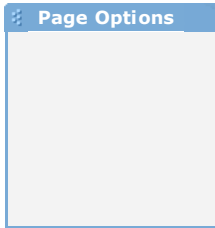


XML Web Services Basics

Roger Wolter
Microsoft Corporation

December 2001

Summary: An overview of the value of XML Web services for developers, with introductions to SOAP, WSDL, and UDDI.
(6 printed pages)



Contents

[What Is an XML Web Service?](#)

[SOAP](#)

[WSDL](#)

[UDDI](#)

[What's Left?](#)

What Is an XML Web Service?

XML Web services are the fundamental building blocks in the move to distributed computing on the Internet. Open standards and the focus on communication and collaboration among people and applications have created an environment where XML Web services are becoming the platform for application integration. Applications are constructed using multiple XML Web services from various sources that work together regardless of where they reside or how they were implemented.

There are probably as many definitions of XML Web Service as there are companies building them, but almost all definitions have these things in common:

- XML Web Services expose useful functionality to Web users through a standard Web protocol. In most cases, the protocol used is SOAP.
- XML Web services provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them. This description is usually provided in an XML document called a Web Services Description Language (WSDL) document.
- XML Web services are registered so that potential users can find them easily. This is done with Universal Discovery Description and Integration (UDDI).

I'll cover all three of these technologies in this article but first I want to explain why you should care about XML Web services.

One of the primary advantages of the XML Web services architecture is that it allows programs written in different languages on different platforms to communicate with each other in a standards-based way. Those of you who have been around the industry a while are now saying, "Wait a minute! Didn't I hear those same promises from CORBA and before that DCE? How is this any different?" The first difference is that SOAP is significantly less complex than earlier approaches, so the barrier to entry for a standards-compliant SOAP implementation is significantly lower. Paul Kulchenko maintains a list of SOAP implementations at:

<http://www.soapware.org/directory/4/implementations> which at last count contained 79 entries. You'll find SOAP implementations from most of the big software companies, as you would expect, but you will also find many implementations that are built and maintained by a single developer. The other significant advantage that XML Web services have over previous efforts is that they work with standard Web protocols—XML, HTTP and TCP/IP. A significant number of companies already have a Web infrastructure, and people with knowledge and experience in maintaining it, so again, the cost of entry for XML Web services is significantly less than previous technologies.

We've defined an XML Web service as a software service exposed on the Web through SOAP, described with a WSDL file and registered in UDDI. The next logical question is, "What can I do with XML Web services?" The first XML Web services tended to be information sources that you could easily incorporate into applications—stock quotes, weather forecasts, sports scores etc. It's easy to imagine a whole class of applications that could be built to analyze and aggregate the information you care about and present it to you in a variety of ways; for example, you might have a Microsoft® Excel spreadsheet that summarizes your whole financial picture—stocks, 401K, bank accounts, loans, etc. If this information is available through XML Web services, Excel can update it continuously. Some of this information will be free and some might require a subscription to

the service. Most of this information is available now on the Web, but XML Web services will make programmatic access to it easier and more reliable.

Exposing existing applications as XML Web services will allow users to build new, more powerful applications that use XML Web services as building blocks. For example, a user might develop a purchasing application to automatically obtain price information from a variety of vendors, allow the user to select a vendor, submit the order and then track the shipment until it is received. The vendor application, in addition to exposing its services on the Web, might in turn use XML Web services to check the customer's credit, charge the customer's account and set up the shipment with a shipping company.

In the future, some of the most interesting XML Web services will support applications that use the Web to do things that can't be done today. For example, one of the services that XML Web Services would make possible is a calendar service. If your dentist and mechanic exposed their calendars through this XML Web service, you could schedule appointments with them on line or they could schedule appointments for cleaning and routine maintenance directly in your calendar if you like. With a little imagination, you can envision hundreds of applications that can be built once you have the ability to program the Web.

For more information on XML Web services and the applications they will help you build, please see the MSDN [XML Web Services Developer Center](#).

SOAP

Soap is the communications protocol for XML Web services. When SOAP is described as a communications protocol, most people think of DCOM or CORBA and start asking things like, "How does SOAP do object activation?" or "What naming service does SOAP use?" While a SOAP implementation will probably include these things, the SOAP standard doesn't specify them. SOAP is a specification that defines the XML format for messages—and that's about it for the required parts of the spec. If you have a well-formed XML fragment enclosed in a couple of SOAP elements, you have a SOAP message. Simple isn't it?

