WhitePaper

Web services: Benefits, challenges, and a unique, visual development solution

Altova, Inc. • 900 Cummings Center, Suite 314-T • Beverly, MA, 01915-6181, USA • Tel: 978-816-1600 • Fax: 978-816-1606 • Web: www.altova.com

Web services: Benefits, challenges, and a unique, visual development solution

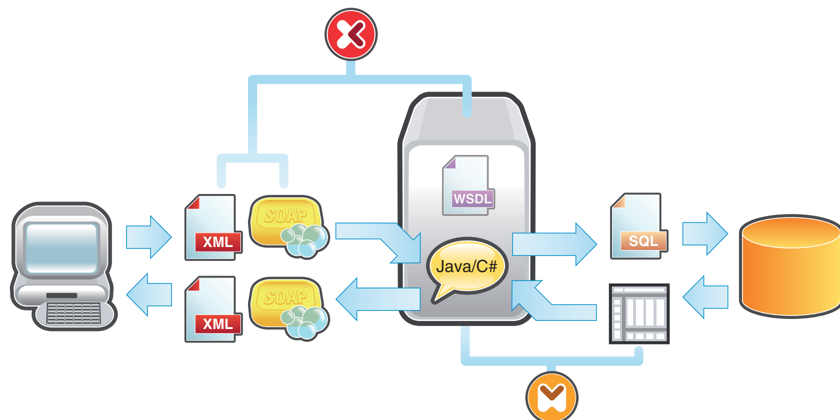
Written by Erin Cavanaugh
Product Marketing Manager
Altova

Executive Summary	3
Web services benefits	7
Web services development challenges	8
Graphical WSDL creation and editing	11
Visual Web services implementation	15
SOAP creation and debugging	18
Conclusion	20
About Altova	20
Resources for further information	21

Executive Summary

Today, the ability to seamlessly exchange information between internal business units, customers, and partners is vital for success; yet most organizations employ a variety of disparate applications that store and exchange data in dissimilar ways and therefore cannot "talk" to one another productively. Web services have evolved as a practical, cost-effective solution for uniting information distributed between critical applications over operating system, platform, and language barriers that were previously impassable.

This whitepaper introduces the concept of Web services and identifies the benefits and challenges associated with their implementation. Then, it discusses how Altova's visual approach to Web services design and implementation can reduce the complexity of Web services development, allowing organizations to quickly reap the benefits of this powerful technology.



What are Web services?

Web services are software components that communicate using pervasive, standards-based Web technologies including HTTP and XML-based messaging. Web services are designed to be accessed by other applications and vary in complexity from simple operations, such as checking a banking account balance online, to complex processes running CRM (customer relationship management) or enterprise resource planning (ERP) systems. Since they are based on open standards such as HTTP and XML-based protocols including SOAP and WSDL, Web services are hardware, programming language, and operating system independent. This means that applications written in different programming languages and running on different platforms can seamlessly exchange data over intranets or the Internet using Web services.

Web services are powered by XML and three other core technologies: WSDL, SOAP, and UDDI. Before building a Web service, its developers create its definition in the form of a WSDL document that describes the service's location on the Web and the functionality the service provides. Information about the service may then be entered in a UDDI registry, which allows Web service consumers to search for and locate the services they need. This step is optional but is beneficial when a company wants its Web services to be discovered by internal and/or external service consumers. Based on information in the UDDI registry, the Web services client developer uses instructions in the WSDL to construct SOAP messages for exchanging data with the service over HTTP. More about these core technologies is detailed below.

XML (eXtensible Markup Language)

XML is a W3C (World Wide Web Consortium) specification that defines a meta-language for describing data. In XML applications, data is described by surrounding it with customizable, text-based tags that give information about the data itself as well as its hierarchical structure.

Because XML syntax consists of text-based mark-up that describes the data being tagged, it is both application-independent and human readable. This simplicity and interoperability have helped XML achieve widespread acceptance and adoption as the standard for exchanging information between heterogeneous systems in a wide variety of applications, including Web services.

XML forms the basis for all modern Web services, which use XML-based technologies to describe their interfaces and to encode their messages. WSDL, SOAP, and UDDI all use XML-based messaging that any machine can interpret.

WSDL (Web Services Description Language)

Also maintained by the W3C, WSDL is an XML-based format for describing Web services. Clients wishing to access a Web service can read and interpret its WSDL file to learn about the location of the service and its available operations. In this way, the WSDL definition acts as the initial Web service interface, providing clients with all the information they need to interact with the service in a standards-based way. Through the WSDL, a Web services client learns where a service can be accessed, what operations the service performs, the communication protocols the service supports, and the correct format for sending messages to the service.

