH

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.

14.5 Menú Vista

Los comandos de este menú sirven para:

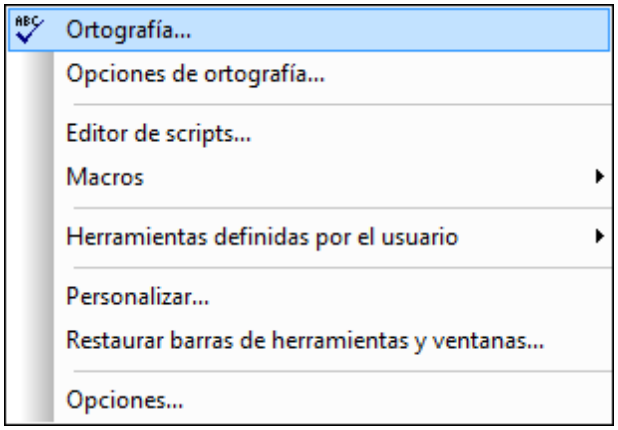
- Mostrar u ocultar las ventanas auxiliares de UModel (*véase* *Interfaz gráfica del usuario de UModel*).
- Definir el criterio de ordenación en los elementos que están dentro de los paneles *Estructura del modelo* y *Favoritos*.
- Definir el criterio de agrupación de los diagramas en la ventana *Árbol 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*).

© 2018 Altova GmbH

14.6 Menú Herramientas

Los comandos del menú **Herramientas** sirven para:

- 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.



14.6.1 Herramientas definidas por el usuario

Al hacer clic 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.

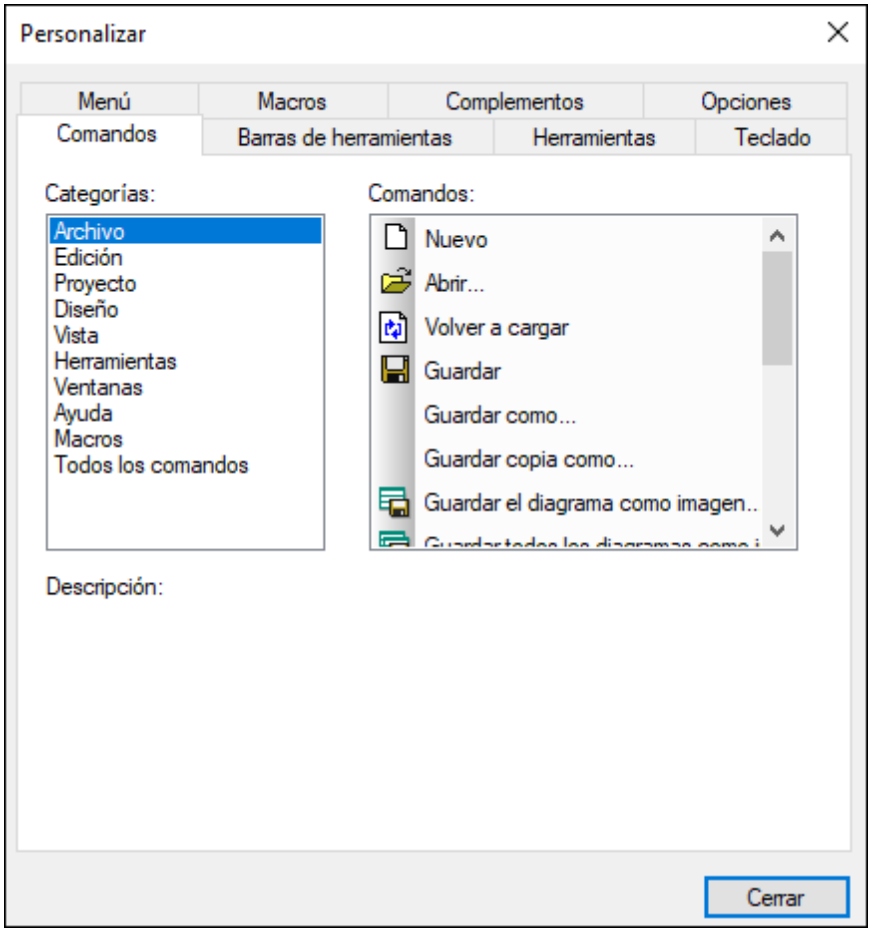
14.6.2 Personalizar

El comando **Personalizar** abre el cuadro de diálogo "Personalizar", que sirve para personalizar los siguientes elementos de la interfaz gráfica de UModel:

- *Comandos*
- *Barras de herramientas*
- *Herramientas*
- *Teclado*
- *Menú*
- *Opciones*

14.6.2.1 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 ya existente. 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.

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

 © 2018 Altova GmbH

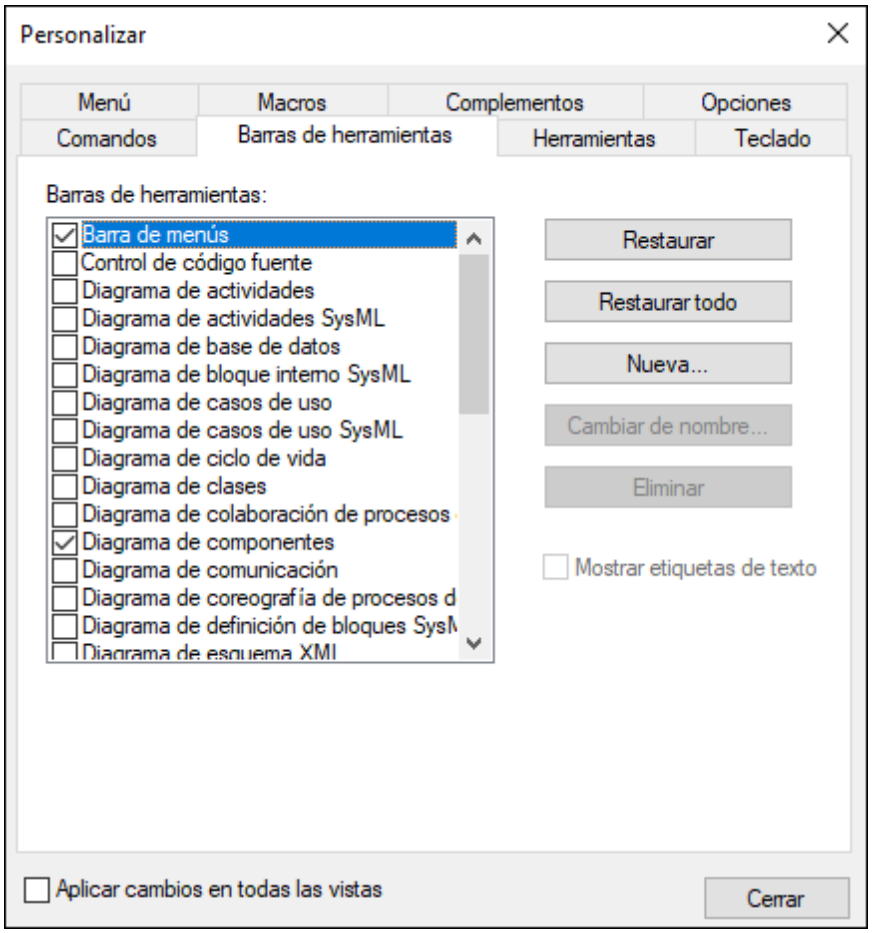
barra de herramientas con comandos.
- También puede añadir comandos a los menús contextuales (los que se abren al hacer clic con el botón derecho en cualquier parte) haciendo clic en la pestaña *Menú* del cuadro de diálogo "Personalizar" y seleccionando un menú contextual del panel *Menús contextuales,* 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.

14.6.2.2 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.

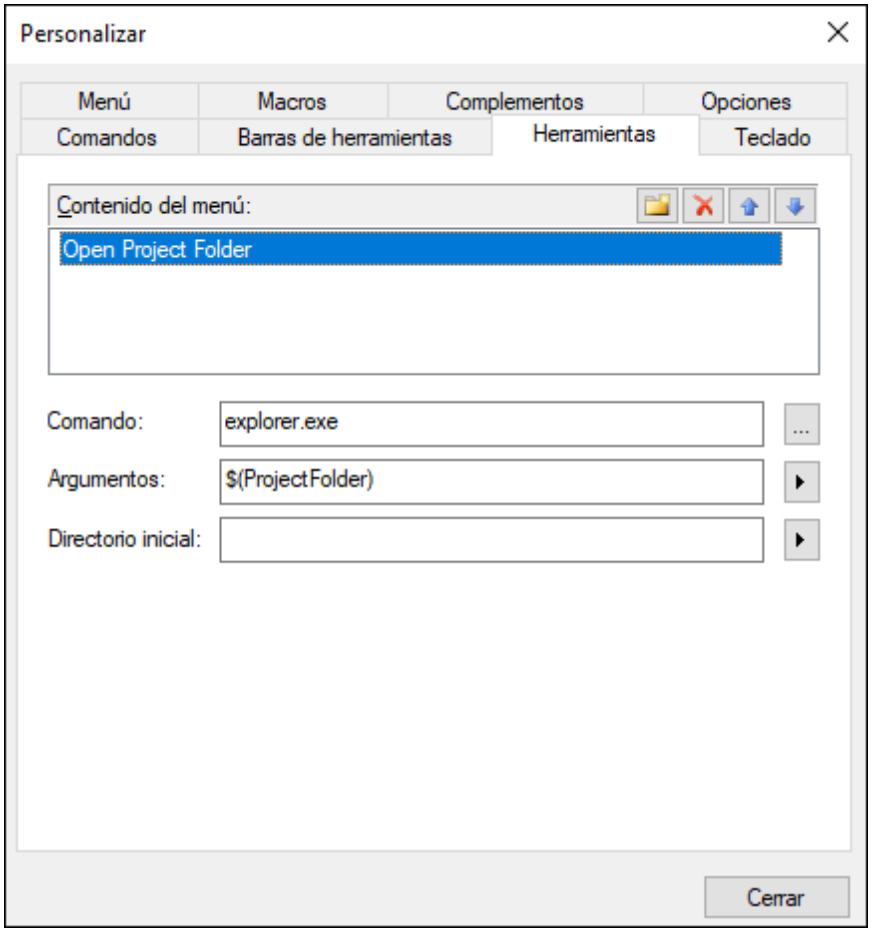
Si marca la casilla *Mostrar etiquetas de texto* parece texto explicativo junto a los iconos de la barra de herramientas.

14.6.2.3 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.