There are other parts of the SOAP specification that describe how to represent program data as XML and how to use SOAP to do Remote Procedure Calls. These optional parts of the specification are used to implement RPC-style applications where a SOAP message containing a callable function, and the parameters to pass to the function, is sent from the client, and the server returns a message with the results of the executed function. Most current implementations of SOAP support RPC applications because programmers who are used to doing COM or CORBA applications understand the RPC style. SOAP also supports document style applications where the SOAP message is just a wrapper around an XML document. Document-style SOAP applications are very flexible and many new XML Web services take advantage of this flexibility to build services that would be difficult to implement using RPC.

The last optional part of the SOAP specification defines what an HTTP message that contains a SOAP message looks like. This HTTP binding is important because HTTP is supported by almost all current OS's (and many not-so-current OS's). The HTTP binding is optional, but almost all SOAP implementations support it because it's the only standardized protocol for SOAP. For this reason, there's a common misconception that SOAP requires HTTP. Some implementations support MSMQ, MQ Series, SMTP, or TCP/IP transports, but almost all current XML Web services use HTTP because it is ubiquitous. Since HTTP is a core protocol of the Web, most organizations have a network infrastructure that supports HTTP and people who understand how to manage it already. The security, monitoring, and load-balancing infrastructure for HTTP are readily available today.

A major source of confusion when getting started with SOAP is the difference between the SOAP specification and the many implementations of the SOAP specification. Most people who use SOAP don't write SOAP messages directly but use a SOAP toolkit to create and parse the SOAP messages. These toolkits generally translate function calls from some kind of language to a SOAP message. For example, the Microsoft SOAP Toolkit 2.0 translates COM function calls to SOAP and the Apache Toolkit translates JAVA function calls to SOAP. The types of function calls and the datatypes of the parameters supported vary with each SOAP implementation so a function that works with one toolkit may not work with another. This isn't a limitation of SOAP but rather of the particular implementation you are

using.

By far the most compelling feature of SOAP is that it has been implemented on many different hardware and software platforms. This means that SOAP can be used to link disparate systems within and without your organization. Many attempts have been made in the past to come up with a common communications protocol that could be used for systems integration, but none of them have had the widespread adoption that SOAP has. Why is this? Because SOAP is much smaller and simpler to implement than many of the previous protocols. DCE and CORBA for example took years to implement, so only a few implementations were ever released. SOAP, however, can use existing XML Parsers and HTTP libraries to do most of the hard work, so a SOAP implementation can be completed in a matter of months. This is why there are more than 70 SOAP implementations available. SOAP obviously doesn't do everything that DCE or CORBA do, but the lack of complexity in exchange for features is what makes SOAP so readily available.

The ubiquity of HTTP and the simplicity of SOAP make them an ideal basis for implementing XML Web services that can be called from almost any environment. For more information on SOAP, see the MSDN [SOAP](#) home page.

What About Security?

One of the first questions newcomers to SOAP ask is how does SOAP deal with security. Early in its development, SOAP was seen as an HTTP-based protocol so the assumption was made that HTTP security would be adequate for SOAP. After all, there are thousands of Web applications running today using HTTP security so surely this is adequate for SOAP. For this reason, the current SOAP standard assumes security is a transport issue and is silent on security issues.

When SOAP expanded to become a more general-purpose protocol running on top of a number of transports, security became a bigger issue. For example, HTTP provides several ways to authenticate which user is making a SOAP call, but how does that identity get propagated when the message is routed from HTTP to an SMTP transport? SOAP was designed as a building-block protocol, so fortunately, there are already specifications in the works to build on SOAP to provide additional security features for Web services. The [WS-Security specification](#) defines a complete encryption system.

WSDL

WSDL (often pronounced whiz-dull) stands for Web Services Description Language. For our purposes, we can say that a WSDL file is an XML document that describes a set of SOAP messages and how the messages are exchanged. In other words, WSDL is to SOAP what IDL is to CORBA or COM. Since WSDL is XML, it is readable and editable but in most cases, it is generated and consumed by software.

To see the value of WSDL, imagine you want to start calling a SOAP method provided by one of your business partners. You could ask him for some sample SOAP messages and write your application to produce and consume messages that look like the samples, but this can be error-prone. For example, you might see a customer ID of 2837 and assume it's an integer when in fact it's a string. WSDL specifies what a request message must contain and what the response message will look like in unambiguous notation.