A WSDL file is an XML document that describes a Web service using six main elements:

- Port type – groups and describes the operations performed by the service through the defined interface.
- Port – specifies an address for a binding, i.e., defines a communication port.
- Message – describes the names and format of the messages supported by the service.
- Types – defines the data types (as defined in an XML Schema) used by the service for sending messages between the client and server.
- Binding – defines the communication protocols supported by the operations provided by the service.
- Service – specifies the address (URL) for accessing the service.

The WSDL document that describes a Web service acts as a contract between Web service client and server. By adhering to this contact the service provider and consumer are able to exchange data in a standard way, regardless of the underlying platforms and applications on which they are operating.

SOAP (Simple Object Access Protocol)

SOAP is an XML-based protocol from the W3C for exchanging data over HTTP. It provides a simple, standards-based method for sending XML messages between applications. Web services use SOAP to send messages between a service and its client(s). Because HTTP is supported by all Web servers and browsers, SOAP messages can be sent between applications regardless of their platform or programming language. This quality gives Web services their characteristic interoperability.

SOAP messages are XML documents that contain some or all of the following elements:

- Envelope – specifies that the XML document is a SOAP message; encloses the message itself.
- Header (optional) – contains information relevant to the message, e.g., the date the message was sent, authentication data, etc.
- Body – includes the message payload.
- Fault (optional) – carries information about a client or server error within a SOAP message.

Data is sent between the client(s) and the Web service using request and response SOAP messages, the format for which is specified in the WSDL definition. Because the client and server adhere to the WSDL contract when creating SOAP messages, the messages are guaranteed to be compatible.

UDDI (Universal Description Discovery and Integration)

UDDI is a standard sponsored by OASIS (Organization for the Advancement of Structured Information Standards). Often described as the yellow pages of Web services, UDDI is a specification for creating an XML-based registry that lists information about businesses and the Web services they offer. UDDI provides businesses a uniform way of listing their services and discovering services offered by other organizations. Though implementations vary, UDDI often describes services using WSDL and communicates via SOAP messaging. Registering a Web service in a UDDI registry is an optional step, and UDDI registries can be public or private (i.e. isolated behind a corporate firewall). To search for a Web service, a developer can query a UDDI registry to obtain the WSDL for the service he/she wishes to utilize. Developers can also design their Web services clients to receive automatic updates about any changes to a service from the UDDI registry.

Web services benefits

Web services provide several technological and business benefits, a few of which include:

- Application and data integration
- Versatility
- Code re-use
- Cost savings

The inherent interoperability that comes with using vendor, platform, and language independent XML technologies and the ubiquitous HTTP as a transport mean that any application can communicate with any other application using Web services. The client only requires the WSDL definition to effectively exchange data with the service – and neither part needs to know how the other is implemented or in what format its underlying data is stored. These benefits allow organizations to integrate disparate applications and data formats with relative ease.

Web services are also versatile by design. They can be accessed by humans via a Web-based client interface, or they can be accessed by other applications and other Web services. A client can even combine data from multiple Web services to, for instance, present a user with an application to update sales, shipping, and ERP systems from one unified interface – even if the systems themselves are incompatible. Because the systems exchange information via Web services, a change to the sales database, for example, will not affect the service itself.

Code re-use is another positive side-effect of Web services' interoperability and flexibility. One service might be utilized by several clients, all of which employ the operations provided to fulfill different business objectives. Instead of having to create a custom service for each unique requirement, portions of a service are simply re-used as necessary.

All these benefits add up to significant cost savings. Easy interoperability means the need to create highly customized applications for integrating data, which can be expensive, is removed. Existing investments in systems development and infrastructure can be utilized easily and combined to add additional value. Since Web services are based on open standards their cost is low and the associated learning curve is smaller than that of many proprietary solutions. Finally, Web services take advantage of ubiquitous protocols and the Web infrastructure that already exists in every organization, so they require little if any additional technology investment.

Web services development challenges

With the numerous advantages of Web services come a few challenges. Most significantly, though Web services themselves are designed to be simple, actually developing and implementing them can be complex. WSDL syntax becomes complicated quickly, especially when building a service with multiple operations in a text-based editor. A snippet of WSDL code is shown in Figure 1. Even looking at the completed code, it's difficult to follow the chain of connections from a service name, to the binding, to the port type, and so on, never mind writing the code correctly by hand.

```
<message name="messageName"/>
<message name="GetAgencyNameINput">
  <part name="parameter" type="xs:string"/>
</message>
<message name="GetAgentNameResponse">
  <part name="parameter" type="xs:string"/>
</message>
<portType name="SOAPport">
  <operation name="GetAgencyName">
    <input message="y:GetAgencyNameINput"/>
    <output message="y:GetAgentNameResponse"/>
  </operation>
</portType>
<binding name="AgencyQuerySOAP" type="y:SOAPport">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetAgencyName">
    <soap:operation/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
<service name="AgencyQuery">
  <port name="QueryPort" binding="y:AgencyQuerySOAP">
    <soap:address location="http://www.xmlspy.com"/>
  </port>
</service>
```