 © 2018 Altova GmbH



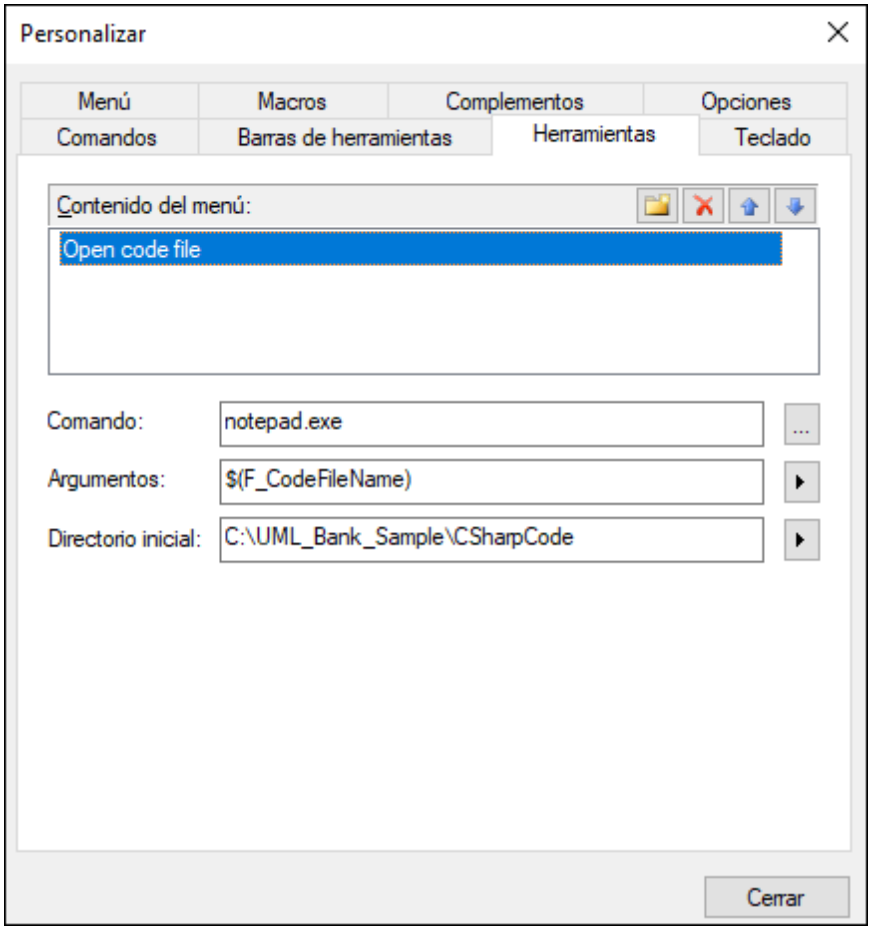
Cuando una herramienta externa toma argumentos (como el explorador de Windows en la imagen anterior), estos se pueden introducir en el campo *Argumentos*. Para usar varios argumentos, sepárelos con espacios. Los valores que introduzca como argumentos pueden ser texto simple (valores incrustados) o se 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\UModel2018\UModelExamples**.
Carpeta temporal	La carpeta en la que se guardan los archivos temporales de la aplicación. Por ejemplo: **C:\Users \<user>\AppData\Local\Temp**.

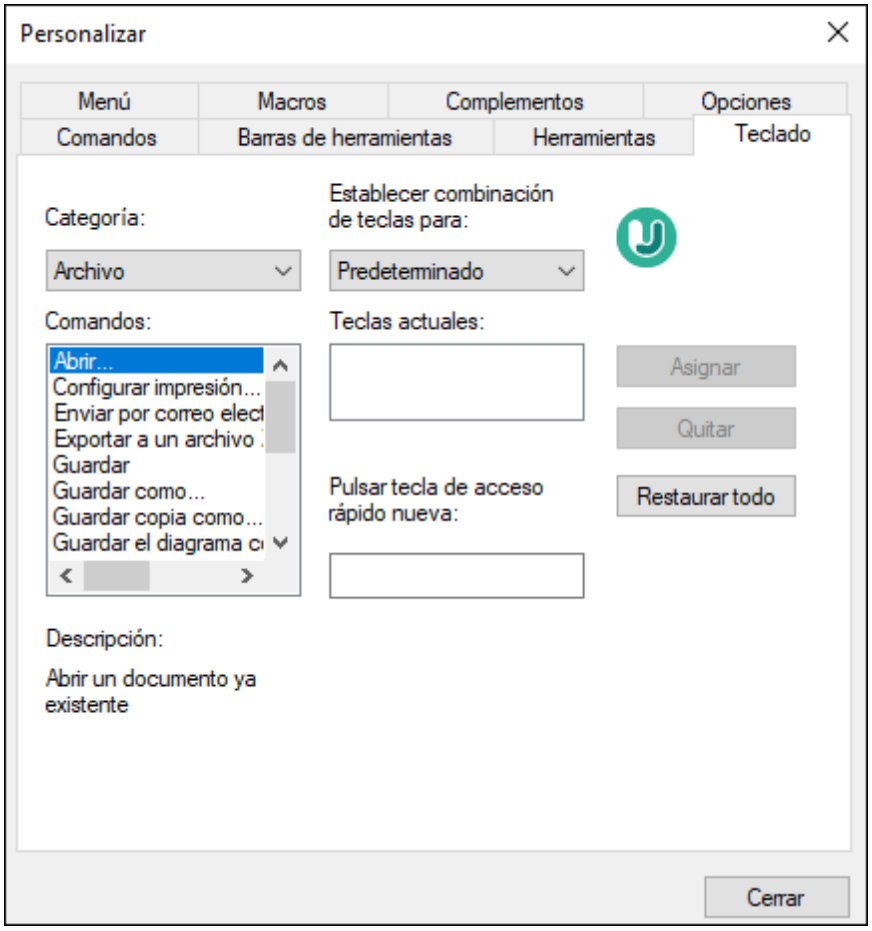
En algunos casos puede que también tenga que introducir un valor en el campo *Directorio inicial*. Por ejemplo, la configuración de la imagen siguiente abre en Notepad el archivo de código del elemento seleccionado en un diagrama. Tenga en cuenta que para que funcione este comando el elemento seleccionado 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.

 © 2018 Altova GmbH



14.6.2.4 Teclado

En la pestaña *Teclado* puede crear teclas de acceso rápido nuevas o cambiar las teclas de acceso rápido ya existentes para cualquier comando de la aplicación.



Para asignar una tecla de acceso rápido nueva a un comando o cambiar una tecla de acceso rápido ya existente:

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:

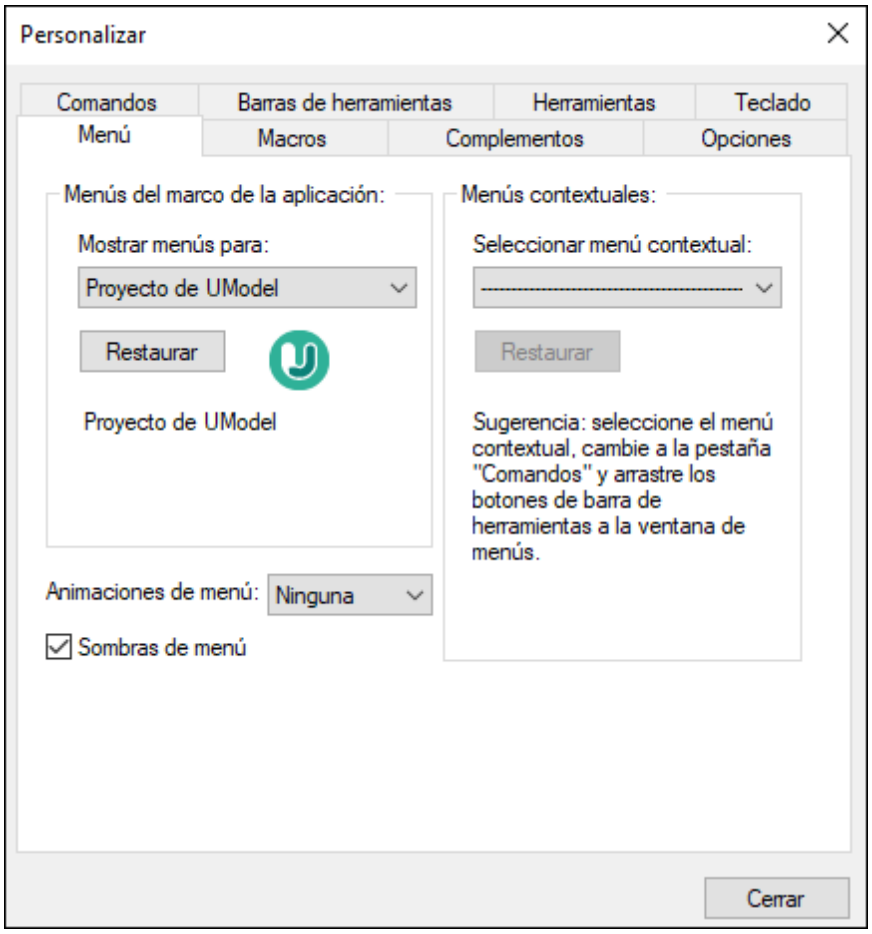
1. En el cuadro de lista *Teclas actuales* seleccione la tecla de acceso rápido que desea eliminar.

 © 2018 Altova GmbH

2. Pulse el botón **Quitar**.
3. Para confirmar los cambios pulse el botón **Cerrar**.

14.6.2.5 Menú

En la pestaña *Menú* puede personalizar las barras de menú principales, así como los menús contextuales de la aplicación.



Para personalizar un menú:

1. Seleccione *Proyecto de UModel* del cuadro combinado *Mostrar menús para:*.
2. Haga clic en la pestaña Comandos y arrastre los comandos hasta la barra de menús que prefiera.

Para eliminar comandos de un menú:

1. Haga clic con el botón derecho en el comando o el icono que representa el comando y
2. Seleccione **Eliminar** del menú contextual.

o

1. Seleccione **Tools | Customize** para abrir el cuadro de diálogo Personalizar y

2. Haga clic en el comando que quiera eliminar y arrástrelo hasta que aparezca una **X** bajo el cursor. Suelte el botón del ratón.

Para restaurar barras de menú:

1. Seleccione *Menú predeterminado* o *Proyecto de UModel* en la lista desplegable del cuadro combinado.
2. Haga clic en el botón **Restaurar** de debajo de la lista desplegable. Aparecerá un aviso preguntándole si está seguro de que desea restaurar la barra de menú.

Para personalizar menús contextuales (los que se abren al hacer clic con el botón derecho):

1. Seleccione el menú contextual del cuadro combinado "Seleccionar menú contextual".
2. Haga clic en la pestaña Comandos y arrastre los comandos que quiera hasta el menú contextual que se habrá abierto.

Para eliminar comandos de un menú contextual:

1. Haga clic con el botón derecho en el comando o en el icono que lo representa.
2. Seleccione **Eliminar** del menú contextual.

o

1. Seleccione **Herramientas | Personalizar**.
2. Arrastre el comando fuera del menú contextual y suelte el ratón cuando aparezca una cruz bajo el cursor.

Para restaurar los menús contextuales:

1. Seleccione el menú contextual en el cuadro combinado "Seleccionar menú contextual".
2. Haga clic en el botón **Restaurar** de debajo de la lista desplegable. Aparecerá un aviso preguntándole si está seguro de que desea restaurar el menú contextual.

Para cerrar una ventana de menú contextual:

1. Haga clic en el icono **Cerrar** en la parte superior derecha de la barra del título.
2. También puede hacer clic en el botón **Cerrar** del cuadro de diálogo "Personalizar".

Seleccione la casilla *Sombras de menú* si quiere que todos sus menús estén sombreados.

14.6.2.6 Opciones

En la pestaña *Opciones* puede configurar varias 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

 © 2018 Altova GmbH

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.