The notation that a WSDL file uses to describe message formats is based on the XML Schema standard which means it is both programming-language neutral and standards-based which makes it suitable for describing XML Web services interfaces that are accessible from a wide variety of platforms and programming languages. In addition to describing message contents, WSDL defines where the service is available and what communications protocol is used to talk to the service. This means that the WSDL file defines everything required to write a program to work with an XML Web service. There are several tools available to read a WSDL file and generate the code required to communicate with an XML Web service. Some of the most capable of these tools are in Microsoft Visual Studio® .NET.

Many current SOAP toolkits include tools to generate WSDL files from existing program interfaces, but there are few tools for writing WSDL directly, and tool support for WSDL isn't as complete as it should be. It shouldn't be long before tools to author WSDL files, and then generate proxies and stubs much like COM IDL tools, will be part of most SOAP implementations. At that point, WSDL will become the preferred way to author SOAP interfaces for XML Web services.

An excellent [description of WSDL](#) is available, and you can find the

WSDL specification at <http://www.w3.org/TR/wsdl>.

UDDI

Universal Discovery Description and Integration is the yellow pages of Web services. As with traditional yellow pages, you can search for a company that offers the services you need, read about the service offered and contact someone for more information. You can, of course, offer a Web service without registering it in UDDI, just as you can open a business in your basement and rely on word-of-mouth advertising but if you want to reach a significant market, you need UDDI so your customers can find you.

A UDDI directory entry is an XML file that describes a business and the services it offers. There are three parts to an entry in the UDDI directory. The "white pages" describe the company offering the service: name, address, contacts, etc. The "yellow pages" include industrial categories based on standard taxonomies such as the North American Industry Classification System and the Standard Industrial Classification. The "green pages" describe the interface to the service in enough detail for someone to write an application to use the Web service. The way services are defined is through a UDDI document called a Type Model or tModel. In many cases, the tModel contains a WSDL file that describes a SOAP interface to an XML Web service, but the tModel is flexible enough to describe almost any kind of service.

The UDDI directory also includes several ways to search for the services you need to build your applications. For example, you can search for providers of a service in a specified geographic location or for business of a specified type. The UDDI directory will then supply information, contacts, links, and technical data to allow you to evaluate which services meet your requirements.

UDDI allows you to find businesses you might want to obtain Web services from. What if you already know whom you want to do business with but you don't know what services are offered? The [WS-Inspection specification](#) allows you to browse through a collection of XML Web services offered on a specific server to find which ones might meet your needs.

More information about UDDI is available at <http://www.uddi.org/about.html>.



What's Left?

So far we've talked about how to talk to XML Web services (SOAP), how XML Web services are described (WSDL) and how to find XML Web services (UDDI). These constitute a set of baseline specifications that provide the foundation for application integration and aggregation. From these baseline specifications, companies are building real solutions and getting real value from them.

While much work has been done to make XML Web services a reality, more is needed. Today, people are having success with XML Web services, but there are still things that are left as an exercise for the developer@e.g. security, operational management, transactions, reliable messaging. The Global XML Web Services Architecture will help take XML Web services to the next level by providing a coherent, general purpose model for adding new advanced capabilities to XML Web services which is modular and extensible.

[WS-Security](#) is one of the specifications in the Global Web Services Architecture. Operational management needs such as routing messages among many servers and configuring those servers dynamically for processing are also part of the Global Web Services Architecture, and are met by the [WS-Routing specification](#) and the [WS-Referral specification](#). As the Global Web Services Architecture grows, specifications for these and other needs will be introduced.

More information is available on the [Global XML Web Services Architecture](#).

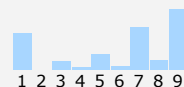
 Print  E-Mail

How would you rate the quality of this content?

Poor 1 2 3 4 5 6 7 8 9 Outstanding

Tell us why you rated the content this way. (optional)

Average rating:
6 out of 9



1851 people have rated this page

Submit

[Contact Us](#) | [E-Mail This Page](#) | [MSDN Flash Newsletter](#) | [Legal](#)

© 2003 Microsoft Corporation. All rights reserved. [Terms of Use](#) [Privacy Statement](#) [Accessibility](#)