Figure 1: Sample WSDL code

There are tools that will auto-generate WSDL code based on an existing application that a developer wants to expose as a Web service. However, best practices dictate that designing the WSDL be the first step in architecting Web services. The contract-first approach to Web services development has many advantages. Designing the interface first results in better overall planning prior to implementing the service and helps ensure the service will be effective in multiple client scenarios. In addition, WSDL uses language-independent XML Schema for type definitions, allowing Web service designers to specify strongly-typed requirements for communicating with the Web service. Using XML Schema, a Web services developer can specify that an email address must follow a certain pattern (username@domainname.xxx), that a product quantity must be a positive whole integer, that a first name and last name are required input values, and so on. This same level of data typing is not provided when a WSDL is auto-generated from an existing Java or C# application.

In addition, because WSDL is standards-based, designing the WSDL first, then building the Web service based on this definition prevents developers from including language-specific types and constructs in their Web service. This ensures that any Web services client can interact with the service without interoperability issues. Though the WSDL contract-first approach may seem more rigorous at first, the resulting benefits make the service more effective by ensuring interoperability – which, after all, is the rationale behind using Web services in the first place. Lack of tool support is often cited as the biggest obstacle to the contract-first approach to Web services design.

Another challenge arises after the WSDL is defined, when the developer must actually write the Java or C# code to connect the required data sources and implement the service on a server. Given that even the simplest of Web services may require thousands of lines of code, this process is often time-consuming and error prone. In addition, given that Web services require the expertise of XML developers who may not necessarily also be Java or C# experts, this step can be especially challenging.

Recognizing these barriers to Web services development, Altova has created a suite of tools for designing and building Web services in a graphical manner. This approach allows developers to build well designed, standards-conformant, interoperable Web services – without manual coding.

Altova Web services development

Altova XMLSpy® 2006 and MapForce® 2006 provide graphical tools for designing and building Web services from start to finish.

Altova XMLSpy 2006 is a complete XML development environment that includes a graphical WSDL editor. This unique graphical approach simplifies WSDL development by enabling an easy-to-understand visual process. Instead of working solely with a text view, developers can build their WSDL files graphically, with full validation and editing help, and the corresponding WSDL code is generated behind the scenes where it can be referenced and edited at any time. This eliminates much of the complexity otherwise associated with the design-before-implementation development approach.

Once a WSDL file is created, the Web services developer must still write the extensive Java or C# code to programmatically connect the operations defined in the WSDL with the data they will return. Altova MapForce automates this step with its graphical WSDL mapping interface and code generation capabilities. MapForce displays WSDL operations as visual mapping items that can be dragged and dropped to connect with their corresponding data sources. You can also filter and process data before returning it as a Web services response message. Once a WSDL mapping design is complete, MapForce auto-generates the Java or C# code required to implement the service server-side.

Completing the Web services design loop are the SOAP features in XMLSpy. XMLSpy includes a SOAP client that automatically creates SOAP messages based on criteria defined in a WSDL document. Then it actually sends the message to the Web service and displays the response. This feature is very useful for testing your own Web services implementations, and it's invaluable for interpreting WSDL documents when you need to create a client based on someone else's WSDL definition.

XMLSpy also includes a SOAP debugger, which acts as a Web services proxy between the client and server, allowing you to intercept and examine the actual messages sent between a client and server. The SOAP debugger allows you to set functional or conditional breakpoints on request and response messages to quickly track down and debug any problems with a Web service client or server implementation.

More about each of these tools and features is detailed below.

Graphical WSDL creation and editing

Shown in Figure 2, the XMLSpy graphical WSDL editor facilitates creating, editing, visualizing, and validating any WSDL file.