14.6.3 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.

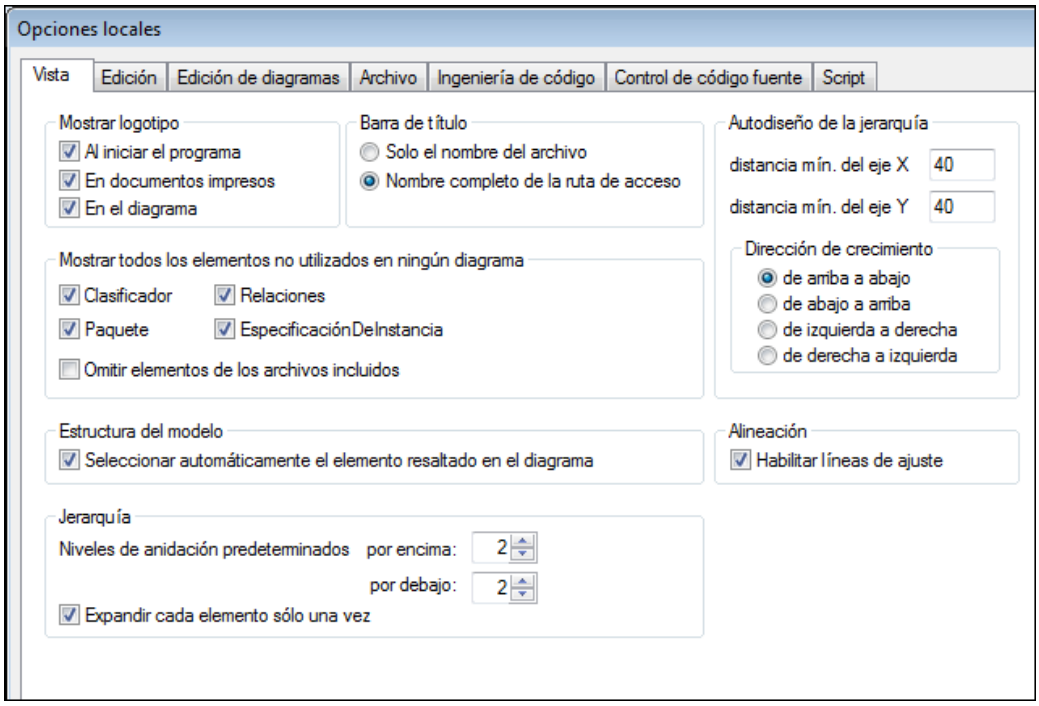
Este comando es muy práctico si movió ventanas o barras de herramientas de sitio, si las ocultó o si ajustó su tamaño y desea poner todas estas barras de herramientas y ventanas como estaban en un principio.

14.6.4 Opciones

El comando **Herramientas | Opciones** abre el cuadro de diálogo "Opciones locales".

En la **pestaña *Vista*** puede definir:

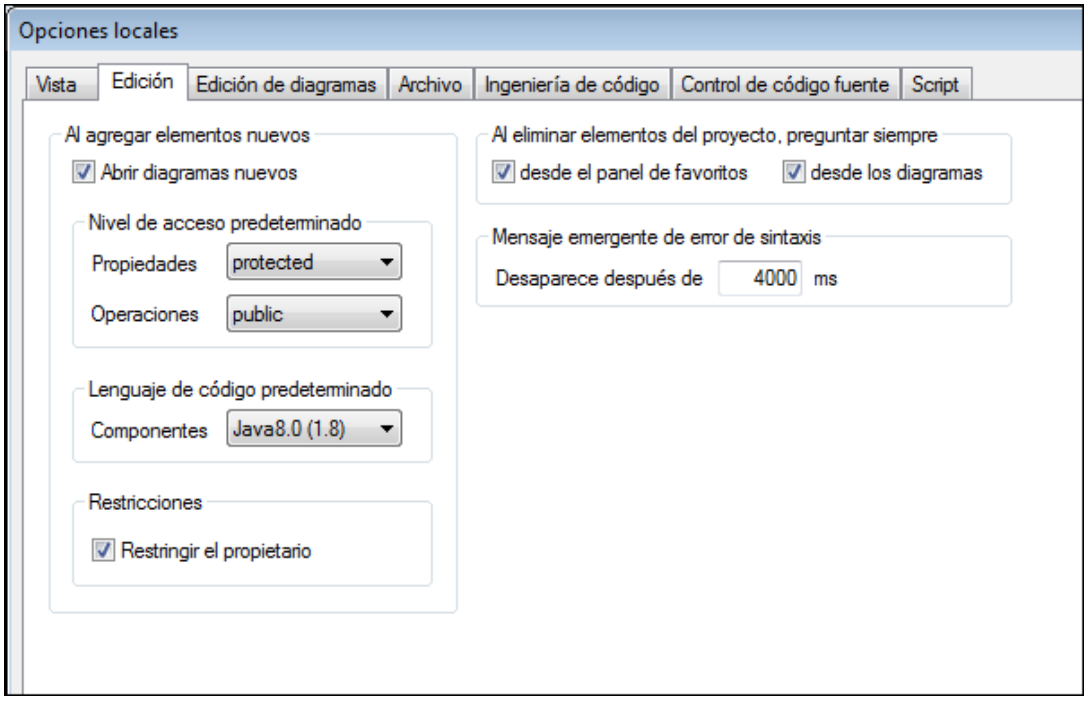
- donde debe aparecer el logotipo del programa.
- el contenido de la barra de título de la aplicación.
- el tipo de elementos que deben aparecer en la lista generada por el comando *Mostrar elementos no utilizados en ningún diagrama* del menú contextual de los paneles *Estructura del modelo* y *Favoritos*. 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:

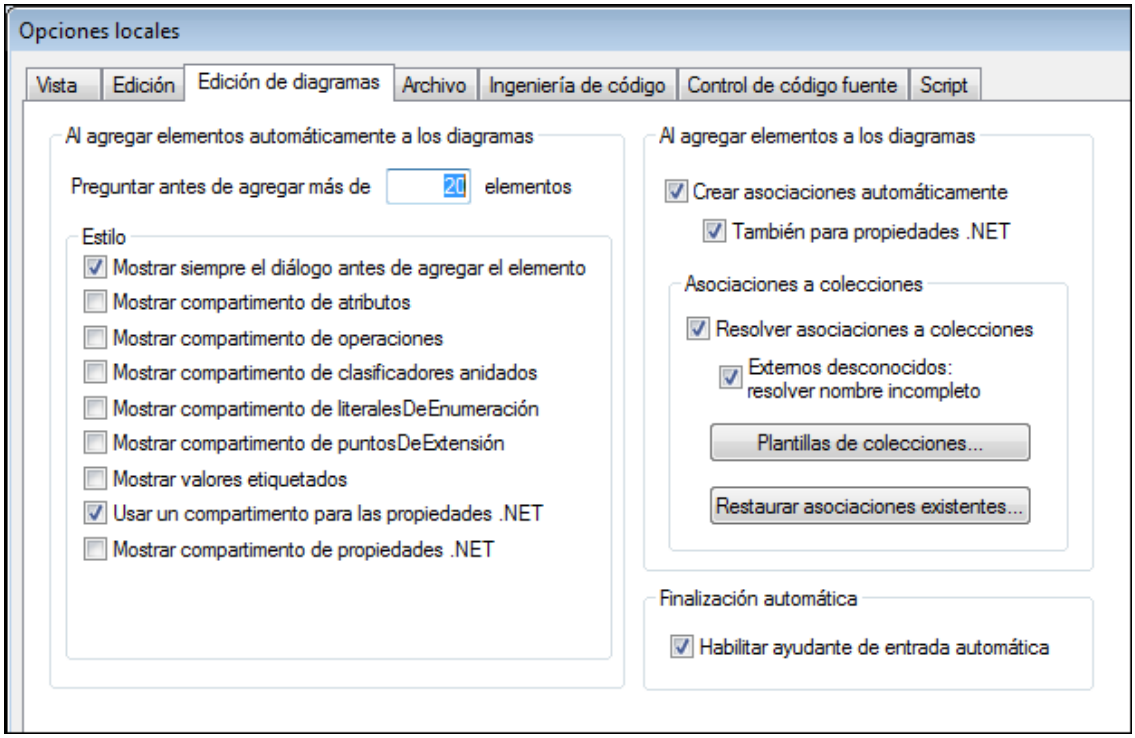
- que cuando se cree un diagrama nuevo en la *Estructura del modelo*, el diagrama se abra automáticamente en el área de trabajo.
- el nivel de acceso predeterminado de los elementos nuevos (propiedades y operaciones).
- el lenguaje de código predeterminado para los componentes nuevos.
- si las restricciones 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.

 © 2018 Altova GmbH



En la **pestaña *Edición de diagramas*** puede definir:

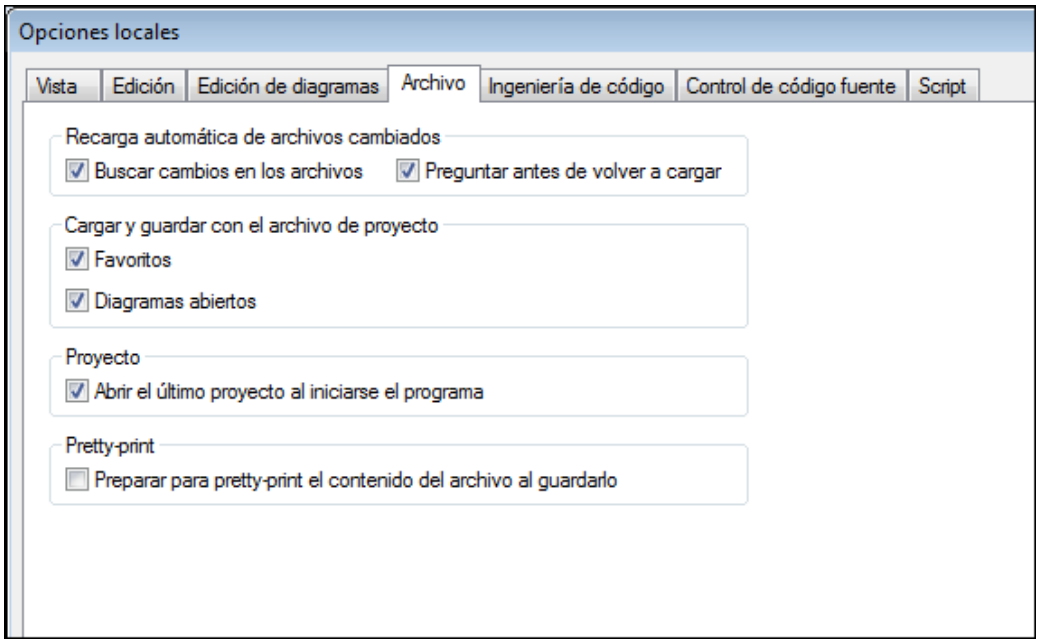
- cuántos elementos se pueden añadir automáticamente a un diagrama sin que aparezca un aviso.
- la presentación de los estilos cuando se añaden automáticamente a un diagrama.
- si se debe crear automáticamente asociaciones entre los elementos de modelado cuando se añaden elementos a un diagrama.
- si 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:

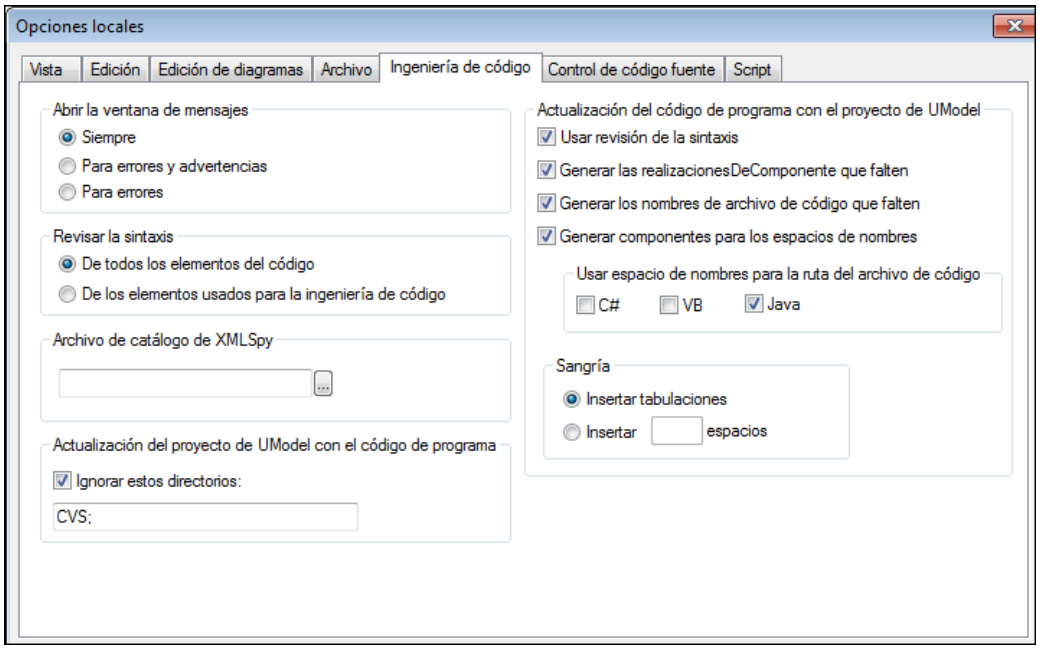
- qué ocurre cuando los archivos cambian.
- si el contenido de la ventana *Favoritos* y los diagramas abiertos deben cargarse y guardarse con el proyecto actual.
- si el último proyecto que se abrió se debe abrir automáticamente cuando se inicia la aplicación.
- si se añaden retornos de carro/saltos de línea y tabulaciones al archivo de proyecto para darle formato **pretty-print**.

© 2018 Altova GmbH



En la **pestaña *Ingeniería de código*** puede definir:

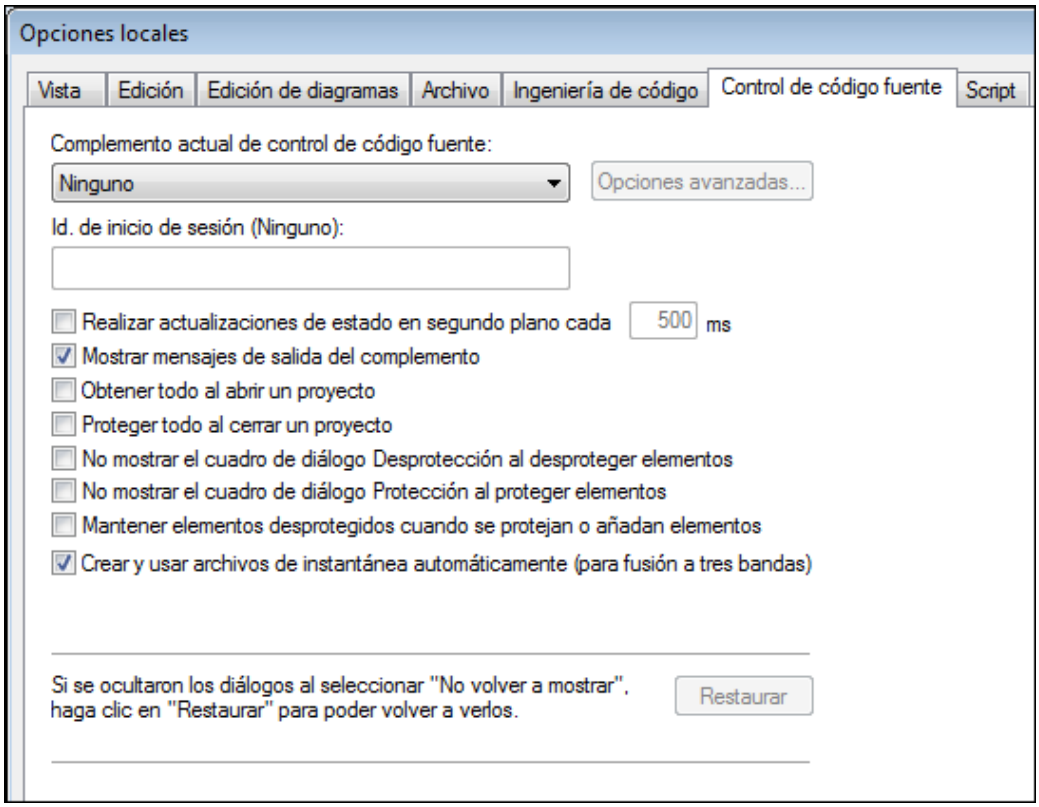
- en qué circunstancias se abre la ventana Mensajes.
- si se revisan **todos los elementos** de código (es decir, los que están en una raíz de espacio de nombres Java / C# / VB y los que están asignados a un componente Java / C# / VB) o **solo los elementos utilizados para ingeniería de código** (es 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.



En la **pestaña *Control de código fuente*** puede definir:

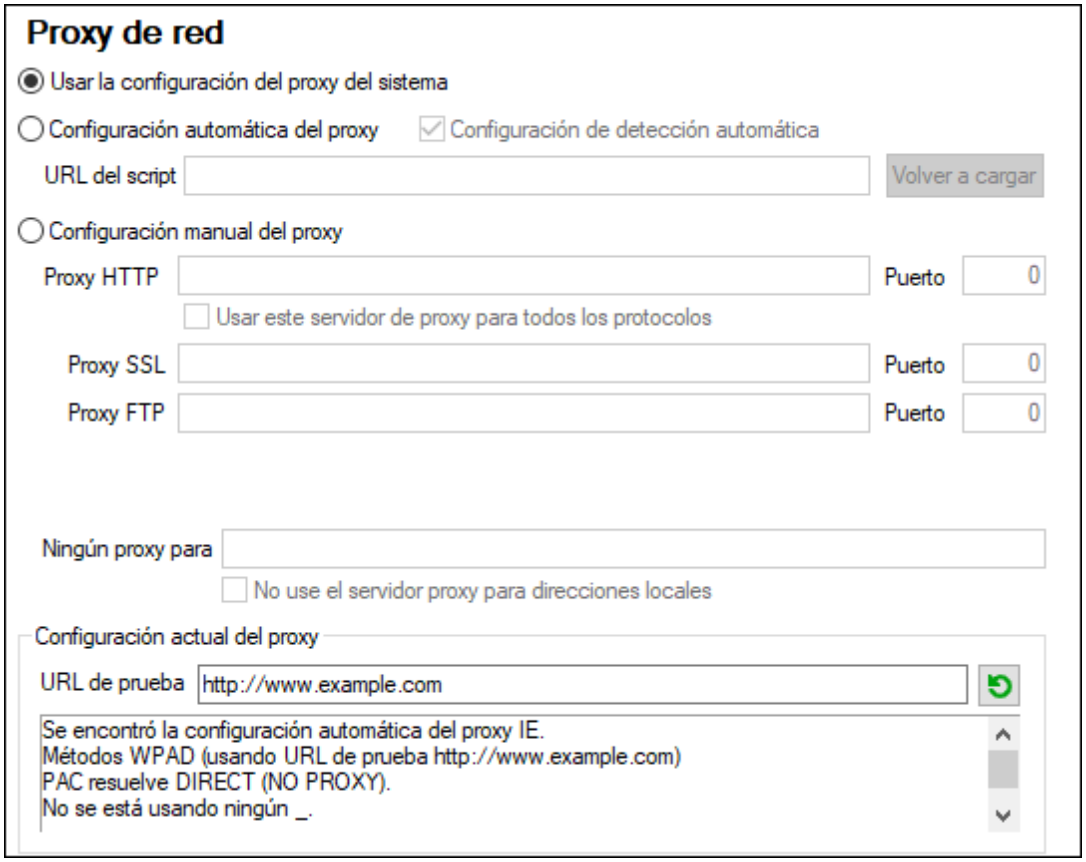
- el complemento de control de código fuente actual. Con el botón **Opciones avanzadas** puede configurar algunas opciones avanzadas del sistema de control de código seleccionado. Estas opciones avanzadas dependen del control de código fuente elegido.
- el id. de inicio de sesión para el proveedor de 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.

 © 2018 Altova GmbH



14.6.4.1 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.



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

 © 2018 Altova GmbH

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 (`.`).

Configuración actual del proxy

Proporciona un registro detallado de la detección del proxy. Se puede actualizar con el botón **Actualizar** a la derecha de *URL de prueba* (por ejemplo, al cambiar la URL de prueba o cuando se ha cambiado la configuración del proxy).

- *URL de prueba:* una URL de prueba se puede usar para ver qué proxy hay que usar para esa URL en concreto. No se trata de una URL de entrada/salida. Este campo no debe estar vacío si se ha optado por la configuración automática del proxy (seleccionando *Usar la configuración del proxy del sistema* o *Configuración automática del proxy*).

14.7 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*.

© 2018 Altova GmbH

14.8 Menú Ayuda

Contenido

Descripción

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

Descripción

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

Descripción

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

Descripción

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.

- ***Clave de evaluación 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 un código clave 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 la clave de evaluación al correo electrónico proporcionado. Ahora introduzca el código clave en el cuadro de diálogo "Activación del software" y haga clic en **Aceptar**

para empezar a trabajar con el software, que 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. Altova ofrece licencias para un solo usuario y licencias para varios usuarios (ambos tipos estarán en el correo electrónico que reciba de Altova). Las licencias para un solo usuario contienen los datos de la licencia, su nombre, el nombre de su compañía, su correo electrónico y un código clave. Las licencias para varios usuarios contienen los datos de la licencia, el nombre de su compañía y un código clave. Recuerde que el contrato de licencia prohíbe instalar más copias de las permitidas por la licencia adquirida. Recuerde introducir los datos que solicita el cuadro de diálogo de registro tal y como aparecen en el correo electrónico que recibió con las licencias.