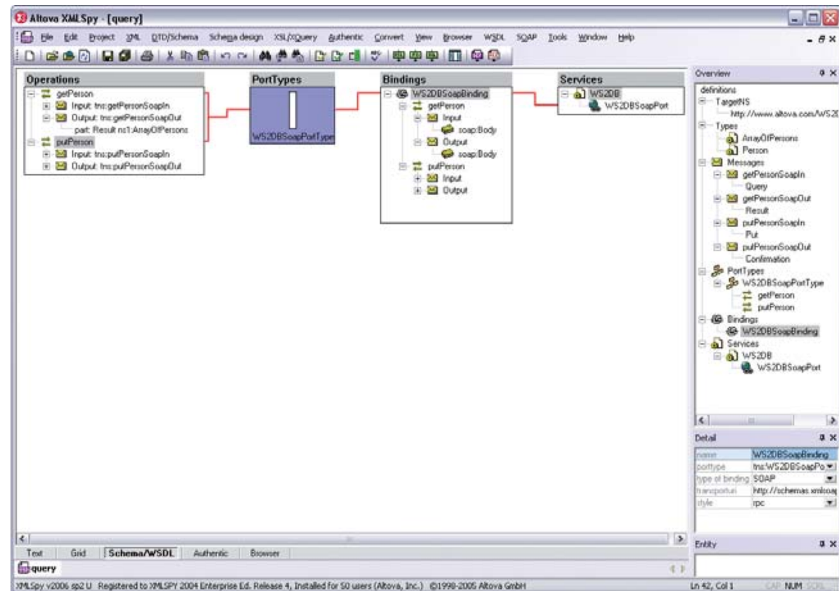


Figure 2: Altova XMLSpy 2006 graphical WSDL editor

It displays the WSDL file structure as well as the WSDL elements grouped by operations, portTypes, bindings, and services. You can manipulate the file by dragging and dropping elements, and context-sensitive windows and entry helpers provide useful editing options.

While you're working in the graphical view, the corresponding XML markup is built behind the scenes and is available under XMLSpy's text view tab, and you can, of course, edit the text directly in its text format.

```
24  <message name="getPersonSoapIn">
25    <part name="Query" type="xsd:string"/>
26  </message>
27  <message name="getPersonSoapOut">
28    <part name="Result" type="ns1:ArrayOfPersons" xmlns:ns1="http://www.altova.com/WS2DB.xsd"/>
29  </message>
30  <message name="putPersonSoapIn">
31    <part name="Put" type="ns1:Person" xmlns:ns1="http://www.altova.com/WS2DB.xsd"/>
32  </message>
33  <message name="putPersonSoapOut">
34    <part name="Confirmation" type="xsd:string"/>
35  </message>
36  <portType name="WS2DBSoapPortType">
37    <operation name="getPerson">
38      <input name="getPersonSoapInput" message="tns:getPersonSoapIn"/>
39      <output name="getPersonSoapOutput" message="tns:getPersonSoapOut"/>
40    </operation>
41    <operation name="putPerson">
42      <input name="putPersonSoapInput" message="tns:putPersonSoapIn"/>
43      <output name="putPersonSoapOutput" message="tns:putPersonSoapOut"/>
44    </operation>
45  </portType>
46  <binding name="WS2DBSoapBinding" type="tns:WS2DBSoapPortType">
47    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
48    <operation name="getPerson">
49      <soap:operation soapAction="getPerson" style="rpc"/>
50      <input>
51        <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
52      </input>
53      <output>
54        <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
55      </output>
56    </operation>
57    <operation name="putPerson">
58      <soap:operation soapAction="putPerson" style="rpc"/>
59      <input>
60        <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
61      </input>
62      <output>
63        <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
64      </output>
65    </operation>
```

Figure 3: Snippet of code auto-generated from WSDL design in Figure 2

When you create a new file from scratch in the graphical WSDL view, XMLSpy presents you with a graphical representation consisting of a box for each type of WSDL element: operations, portTypes, bindings, and services. This visual representation allows you to easily see the structure of the WSDL file and edit each portion in a logical way. Since the "skeleton" WSDL structure has already been created, you can edit each portion separately in any order – whatever makes the most sense to you. In general, to create a WSDL, the following must be completed:

• Specify an XML Schema to define message formatting
• Name the service
• Associate ports with the service
• Associate a binding name, type, and location for each port
• Add operations to the service
• Name each operation, then add input, output, and/or fault elements
• Validate the WSDL file and correct any validation errors

There are several options and helpful features available in XMLSpy for completing these steps, as described below.

In addition to the visual WSDL design view, XMLSpy also includes a graphical XML Schema editor, shown in Figure 4. Using the Types entry on the WSDL menu, you can switch to this view to create an XML Schema from scratch, import a schema, or edit an existing schema easily, using the same visual paradigm employed for WSDL design.