Nota: recuerde que los datos de la licencia en el cuadro de diálogo "Activación del software" deben ser idénticos a los que aparecen en el correo electrónico que recibió con las licencias. En el caso de licencias para varios usuarios, cada usuario debe introducir su nombre en el campo *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:

- Los detalles de su licencia (nombre, compañía, correo electrónico, código clave)
- En un adjunto, un archivo de licencia con la extensión `.altova_licenses`

Para activar su producto de Altova, puede optar por una de las siguientes opciones:

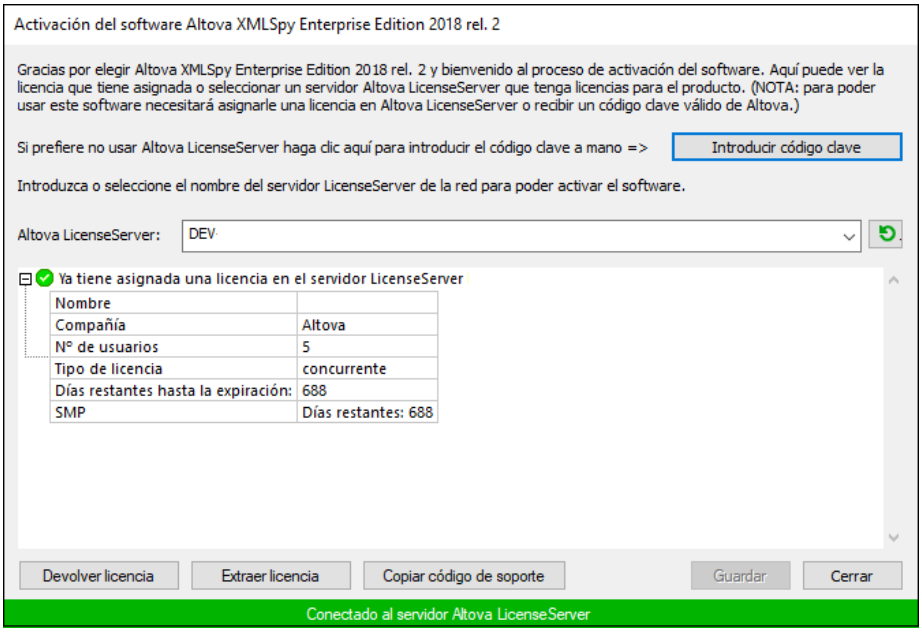
- Introducir los detalles de la licencia recibidos con el correo electrónico en el cuadro de diálogo de activación de software de su producto de Altova y hacer clic en **Activar**.
- Guardar el archivo de licencia (`.altova_licenses`) en su equipo, hacer doble clic en el archivo de licencia, introducir 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 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 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**.

 © 2018 Altova GmbH

Hay dos maneras de activar el software:

- Introduciendo la información de la licencia (con el botón **INTRODUCIR código clave...**) o
- Adquiriendo una licencia a través de un servidor Altova LicenseServer de la red (haciendo clic en el botón **Usar Altova LicenseServer**, situado al final del cuadro de diálogo). 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*) y basta con hacer 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 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. Ahora, 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.

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 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

Descripción

Hay dos maneras de comprar licencias para los productos de Altova: con el botón **Comprar código clave...** 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.

© 2018 Altova GmbH

Registro del software

Descripción

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

Descripción

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

Descripción

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

Descripción

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

Descripción

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

Descripción

Es un enlace al sitio web de Altova, donde encontrará más información sobre UModel, otros productos de Altova y tecnologías relacionadas.

Acerca de UModel

Descripción

Abre la pantalla de presentación de la aplicación, que incluye el número de versión del producto e información sobre copyright.

© 2018 Altova GmbH

Altova UModel 2018

Generador de código

15 Generador de código

© 2018 Altova GmbH

15.1 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 **...\UModelspl**. 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.

15.1.1 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 = "cadenaDePrueba"]
```
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.

© 2018 Altova GmbH

Variables predefinidas

Elemento UML	Propiedad SPL	Multi-plicidad	UML	UModel	Descripción
			Attribute / Association	Attribute / Association	
CaracterísticaDeComportamiento	isAbstract		isAbstract:Boolean		
CaracterísticaDeComportamiento	raisedException	*	raisedException:Type		
CaracterísticaDeComportamiento	ownedParameter	*	ownedParameter:Parameter		
ClasificadorConComportamiento	interfaceRealization	*	interfaceRealization:InterfaceRealization		
Clase	ownedOperation	*	ownedOperation:Operation		
Clase	nestedClassifier	*	nestedClassifier:Clasificador		
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:LiteralDeEnumeración		
Enumeración	nestedClassifier	*		nestedClassifier::Clasificador	
Enumeración	interfaceRealization	*		interfaceRealization:Interface	
LiteralDeEnumeración	ownedAttribute	*		ownedAttribut	

				e:Property	
LiteralDeEnumeración	ow nedOperation	*		ow nedOperati on:Operation	
LiteralDeEnumeración	nestedClassifier	*		nestedClassifi er:Clasificador	
Característica	isStatic		isStatic:Boolean		
Generalización	general	1	general:Clasificador		
Interfaz	ow nedAttribute	*	ow nedAttribute:Prope rty		
Interfaz	ow nedOperation	*	ow nedOperation:Oper ation		
Interfaz	nestedClassifier	*	nestedClassifier:Clasif icador		
RealizaciónDeInterfaz	contract	1	contract:Interface		
ElementoMultiplicidad	low erValue	0..1	low erValue:ValueSpe cification		
ElementoMultiplicidad	upperValue	0..1	upperValue:ValueSpe cification		
ElementoConNombre	name		name:String		
ElementoConNombre	visibility		visibility:VisibilityKind		
ElementoConNombre	isPublic			isPublic:Boolea n	visibility <public>
ElementoConNombre	isProtected			isProtected:Bo olean	visibility <protected>
ElementoConNombre	isPrivate			isPrivate:Boole an	visibility <private>
ElementoConNombre	isPackage			isPackage:Bool ean	visibility <package>
ElementoConNombre	namespacePrefix			namespacePre fix:String	XSD only - namespace prefix w hen exists
ElementoConNombre	parseableName			parseableNam e:String	CSharp, VB only - name w ith escaped keyw ords (@)
EspacioDeNombres	elementImport	*	elementImport:Importa ciónDeElemento		
Operación	ow nedReturnParameter	0..1		ow nedReturnP arameter:Para meter	parameter w ith direction return set
Operación	type	0..1		type	type of parameter w ith direction return set
Operación	ow nedOperationParameter	*		ow nedOperati onParameter:P arameter	all parameters excluding parameter w ith direction return set
Operación	implementedInterface	1		implementedInt erface:Interfac	CSharp only - the implemented

© 2018 Altova GmbH

				e	interface
Operación	ow nedOperationImplementations	*		implementedOperation:OperationImplementation	VB only - the implemented interfaces/ operations
ImplementaciónDeOperación	implementedOperationOw ner	1		implementedOperationOw ner:Interface	interface implemented by the operation
ImplementaciónDeOperación	implementedOperationName			name:String	name of the implemented operation
ImplementaciónDeOperación	implementedOperationParseableName			parseableName:String	name of the implemented operation w ith esacped keyw ords
Paquete	namespace	*		namespace:Package	packages w ith code language <<namespace> > set
ElementoEmpaquetable	ow ningPackage	0..1		ow ningPackage	set if ow ner is a package
ElementoEmpaquetable	ow ningNamespacePackage	0..1		ow ningNamespacePackage: Package	ow ning package w ith 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	ownedAttribute	*	ownedAttribute:Property		
EnlaceDePlantilla	signature	1	signature:TemplateSignature		
EnlaceDePlantilla	parameterSubstitution	*	parameterSubstitution:SustituciónDeParámetroDePlantilla		
ParámetroDePlantilla	paramDefault			paramDefault:String	template parameter default value
ParámetroDePlantilla	ownedParameteredElement	1	ownedParameteredElement:ParameterableElement		
SustituciónDeParámetroDePlantilla	parameterSubstitution			parameterSubstitution:String	Java only - code wildcard handling
SustituciónDeParámetroDePlantilla	parameterDimensionCount			parameterDimensionCount:Integer	code dimension count of the actual parameter
SustituciónDeParámetroDePlantilla	actual	1	OwnedActual:ParameterableElement		
SustituciónDeParámetroDePlantilla	formal	1	formal:TemplateParameter		
FirmaDePlantilla	template	1	template:TemplateableElement		
FirmaDePlantilla	ownedParameter	*	ownedParameter:TemplateParameter		
ElementoQueSePuedeConvertirEnPlantillas	isTemplate			isTemplate:Boolean	true if template signature set
ElementoQueSePuedeConvertirEnPlantillas	ownedTemplateSignature	0..1	ownedTemplateSignature:FirmaDePlantilla		
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.

© 2018 Altova GmbH

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.

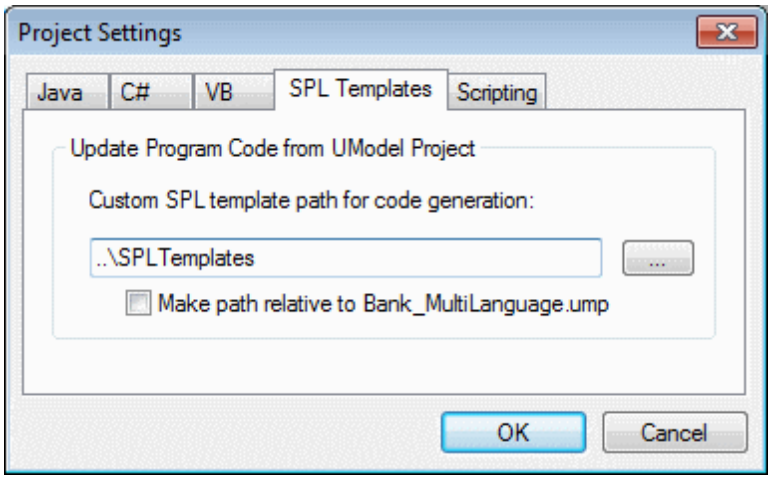
Por ejemplo, busque `$Property.name` y reemplácela con `"m_" & $Property.name`

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).

 © 2018 Altova GmbH

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.

15.1.2 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

15.1.3 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`.

© 2018 Altova GmbH

15.1.4 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).

15.1.5 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)

15.1.5.1 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.

 © 2018 Altova GmbH

- **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
]
```

15.1.5.2 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")
```


© 2018 Altova GmbH

15.2 Códigos de error

Códigos de error del sistema operativo

201 No se encontró el archivo: "%s"
202 No se puede crear el archivo "%s"
203 No se puede abrir el archivo "%s"
204 No se puede copiar el archivo "%s" en "%s"

Códigos de error de sintaxis

401 Se esperaba una palabra clave
402 Se esperaba "%s"
403 No se especificó el archivo de salida
404 Fin inesperado del archivo
405 Palabra clave no permitida

Códigos de error de tiempo de ejecución

501 Variable desconocida "%s"
502 Redefinición de la variable "%s"
503 La variable "%s" no es un contenedor
504 Propiedad desconocida "%s"
505 No se puede convertir %s en %s
507 Función desconocida
508 Ya se definió esta función
509 Parámetro no válido
510 División por cero
511 Método desconocido
512 Número incorrecto de parámetros
513 Desbordamiento de pila

Altova UModel 2018

Información sobre licencias

16 Información sobre licencias

En esta sección encontrará:

- información sobre la distribución de este producto de software
- información sobre activación del software y medición de licencias
- información sobre derechos de propiedad intelectual de este producto de software
- 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.

© 2018 Altova GmbH

16.1 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 antes de decidir si compra el producto.
- 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 totalmente integrado. 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 y aceptar el contrato de licencia de software de Altova, que se entrega en forma de código clave. La licencia puede comprarse directamente en la tienda en línea del sitio web de Altova. Después de comprar la licencia recibirá el código clave, que debe introducir en el cuadro de diálogo "Activación del software" para desbloquear el producto de forma permanente.

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.

Para más información consulte el contrato de licencia de software de Altova que aparece al final de esta sección.

16.2 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.

También notará que su producto de Altova ofrece varias funciones prácticas si está conectado a Internet. Estas funciones no tienen nada que ver con la tecnología de medición de licencias.

© 2018 Altova GmbH

16.3 Derechos de propiedad intelectual

El software de Altova y sus copias (si tiene permiso de Altova para realizar copias) es propiedad intelectual de Altova y de sus proveedores. La estructura, la organización y el código del software se considera secreto comercial e información confidencial de Altova y de sus proveedores. El software está protegido por las leyes de derechos de autor, como la ley de derechos de autor de EE UU, tratados internacionales y la legislación vigente del país donde se utiliza, entre otras. Altova conserva los derechos de propiedad de todas las patentes, derechos de autor, secretos comerciales, marcas registradas y otros derechos de propiedad intelectual pertenecientes al software y los derechos de propiedad de Altova abarcan también imágenes, fotografías, animaciones, vídeos, audio, música, texto y otros applets incorporados al software y al material impreso que viene con el software. Las notificaciones de infracción de dichos derechos de autor debe enviarse al agente de derechos de autor de Altova, cuyos datos de contacto aparecen en el sitio web de Altova.

El software de Altova contiene software de terceros que también está protegido por las leyes de propiedad intelectual, incluida, entre otras, la legislación de derechos de autor mencionada en http://www.altova.com/es/legal_3rdparty.html.

Los demás nombres y marcas registradas son propiedad de sus respectivos propietarios.

© 2018 Altova GmbH

16.4 Contrato de licencia para el usuario final

- Altova End User License Agreement: http://www.altova.com/eula
- Altova Privacy Policy: http://www.altova.com/privacy

Índice

.

.NET Framework,
incluir archivo, 153

A

Abrir,
URL, 481
Abrir proyecto,
control de código fuente, 432
Absolutos,
vínculos absolutos y relativos, 262
Abstracta,
clase, 25
Acceso directo,
asignar/eliminar, 497
mostrar en información rápida, 500
tecla, 497
teclado, 497
Acelerar,
rendimiento, 158
Activar/desactivar,
modo compacto, 399
Actividad, 293
agregar a estado, 293
agregar operación, 293
iconos, 461
Actividades,
agregar diagrama de actividades a transición, 293
diagrama de, 275
Actor,
definido por el usuario, 14
personalizar, 14
Actualizar,
archivo de proyectos, 193
Actualizar estado,
control de código fuente, 451
Advertencias,
durante ingeniería de código, 90
Agregar, 14, 443
al control de código fuente, 443
diagrama a un paquete, 14
paquete a un proyecto, 14
proyecto al control de código fuente, 443
proyecto nuevo, 146
Ajuste,
líneas de ajuste mientras se arrastran objetos, 501
Alinear,
elementos mientras se arrastran, 14
Anotación,
esquema XML, 399
Aplicación,
externa (argumentos), 494
Archivo, 481
abrir desde URL, 481
combinar archivos de proyecto, 250
ejemplo del tutorial, 10
nuevo / cargar / guardar con archivo de procesamiento por lotes, 102
ump, 146
Archivos binarios,
importar C# y Java, 187
Archivos de proyecto,
Borland - MS Visual Studio .Net, 485
Archivos locales,
vínculos absolutos o relativos, 262
Argumentos,
herramientas externas, 494
Artefacto,
agregar al nodo, 52
manifestación, 52
Asignar,
acceso directo a un comando, 497
estereotipo, 386
Asociación, 25
agregado/compuesto, 25
asociaciones reflexivas, 138
cambiar las propiedades de, 138
caso de uso, 14
como relación, 133
crear, 133, 138
entre clases, 25
ver, 138
ver propiedad con tipo, 259
vínculos entre objetos, 39
Asociación de colecciones,
crear, 141

Asociación de colecciones,
requisitos previos, 141
resolver a plantillas de colecciones, 141
Asociaciones,
ver, 86
Atributo,
estereotipo, 386
mostrar / ocultar, 361
mostrar / ocultar valores etiquetados, 386
ventana de finalización automática, 501
Attribute,
coloring, 367
Automática,
respuesta automática a mensajes, 335
Automáticamente,
agregar operación, 293
Ayuda, 511

B

Ball and socket,
notación de forma esférica, 361
Barra de herramientas, 493
activar/desactivar, 493
agregar comando, 491
crear nueva, 493
mostrar iconos de tamaño grande, 500
restaurar comandos, 493
Barras de herramientas,
restaurar, 491
Base,
clase, 34
Binarios,
compatibilidad con binarios confusos, 187
Borland,
archivo de proyecto BSDJ, 485
BSDJ,
proyecto Borland, 485
Buscar, 483
diagramas, 112
elementos, 112
elementos de modelado, 483
pestañas de búsqueda, 76
texto, 112
y reemplazar, 483
Búsqueda,
diagramas, 112
elementos, 112
texto, 112

C

C#,
código, 518
generar código, 159
importar archivo binario, 187
importar código fuente, 176
opciones de generación de código, 164
opciones de importación de código, 178
C++,
código, 518
Calificador de asociación,
crear, 138
CallBehavior,
insertar, 276
CallOperation,
insertar, 276
Cambiar de nombre,
clasificador, 194
Cambiar de proveedor,
control de código fuente, 451
Cambiar nombres de clases,
efecto en el nombre del archivo de código, 196
Cambio de estado,
definir en una línea de tiempo, 353
Carpeta,
carpeta de ejemplos, 10
Carpetas,
obtener carpetas en control de código fuente, 438
Casilla de activación,
especificación de ejecución, 329
Caso de uso, 14
agregar, 14
asociación, 14
compartimento, 14
multilínea, 14
Casos de uso,
diagrama de, 317
iconos, 476
Catálogo,
archivo de catálogo, 501
Ciclo de vida,

© 2018 Altova GmbH

Ciclo de vida,
diagrama de, 352
iconos, 475
Clase, 361
en diagramas de componentes, 46
expandir y contraer compartimentos, 361
iconos, 463
interfaz con ball and socket (notación de forma esférica), 361
operación de clase (invalidar), 361
varias instancias en el diagrama, 361
Clase base,
invalidar, 361
Clases,
abstractas y concretas, 25
agregar, 25
agregar operaciones, 25
agregar propiedades, 25
asociaciones, 25
base, 34
cambios de nombre (sincronización), 196
derivadas, 34
diagrama de, 361
diagramas, 25
habilitar ventana de finalización automática, 501
sincronización, 193
Clasificador,
cambiar de nombre, 194
nuevo, 194
restringir, 256
Class,
syntax coloring, 367
Código, 196
agregar código a diagrama de secuencia, 349
código Java y nombres de archivos de clases, 196
generar código a partir de diagramas de secuencia, 345
generar diagrama de secuencia a partir de código, 340
generar varios diagramas de secuencia a partir de código, 343
predeterminado, 501
refactorizar, 196
sincronización, 193
SPL, 519
Código de refactorización,
nombres de clases (sincronizar), 196
Colaboración,
diagrama de estructura de un compuesto, 376
Color,
syntax coloring - enable/disable, 367
Color de fondo,
transparente, 394
Comando,
agregar a barra de herramientas / menú, 491
eliminar de menú, 499
menú contextual, 499
procesamiento de la línea de comandos, 98
restaurar menú, 499
Comandos,
línea de comandos: nuevo / cargar / guardar, 102
procesamiento de la línea de comandos, 102
CombinaciónDePaquete, 381
ver, 86
Combinar,
código con el modelo, 485
modelo con el código, 485
omitir directorio, 501
proyectos, 250
Compacto,
modo (activar/desactivar), 399
Comparar archivos de código fuente, 449
Compartimento,
expandir uno / varios, 361
Compartir,
desde el control de código fuente, 446
paquete y diagrama, 155
Compatibilidad,
actualizar proyectos, 193
Componentes, 46
diagrama, 46
diagrama de, 378
iconos, 466
insertar clase, 46
realización, 46
Comportamiento,
diagramas de, 275
Composición,
crear asociación, 25
Comunicación,
diagrama de, 318
iconos, 464
Concreta,
clase, 25
Configuración,
control de código fuente, 501
Configurar,
sincronización, 193
Confusos,
compatibilidad con binarios confusos, 187

© 2018 Altova GmbH

Contención,
dibujar en un diagrama, 144
Contraer,
compartimentos de clase, 361
Contrato de licencia para el usuario final, 536, 541
Control de código fuente, 428
abrir proyecto, 432
actualizar estado, 451
agregar al control de código fuente, 443
anular desprotección, 441
cambiar de proveedor, 451
comandos, 432
desproteger, 439
ejecutar interfaz nativa, 451
habilitar / deshabilitar, 435
mostrar diferencias, 449
mostrar historial, 447
obtener archivo, 436
obtener la versión más reciente, 436
opciones / configuración, 501
propiedades, 450
proteger, 440
quitar del, 445
Control de versiones,
comandos, 432