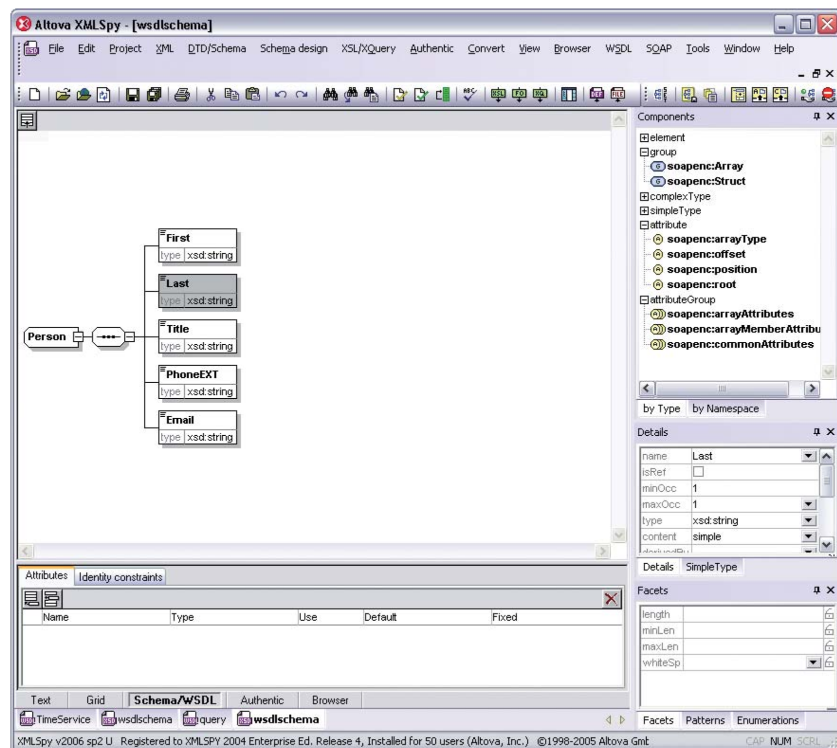


Figure 4: Graphical view of XML Schema associated with WSDL in Figure 2

Once the XML Schema is created or imported, the datatypes specified in the schema become available in the WSDL design view entry helpers.

To the right of the design window, the overview window reveals the structure of the WSDL document. It displays the components of the WSDL file in a tree view in the sequence in which they appear in the code. You can expand and collapse components to reveal and hide sub-components. Selecting a component in the overview window or from within the design itself displays that component and its properties, for example, the name, binding, and location of each port, in the details window. These entries are editable by hand or by using context-sensitive drop-down menus.

The graphical WSDL display helps you immediately understand the relationships between the different components in the file and add new elements in a straightforward way. The operations component of the design window lists all the operations in the WSDL, and each is connected to its respective port type with a color-coded line.

Expanding any operation displays its input and output components, and a context-sensitive right click menu lets you append, insert, or delete operations, as well as add input and output messages to any operation. To associate a port type with an operation, simply drag the operation to the desired port – the same is true for binding and services associations.

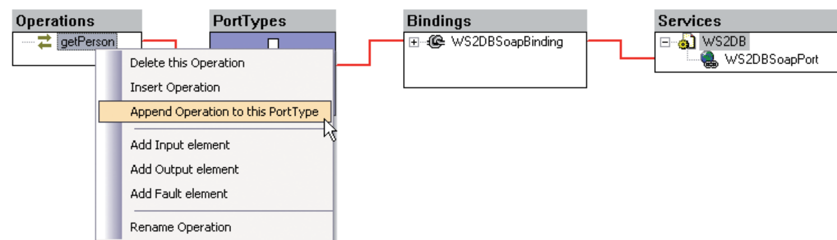


Figure 5: Editing WSDL operations using right-click menu

In addition to connections indicated by colored links, clicking any port type in the port type component highlights all the associated operations in the operations component and in the overview window. Each port type is also linked to the associated binding with the same color-coded line, and bindings are linked to their respective services in the services component. At a glance, you can tell which operations are associated with which port types, bindings, and services, and you can adjust these associations by dragging and dropping elements. This is infinitely easier than trying to extrapolate and edit the same information from a text-based view of a WSDL file.

Selecting a binding lets you view or edit all relevant parameters in the details window:

- The name attribute
- The type attribute, which refers to the specific port type
- The type of binding: SOAP, http/GET, http/POST
- The transport URI
- The style attribute: RPC or document

You can also add, edit, and delete these connections using drag and drop functionality and context-sensitive menus (right click, tool bar, or drop down – whichever you prefer).

One last note about XMLSpy is that it also includes a WSDL documentation generator that will output documentation for any standards-conformant WSDL file. Documentation can be in HTML or Microsoft® Word® and is useful for giving business partners, other developers, or customers a detailed overview of the Web service interface. This feature is also useful for understanding WSDL files that are generated by other platforms.

Overall, designing WSDL using a graphical paradigm makes WSDL development more accessible and productive, which in turn makes the Web services implementation phase more straightforward.

Visual Web services implementation

Once a WSDL is defined, implementing the Web service it describes involves writing the code to access the data required for each transaction. Given that even simple Web services can require thousands of lines of code, this process quickly snowballs. Altova MapForce 2006 adopts the same visual design strategy used in the XMLSpy WSDL design view for building Web services.