subproyectos y trabajo en equipo, 418
Convergencia,
crear en diagrama de actividades, 278
Correo electrónico,
enviar proyecto, 481
CPU,
carga - acelerar actualización de estado en segundo plano, 428
CR/LF,
en el archivo ump al guardarlo, 146
Crear,
esquema XML, 409
métodos getter / setter, 361
CSPROJ - CSDPROJ,
MS Visual Studio .Net, 485
Cuadrícula,
líneas de ajuste, 501
líneas de ajuste mientras se arrastran objetos, 14
CVS, 428

D

Definidas por el usuario,
plantillas SPL, 193
Definido por el usuario,
actor, 14
Definidos por el usuario,
estilos de estereotipo, 392
Dependencia,
incluir, 14
uso, 46
Dependencias,
ver, 86
Derivada,
clase, 34
Descargar proyecto de control de código fuente, 432
Deshabilitar el control de código fuente, 435
Deshacer desprotección, 441
Desproteger, 439
Diagram,
de perfil, 385
Diagrama, 361, 501
agregar a Favoritos, 83
agregar actividad a transición, 293
agregar código a diagrama de secuencia, 349
compartir paquete y diagrama, 155
de actividades, 275
de casos de uso, 317
de ciclo de vida, 352
de clases, 361
de componentes, 378
de comunicación, 318
de esquema XML, 398
de esquema XML (importación), 399
de estructura de un compuesto, 376
de implementación, 378
de máquina de estados, 291
de objetos, 379
de paquetes, 379
de perfil y estereotipos, 386
de secuencia, 326
desplazamiento rápido, 88
encontrar elementos no usados, 113
estilos, 85
generar código a partir de diagramas de secuencia, 345

© 2018 Altova GmbH

Diagrama, 361, 501
generar diagrama de dependencias entre paquetes, 379
global de interacción, 322
guardar como PNG, 481
guardar elementos como mapa de bits, 483
guardar los diagramas abiertos con el proyecto, 501
iconos, 460
insertar elementos en, 108
omitir elementos de archivos incluidos, 501
relación de iconos, 78
varias instancias de la clase, 361
ver un resumen de, 88
Diagrama de actividades,
crear rama / convergencia, 278
elementos, 281
insertar elementos, 276
Diagrama de ciclo de vida,
cambiar de un tipo a otro, 353
ciclo de vida, 353
evento/estímulo, 357
insertar elementos, 353
línea de vida, 353
marca de graduación, 356
mensaje, 359
restricciónDeDuración, 358
restricciónDeTiempo, 359
valor general de la línea de vida, 353
Diagrama de comunicación,
generar a partir de un diagrama de secuencia, 319
Diagrama de paquetes,
insertar elementos, 381
Diagrama de secuencia, 340
agregar código a, 349
fragmento combinado, 330
generar a partir de código, 340
generar a partir de un diagrama de comunicación, 319
generar código a partir de, 345
generar diagrama de secuencia a partir de métodos getter/setter, 343
generar varios diagramas de secuencia a partir de código, 343
insertar elementos, 327
invariante de estado, 335
línea de vida, 329
mensajes, 335
nombres de operación que se deben omitir, 340
puerta, 334
uso de interacción, 334
Diagrama global de interacción,
insertar elementos, 323
Diagramas, 274
abrir, 125
acercar/alejar, 132
ajustar a la ventana, 132
borrar del proyecto, 126
cambiar el tamaño de, 127
cambiar la apariencia de, 127
crear, 92, 122
de comportamiento, 275
de estructura, 361
generar, 123
generar desde Panel Jerarquía, 86
ver dentro del proyecto, 82
Directorio,
cambiar ubicación del proyecto, 146
carpeta de ejemplos, 10
omitir durante la combinación, 501
Directorio de trabajo,
control de código fuente, 432
Diseño, 488
Disparador,
definir disparador de transición, 293
Distribución,
de productos de software de Altova, 536, 537, 540
División,
evitar división entre las páginas, 481
Documentación, 262
añadir a elementos, 118
generar código fuente con, 118
generar proyecto UML, 262
oimportar desde código fuente, 118
vínculos relativos, 262

E

Edición, 483
Ejecutar interfaz nativa, 451
Ejemplos,
carpeta del tutorial, 10
Elemento,
agregar a Favoritos, 83
cambiar propiedades de, 84
estilos, 85
generar documentación, 262
guardar elemento seleccionado como mapa de bits, 483

© 2018 Altova GmbH

Elementos,
agregar a diagrama, 108
agregar al modelo, 107
alinear con un diagra,a, 128
añadir a diagrama, 108
añadir al modelo, 78, 107
aplicar imágenes personalizadas a, 120
borrar del diagrama, 111
borrar del proyecto, 111
buscar, 112
cambiar el tamaño, 128
cambiar la apariencia de, 120
copiar, 110
diseño automático, 128
documentar, 89, 118
encontrar en un diagrama, 113
hipervínculos, 116
insertar en diagrama de máquina de estados, 292
mover, 110
omitir elementos de archivos incluidos, 501
reemplazar, 112
renombrar, 110
restringir, 114
Elimina,
acceso directo, 497
Eliminar, 491
barra de herramientas, 493
comando de menú contextual, 499
comando de una barra de herramientas, 491
icono de una barra de herramientas, 491
Enlace,
plantilla, 258
Enumeración,
valor predeterminado, 390
y estereotipos, 390
Enviar por correo electrónico,
proyecto, 481
Errores,
durante ingeniería de código, 90
Especializar,
generalizar, 34
Especificación de ejecución,
línea de vida, 329
Esquema,
crear esquema XML, 409
generador de código, 518
tipo de datos (definir), 406
XML, 398
XML (importación), 399
Esquema XML,
anotación, 399
crear/generar, 409
diagrama de, 398
iconos, 477
insertar elementos, 406
modelo de contenido, 406
Estado, 293
agregar actividad, 293
definir transición entre, 293
insertar estado simple, 293
ortogonal, 299
submáquina, 299
Estado compuesto, 299
agregar región, 299
Estado de submáquina,
agregar punto de entrada/salida, 299
Estereotipo,
agregar a diagrama de perfil, 386
asignar, 386
atributos (definir), 386
estilos definidos por el usuario, 392
icono personalizado, 394
memberEnd, 386
perfiles, 386
valor etiquetado predeterminado, 390
y enumeración, 390
Estereotipos,
añadir al panel Propiedades, 84
definición, 385
Estilos,
aplicar a diagramas, 127
aplicar a elementos, 120
aplicar a líneas, 135
elemento precedente, 120, 127
en cascada, 120, 127, 135
estereotipo definido por el usuario, 392
precedencia, 135
Estructura,
diagramas de, 361
Estructura de un compuesto,
diagrama de, 376
iconos, 465
insertar elementos, 376
Etiquetado,
valor (predeterminado), 390
Etiquetados,

© 2018 Altova GmbH

Etiquetados,
valores, 386
valores (definición), 385
valores (ocultar/mostrar atributos), 386
Etiquetas,
identificadores ID y UUID, 414
Evento/estímulo,
diagrama de ciclo de vida, 357
Excepción,
agregar excepción generada, 361
Excepciones generadas,
agregar, 361
Expandir,
todos los compartimentos de clase, 361
Exportar,
como XMI, 414
Extensión,
XMI, 414

F

Favoritos,
agregar a, 83
eliminar de, 83
panel, 83
Finalización automática,
función, 25
ventana para edición de clases, 501
Finalizaciópn automática de tipos de datos,
desencadenar, 130
deshabilitar, 130
Firma,
plantilla, 256, 257
Flujo de trabajo,
proyecto, 146
Formato,
estereotipo definido por el usuario, 392
Fragmento combinado, 330
Fusión,
a 3 bandas, 251
a 3 bandas manual, 253

G

Generador de código, 518
Generalización,
como relación, 108, 133
crear, 133
Generalizaciones,
ver, 86
Generalizar,
especializar, 34
Generar,
código a partir de un esquema, 518
diagrama de secuencia a partir de código, 340
diagrama de secuencia a partir de diagrama de comunicación, 319
documentación del proyecto UML, 262
esquema XML, 409
RealizacionesDeComponente automáticamente, 194
respuesta a mensajes automáticamente, 335
varios diagramas de secuencia a partir de código, 343
Get,
métodos getter / setter, 361
Getter / Setter,
generar diagrama de secuencia a partir de métodos getter/setter, 343
Guardar,
archivos de subproyectos, 420
diagrama como imagen, 481
elementos como mapas de bits, 483

H

Habilitar,
líneas de ajuste mientras se arrastran objetos, 501
Habilitar el control de código fuente, 435
Herramientas, 491
agregar al menú Herramientas, 494
opciones, 501
Hipervínculos,
en el texto de la documentación, 118
Historial,
mostrar, 447

© 2018 Altova GmbH

I

Icono,
actividad, 461
agregar a barra de herramientas / menú, 491
caso de uso, 476
ciclo de vida, 475
clase, 463
componentes, 466
comunicación, 464
esquema XML, 477
estructura de un compuesto, 465
icono de estereotipo personalizado, 394
implementación, 467
interacción global, 468
máquina de estados, 474
mostrar iconos de tamaño grande, 500
objeto, 469
paquete, 470
secuencia, 473
Iconos,
visibilidad, 361
Iconos de los diagramas de UModel, 460
ID,
identificadores ID y UUID, 414
Imágenes,
usar como fondo de elemento, 120
Implementación,
diagrama, 52
diagrama de, 378
iconos, 467
ImportaciónDeElemento,
ver, 86
ImportaciónDePaquete, 381
ver, 86
Importar, 414
archivo XMI, 414
archivos binarios, 187
esquema XML, 399
XMI generado con UModel, 414
Imprimir,
vista previa, 481
Incluir, 153, 155
.NET Framework, 153
cambiar estado, 155
compartir paquete y diagrama, 155
dependencia, 14
proyecto de UModel, 153
subproyectos en el proyecto principal, 420
Información legal, 536
Información rápida, 500
mostrar accesos directos en, 500
ver, 500
Información sobre derechos de autor, 536
Ingeniería de código,
advertencias, 90
del código al modelo, 67
del modelo al código, 57
errores, 90
generar RealizacionesDeComponente, 194
mensajes de informaciçon, 90
mover archivo de proyecto a una ubicación nueva, 146
resolver asociaciones, 141
Ingeniería directa, 57
Ingeniería inversa, 67
Iniciar,
con el proyecto anterior, 501
UModel, 11
Insertar, 276
acción (CallBehavior), 276
acción (CallOperation), 276
elementos en diagrama de paquetes, 381
elementos en diagrama global de interacción, 323
elementos en diagramas de ciclo de vida, 353
elementos en estructura de un compuesto, 376
estado simple, 293
Instalación,
carpeta de ejemplos, 10
Instalador,
multiusuario, 10
Instancia,
diagrama, 39
objeto, 39
varias clases, 361
Integrar,
subproyectos en el proyecto principal, 420
Inteligente,
finalización automática, 25
Interacción,
diagrama global de, 322
Interacción global,
iconos, 468
Interfaz, 361

© 2018 Altova GmbH

Interfaz, 361
ball and socket (notación de forma esférica), 361
implementar, 361
Interfaz del usuario, 76
Introducción, 6
Invalidar,
clase base, 361
operaciones de clase, 361
plantillas SPL predeterminadas, 193
Invariante de estado, 335
Ir a,
línea de vida, 329

J

Java,
código, 518
código y nombres de archivos de clases, 196
generar código, 159
importar archivo binario, 187
importar código fuente, 176
opciones de generación de código, 164
opciones de importación de código, 178
Jerarquías (diagrama),
niveles en la documentación, 262

L

Licencia, 541
información sobre, 536
Licencia del producto de software, 541
Línea,
ortogonal, 46
Línea de tiempo,
definir cambios de estado, 353
Línea de vida,
atributos, 329
ir a, 329
propiedad de tipo, 329
valor general, 353
Línea nueva,
en línea de vida, 319
operandoDeInteracción, 330
Líneas,
cambiar el estilo de, 135
de ajuste, 501
directa, 135
formato, 39
ortogonal, 135
personalizada, 135
Líneas de ajuste, 14
Llamada,
mensaje, 335
Llamada del mensaje,
operación ir a, 335

M

Manifestación,
artefacto, 52
Mapa de bits,
guardar elementos como, 483
Máquina de estados,
diagrama de, 291
elementos, 310
estados compuestos, regiones, 299
estados, actividades, transiciones, 293
iconos, 474
insertar elementos, 292
Marca de graduación,
diagrama de ciclo de vida, 356
Medición de licencias,
en los productos de Altova, 538
Mejorar,
rendimiento, 158
Member end,
estereotipo, 386
Mensaje, 335
crear objeto, 335
diagrama de ciclo de vida, 359
flechas, 335
insertar, 335
llamada, 335
mover, 335
numeración, 335
operación ir a, 335
Menú, 499
agregar / eliminar comando, 491
agregar menú a, 494
archivo, 481

© 2018 Altova GmbH

Menú, 499
ayuda, 511
diseño, 488
edición, 483
eliminar comandos de, 499
herramientas, 491
personalizar, 499
predeterminado/XMLSPY, 499
proyecto, 485
ventanas, 510
vista, 490
Menú contextual,
comandos, 499
Metadatos,
salida XMI, 414
Método,
agregar excepción agregada, 361
generar diagrama de secuencia a partir de métodos, 340
generar diagrama de secuencia a partir de métodos getter/setter, 343
generar varios diagramas de secuencia a partir de métodos, 343
Métodos,
getter / setter, 361
MisDocumentos,
archivos de ejemplo, 10
Modelado,
mejorar el rendimiento, 158
Modelo,
agregar elementos a, 107
añadir elementos a, 78, 107
cambiar el nombre de las clases (efecto en Java), 196
Modelo de contenido,
del esquema XML, 406
Mostrar,
ocultar slot, 361
ocultar valores etiquetados / atributos, 386
valores etiquetados, 399
Mostrar / ocultar,
atributos, operaciones, 361
Mostrar diferencias, 449
Mostrar historial, 447
Mover,
proyecto, 146
Mover las flechas de los mensajes, 335
MS Visual Source Safe, 428
MS Visual Studio .Net,
archivo de proyecto CSPROJ - CSDPROJ, 485
Multilínea, 14
caso de uso, 14
operandoDeInteracción, 330
texto del actor, 14
Multiusuario,
carpeta de ejemplos, 10

N

Nodo, 52
agregar, 52
agregar artefacto, 52
estilos, 85
Nombre,
de región (ver/ocultar), 299
Nuevo,
clasificador, 194
Numeración,
mensajes, 335

O

Objeto,
crear mensaje, 335
iconos, 469
vínculos (asociaciones), 39
Objetos,
diagrama, 39
diagrama de, 379
Obtener archivo,
control de código fuente, 436
Obtener carpetas,
control de código fuente, 438
Obtener la versión más reciente, 436
Ocultar,
mostrar slot, 361
valores etiquetados, atributos, 386
Omitir, 501
directorios, 501
elementos en la lista, 501
nombres de operación, 340
Opciones, 501
control de código fuente, 501
herramientas, 501

© 2018 Altova GmbH

Operación, 361
agregar automáticamente en actividad, 293
invalidar, 361
ir a (en la llamada del mensaje), 335
mostrar / ocultar, 361
omitir en la generación de diagramas de secuencia, 340
plantilla, 259
ventana de finalización automática, 501
volver a utilizar, 34
Operaciones,
agregar, 25
Operador,
interacción, 330
Operador de interacción,
definir, 330
Operando,
interacción, 330
Operando de interacción,
multilínea, 330
Operation,
coloring, 367
Ortogonal,
estado, 299
línea, 46
Ortografía,
corrector, 89

P

Página,
evitar división entre las páginas, 481
Panel Árbol de diagramas, 82
Panel Documentación, 89
Panel Estilos, 85
Panel Estructura del modelo,
expandir o contraer elementos, 78
explorar el proyecto desde, 78
mostrar u ocultar elementos, 78
ordenar elementos, 78
relación de iconos, 78
Panel Jerarquía, 86
Panel Mensajes,
referencia, 90
Panel Propiedades,
añadir propiedades personalizadas, 84
Panel Vista general,
desplazarse, 88
Paquete,
compartir, 155
iconos, 470
paquetes predeterminados, 78
perfil, 386
relación de iconos, 78
Paquetes,
diagrama de, 379
generar diagrama de dependencias entre, 379
Parámetro,
de procesamiento por lotes, 98
plantilla, 259
Parcial,
generar documentación parcial, 262
Perfil, 385
aplicación, 385
diagrama, 385
diagrama de, 385, 386
estereotipos, 386
Período de evaluación,
de los productos de software de Altova, 536, 537, 540
Personalizado,
icono de estereotipo, 394
Personalizar, 491
actor, 14
comandos de las barras de herramientas / menús, 491
menú, 499
menú contextual, 499
Pestañas de búsqueda, 76
Plantilla,
enlace, 258
firma, 256, 257
operación/parámetro, 259
Plantillas,
plantillas SPL, 519
SPL definidas por el usuario, 193
Plantillas SPL,
ruta de acceso, 519
PNG,
guardar diagrama, 481
Por lotes, 102
modo de procesamiento por lotes, 102
nuevo / cargar / guardar, 102
procesamiento, 98, 102
Predeterminadas,
plantillas SPL, 193
Predeterminado,

© 2018 Altova GmbH

Predeterminado,
código de proyecto, 501
menú, 499
valor etiquetado, 386
Pretty print,
preparar proyecto para pretty print al guardarlo, 146
salida XMI, 414
Procesamiento por lotes,
modo de procesamiento por lotes, 102
Property,
coloring, 367
Propiedad,
con tipo (ver), 259
de tipo línea de vida, 329
volver a utilizar, 34
Propiedades,
agregar, 25
control de código fuente, 450
Proteger, 440
Proveedor,
de control de código fuente, 428
seleccionar, 432
Proyecto, 485, 501
abrir último proyecto al iniciar UModel, 501
actualizar archivo de proyecto, 193
agregarlo al control de código fuente, 443
añadir o eliminar elementos, 78
código predeterminado, 501
combinar, 250
crear, 146
crear subproyectos, 420
enviar por correo electrónico, 481
estilos, 85
explorar, 78
flujo de trabajo, 146
fusión a 3 bandas, 251
fusión a 3 bandas manual, 253
generar documentación, 262
guardar los diagramas abiertos, 501
guardar para pretty print, 146
incluir proyecto de UModel, 153
insertar paquete, 146
mover, 146
quitarlo del control de código fuente, 445
revisión de la sintaxis, 485
Proyecto local, 432
Puerta,
diagrama de secuencia, 334
Punto de entrada,
agregar a submáquina, 299
Punto de salida,
agregar a submáquina, 299
PVCS Version Manager, 428

Q

Quitar,
del control de código fuente, 445

R

Raíz,
catálogo, 501
como paquete, 110
sincronizar raíz / paquetes / clases, 193
Rama,
crear en diagrama de actividades, 278
Realización,
componente, 46
Realizaciones,
generar RealizacionesDeComponente, 194
RealizacionesDeComponente,
generación automática, 194
Rechazar cambios, 441
Recuperar archivo,
control de código fuente, 436
Recursos,
acelerar actualización de estado en segundo plano, 428
Referencia, 480
Región,
agregar a estado compuesto, 299
nombre de región (ver/ocultar), 299
Relaciones,
agregación, 133
asociación, 108, 133
cambiar el estilo de, 135
composición, 133
dependencia, 133
generalización, 108, 133
realización, 133
ver, 137
Relativos,

© 2018 Altova GmbH

Relativos,
vínculos relativos en la documentación, 262
Rendimiento,
mejorar, 158
Repositorios, 428
Respuesta,
mensaje (generar automáticamente la respuesta), 335
Restaurar,
acceso directo, 497
barras de herramientas y ventanas, 491
comandos de la barra de herramientas, 493
comandos de menú, 499
Restricción,
revisar sintaxis, 485
RestricciónDeDuración,
diagrama de ciclo de vida, 358
RestricciónDeTiempo,
diagrama de ciclo de vida, 359
Restringir,
clasificadores, 256
Reutilizar,
subproyectos, 420
Revisar,
sintaxis del proyecto, 485
Ruta de acceso,
cambiar ubicación del proyecto, 146
carpeta de ejemplos, 10
plantillas SPL, 519

S

Salida,
archivo XMI de salida, 414
Salto de línea,
en el texto del actor, 14
SC,
syntax coloring, 367
Secuencia,
diagrama de, 326
iconos, 473
Segundo plano,
actualización de estado - aumentar el intervalo, 428
Set,
métodos getter / setter, 361
Setter / Getter,
generar diagrama de secuencia a partir de métodos setter/getter, 343
Show,
tagged values, 403
Símbolos,
iconos de visibilidad, 361
Sincronización,
cambiar nombres de las clases, 196
opciones de configuración, 193
Sincronizar,
en la ubicación nueva, 146
nombre de las clases y el nombre del archivo de código, 196
raíz / paquetes / clases, 193
Sintaxis,
archivo de procesamiento por lotes, 98
revisar sintaxis del proyecto, 485
Sintaxis del proyecto,
revisar, 90
Slot,
mostrar / ocultar, 361
Sobrescribir,
código con el modelo, 485
modelo con el código, 485
Socket,
Ball and socket (notación de forma esférica), 361
SPL, 519
condiciones, 528
foreach, 529
plantillas SPL definidas por el usuario, 193
subrutinas, 530
StarTeam, 428
Subproyecto,
abrir / editar, 420
control de versiones, 418
crear, 420
guardar archivos de subproyectos, 420
opciones para trabajo en equipo, 418
Syntax coloring, 367

T

Tagged values,
show, 403
Tecla de acceso rápido, 497
Tipo,
propiedad (ver), 259

Tipo,
propiedad de tipo línea de vida, 329
Tipo de datos,
definir en esquema, 406
Tipo de diagrama,
identificar, 92
Todos,
expandir / contraer, 361
Trabajo en equipo,
subproyectos, 418
Transición, 293
agregar diagrama de actividades a, 293
definir disparador, 293
definir entre estados, 293
Transparente,
color de fondo, 394
Tres bandas,
fusión del proyecto, 251
fusión manual del proyecto, 253
Tutorial, 10
archivos de ejemplo, 10
carpeta de ejemplos, 10
objetivos, 10

U

Ubicación,
mover proyecto, 146
UML,
compartir diagrama UML, 155
diagramas, 274
iconos de visibilidad, 361
plantillas, 256
variables, 519
UModel,
importar XMI generado, 414
iniciar, 11
UMP,
cambiar ubicación del proyecto, 146
extensión de archivo, 146
URL,
abrir archivo desde, 481
Uso,
dependencia, 46
Uso de interacción, 334
Usuario,
carpeta de ejemplos multiusuario, 10
UUID,
identificadores únicos universales, 414

V

Valor general de la línea de vida,
diagrama de ciclo de vida, 353
Valores,
etiquetados, 385, 386
etiquetados (mostrar), 399
Valores etiquetados,
mostrar, 399
value,
tagged, show, 403
Variables,
argumentos de herramientas externas, 494
UML, 519
VB.NET,
generar código, 159
importar código fuente, 176
opciones de generación de código, 164
opciones de importación de código, 178
Velocidad,
acelerar actualización de estado en segundo plano, 428
Ventanas, 510
restaurar, 491
Ver,
propiedad como asociación, 259
u ocultar nombre de región, 299
varias instancias del elemento, 361
Vínculos,
relativos a la documentación, 262
Visibilidad,
iconos (seleccionar), 361
Vista, 490
Vista de elementos,
como paquete, 110

X

XMI, 414
extensiones, 414
pretty print, 414

© 2018 Altova GmbH