MapForce itself is a multi-faceted tool. It's a visual data mapper with full support for XML, any relational database, flat files, EDI messages (EDIFACT and ANSI X12), and with the October 2005 release, WSDL. With its new WSDL mapping support, MapForce automates the Web services implementation process by allowing you to connect data sources and operations visually, then auto-generate the code required to implement the service server-side.

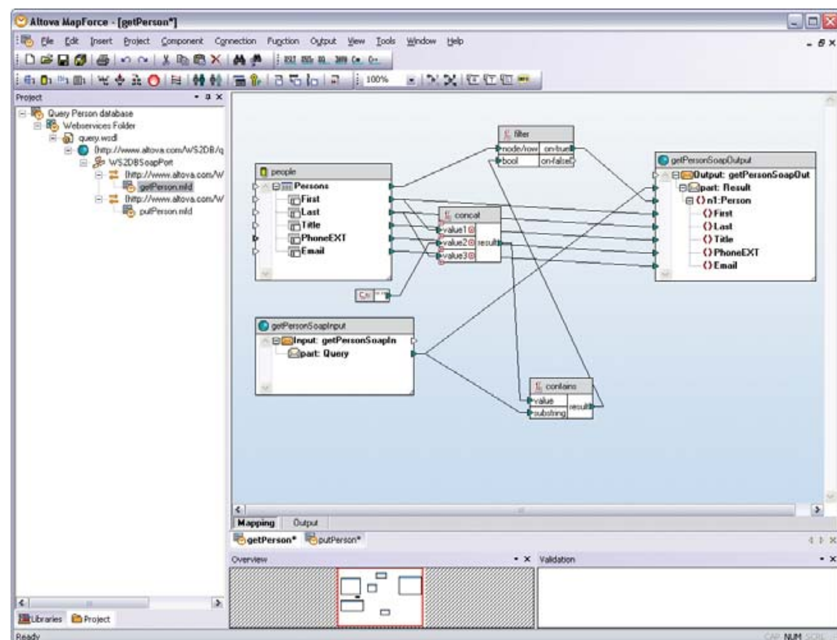


Figure 6: Altova MapForce mapping of a WSDL operation defined in Figure 2

When you open an existing, standards-conformant WSDL file (like one created in XMLSpy), MapForce creates a project view that displays the WSDL and all the operations that it includes. This is shown in the left-hand panel of Figure 6. Double-clicking any operation opens its input and output schemas as graphical components in the mapping design window.

To implement each operation, you have to supply the input data or connect a data source. The MapForce function library, which you can toggle to by clicking its tab in the project window, lets you drag functions onto the mapping design to supply data values with constant components, or to filter and manipulate data before returning it as a response. The data processing function library contains an extensive array of functions: string, mathematical, Boolean, language-related, logical, node-tests, etc., that you can drag and drop onto the mapping design to process source data. You can create and save user-defined functions as well.

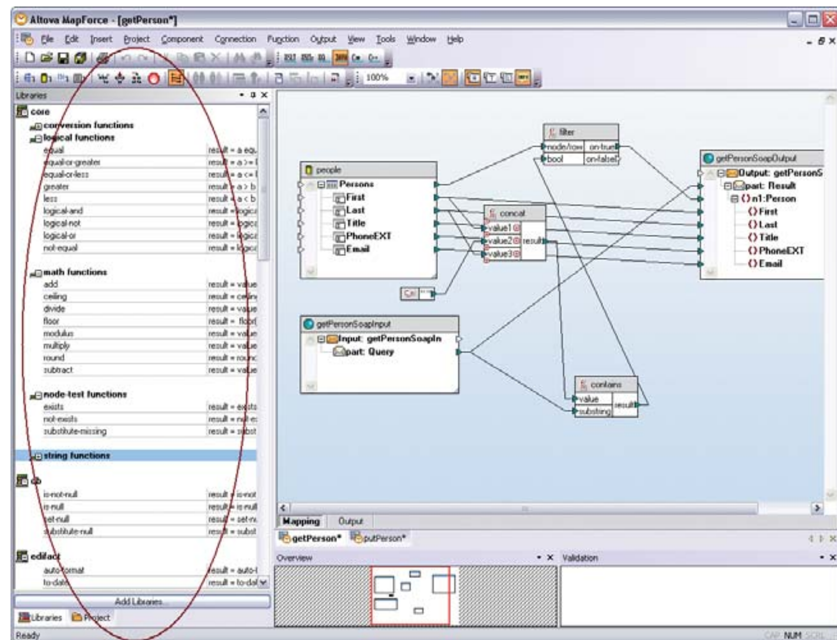


Figure 7: MapForce drag-and-drop data processing function library

You can supply constant data values to Web services operations, and you can connect dynamic data sources in XML, database, flat file, or EDI formats. This way, you can access and combine data from multiple disparate systems within the same operation. This is still a visual drag and drop process – you insert the required data source(s), and MapForce lists their elements in a hierarchical tree view. Then you map the WSDL operation by dragging connecting lines from elements in your data sources, through the data processing functions, to the corresponding targets in the response component. That's all there is to it. A Web services mapping using a database source is shown in Figure 6.

The MapForce output tab allows you to test each mapping by specifying a SOAP request message with sample input value(s). (You can auto-generate the SOAP request message in XMLSpy as described in the next section.) Clicking the output tab displays the SOAP response message that will be returned by the Web service based on your input SOAP request.

After mappings are defined and (optionally) tested for all the operations in given WSDL, MapForce will auto-generate code for the entire project in either Java or C# to implement the service on a server. Alternately, you can generate code for one or a few of the operations in the WSDL separately. All that's left to do is to compile the generated code and place it on a Web server.

There are several advantages to using this tool-generated code. First, it greatly simplifies Web services development by allowing you to concentrate on implementing the business logic of a service – instead of manually writing thousands of lines of complicated code in a programming language that may be unfamiliar. Code generation based on visual design also ensures that the code is written consistently across the entire project, since it's produced according to industry standards and globally defined parameters, rather than having multiple engineers manually writing the code. This high degree of software code consistency ensures that code is standards-conformant, interoperable, and reusable.

SOAP creation and debugging

Once a Web service is live on a server, the Web service developer will need to test it, and eventually a client developer will need to communicate with it. Either scenario requires the use of SOAP messaging based on the WSDL contract. XMLSpy will interpret an existing WSDL file to automatically generate SOAP messages. When you open an existing WSDL in XMLSpy, the SOAP client presents you with a list of the operations included in the service. You can select any operation, and the SOAP client will auto-generate a standards-conformant SOAP request message based on the criteria defined in the WSDL.

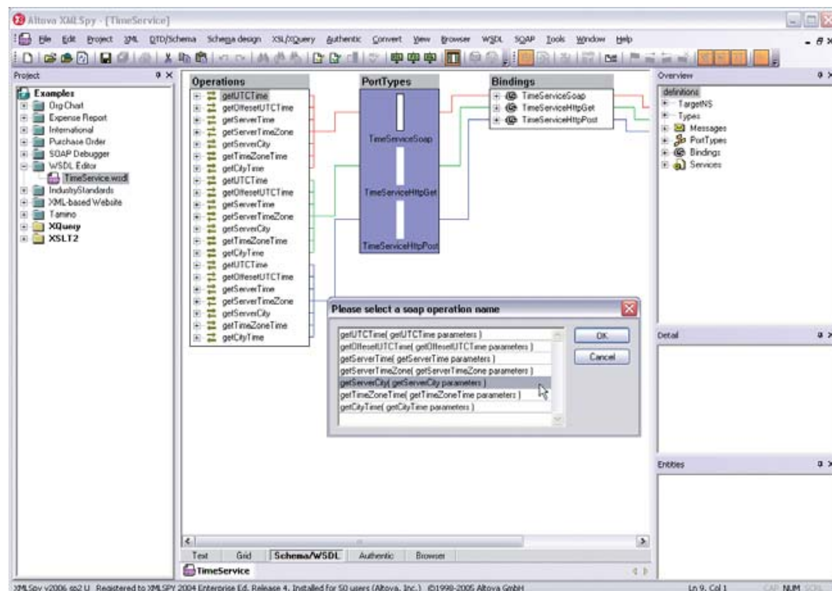


Figure 8: Auto-generating a SOAP request

Then, you can actually send that generated request to the Web service, and XMLSpy displays the response SOAP message returned by the server.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="
   http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <soap:Body>
4      <getServerCityResponse xmlns="http://www.Nanonull.com/TimeService/">
5        <getServerCityResult>BOSTON</getServerCityResult>
6      </getServerCityResponse>
7    </soap:Body>
8  </soap:Envelope>
9
```

Figure 9: SOAP response message

The XMLSpy SOAP client lets you create SOAP messages without manual coding to test a Web service or to write client-side Web services code.

Further testing capabilities are provided by the SOAP debugger. The SOAP debugger acts as a Web services proxy that allows you to intercept and inspect the SOAP request and response messages that are sent between the client and server. The SOAP debugger displays two panes: one for the SOAP request messages and one for the response messages. If the service you're working with has an Web-based client interface, you can display that as well in the XMLSpy browser view to see the request messages, response messages, and the client interface side-by-side.

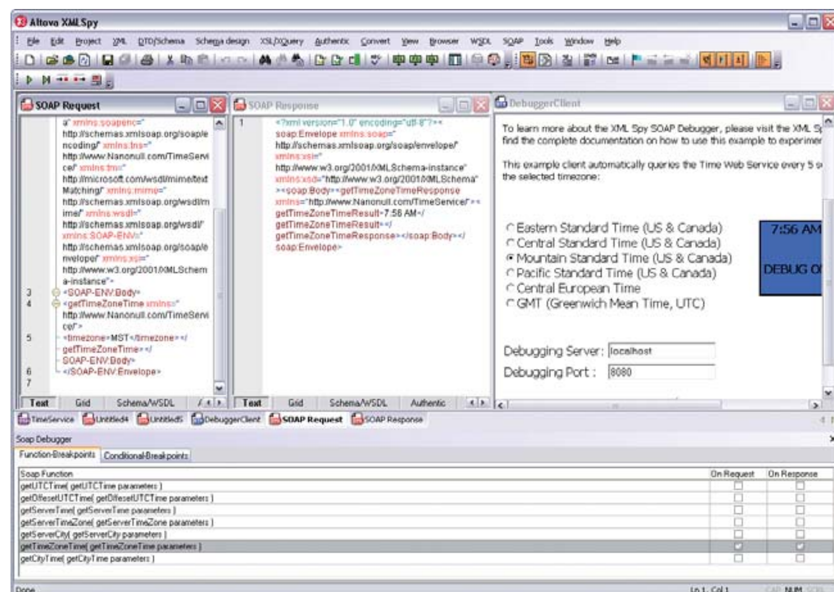


Figure 10: XMLSpy SOAP debugger

Below the message panes are two breakpoint tabs, which allow you to automatically halt the debugger when a particular function or condition is triggered in a request or response message. This way, you can see exactly what data is being sent between the client and server at any given point in the transaction. While functional breakpoints allow you to stop the debugger during a particular operation, conditional breakpoints stop the debugger when a particular value is returned in a request or response message. For instance, if a SOAP request causes an error, the response message will contain a "faultcode" element. You can set a breakpoint to trigger whenever a faultcode element appears to test the service for any errors.

The XMLSpy SOAP client and debugger complete the Web services development cycle by allowing you to connect to, test, debug, and perfect your Web services graphically. Because XMLSpy automatically interprets WSDL documents to create and debug SOAP messages, it allows you to focus on the logic of your service without being mired in manual code writing.

Conclusion

Using Altova XMLSpy 2006 and MapForce 2006 together, you can create Web services from start to finish in a visual manner. This speeds development and reduces the occurrence of errors that may be introduced by manual coding.

By creating a WSDL file in XMLSpy, then building the corresponding Web service and generating program code in MapForce, and creating and testing SOAP messages in XMLSpy, you can effectively build a complete Web service without having to write a single line of code. These Altova tools allow you to reap all the benefits of Web services without being slowed by their challenges. The result is effective Web services with increased standards-conformance and code quality.

XMLSpy and MapForce are available separately or as part of the Altova XML Suite. The Altova XML Suite delivers the five leading XML and Web services development tools at a special ½ off price. You'll get all the Web services development power of XMLSpy and MapForce, plus three other leading XML tools, all for less than purchasing XMLSpy and MapForce separately.

Free trial versions are available for download at **www.altova.com/download.**

About Altova

Altova accelerates application development and data management projects with software and solutions that enhance productivity and maximize results. As an innovative, customer-focused company and the creator of XMLSpy and other leading XML, data management, UML, and Web services tools, Altova is the choice of over 2 million clients worldwide, including virtually every Fortune 500 company. With customers ranging from vast development teams in the world's largest organizations to progressive one-person shops, Altova's line of software tools fulfills a broad spectrum of business needs. Altova is an active member of the World Wide Web Consortium (W3C) and Object Management Group (OMG) and is committed to delivering standards-based, platform-independent solutions that are powerful, affordable, and easy-to-use. Altova was founded in 1992 and has headquarters in Beverly, Massachusetts and Vienna, Austria.

Visit Altova on the Web at **www.altova.com.**

Resources for further information

- The World Wide Web Consortium: **http://www.w3.org/**
  W3C Web Services Activity: **http://www.w3.org/2002/ws/**
  WSDL Primer: **http://www.w3.org/TR/wsdl20-primer/**
  SOAP Primer: **http://www.w3.org/TR/soap12-part0/**
- Web Services Interoperability Organization: **http://www.ws-i.org/**
- Organization for the Advancement of Structured Information Standards: **http://www.oasis-open.org/**
- W3 Schools: **http://www.w3schools.com/**

© Copyright 2006 Altova® GmbH. All rights reserved. Altova®, XMLSpy®, MapForce®, StyleVision®, SemanticWorks™, SchemaAgent™, UModel®, DiffDog®, and Authentic® are trademarks or registered trademarks of Altova® GmbH in the United States and/or other countries. Microsoft and Windows are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both. Other company, product, and service names may be trademarks or service marks of others.