Using WSDL in SOAP applications

An introduction to WSDL for SOAP programmers

Uche Ogbuji (uche@ogbuji.net)  
Consultant  
Fourthought, Inc.

Web Services Description Language (WSDL) is a new specification to describe networked XML-based services. It provides a simple way for service providers to describe the basic format of requests to their systems regardless of the underlying protocol (such as Simple Object Access Protocol or XML) or encoding (such as Multipurpose Internet Messaging Extensions). WSDL is a key part of the effort of the Universal Description, Discovery and Integration (UDDI) initiative to provide directories and descriptions of such on-line services for electronic business. This article provides a brief background and technical introduction to WSDL. Knowledge of XML and XML Namespaces is required and some familiarity with XML Schemas and SOAP is useful.

In the past few years, a number of standards proposals have emerged in the past few years to provide a key piece of the XML middleware story: networked service requests. These networked service requests are a way to request XML-related functionality from a remote machine over a network such as the Internet.)

This has led to a standards race including notable entries such as Allaire Corp's Web Distributed Data eXchange (WDDX) (see Resources), UserLand Inc's XML Remote Procedure Call (XML-RPC), Microsoft's Simple Object Access Protocol (SOAP) by David Winer, IBM, Microsoft, and others (see Resources). At the same time, some developers have even done quite well building applications over plain old HTTP. The biggest growth area for such XML-based networked services have been in content exchange and syndication.

Similarly, there have also been a number of proposals for defining the descriptions and structure of such content. Of these, the notable ones include Information Content Exchange (ICE) from Vignette Corp and its partners (see Resources), and the RDF (Resource Description Framework) Site Summary (RSS) from Netscape and its partners (see Resources). Many developers have also done very well using the common Internet standard of Multipurpose Internet Messaging Extensions (MIME).

There are many, many other XML protocol initiatives out there; enough so that the W3C's has a brand new XML Protocol Working Group just for addressing these issues (see Resources).
should be very interesting to watch the political sparks fly as the W3C tries to extract something coherent from this welter.

In this bewildering array of ways to communicate between web applications, a clear need has emerged for a mechanism to describe XML-based network services regardless of communications protocol and request structure. With such a mechanism many advanced web development tasks could gain an additional measure of automation. For example:

- Portal toolkits could provide a plug-in system for content sections to make it easier for designers to pick from a wide range of on-line services without delving into a lot of technical details.
- Industry groups and service brokers could publish comprehensive white pages and yellow pages of on-line XML services, allowing developers to make quick technological assessments and comparisons.
- Service providers could quickly publish updates and versions of their request structures in a standard format to help automate adoption by developers.

IBM, Ariba, and Microsoft set out to craft just such a mechanism, and on September 25th emerged with the Web Services Description Language (WSDL) version 1.0 (see Resources). It is rather odd that this "1.0" spec was pretty much under wraps until then; thus the XML community was left with no chance at a public review before the release date. At any rate, WSDL is a format for describing networked XML services, filling a large portion of the need I described earlier.

Background

WSDL occupies a space with many of its precedents overlapping some of the specifications. Let's take a brief look at the menagerie to provide some background. WebMethods' Web Interface Definition Language (WIDL), one of the pioneering specifications for description of remote web services, was an XML format that took familiar approach (which was accessing functionality on a remote machine as if it were on a local machine) to users of remote procedural technologies, such as RPC and CORBA. There was some fit between WIDL and the XML-RPC system by UserLand. The former has since faded away, as message-based XML technologies have proven more popular than their procedural equivalents. The latter seems to be giving way to SOAP, which has support for message-oriented as well as procedural approaches.

SOAP describes envelope and message formats, and has a basic request/response handshake protocol. Additionally, Microsoft developed the SOAP Contract Language (SCL) earlier this year to provide a system for on-line service descriptions for SOAP-based applications. This work in SCL, in addition to other protocols and related work from IBM and Ariba, has pretty much been phased into WSDL.

Just before WSDL emerged, a consortium of 36 companies, including IBM, Ariba, and Microsoft, launched the Universal Description, Discovery and Integration (UDDI) system (see Resources), an initiative to provide a standard directory of on-line business services with an elaborate API for querying the directories and service providers.
Microsoft kept itself busy in the area of web services description before WSDL emerged. It had created another entrant, Discovery of Web Services (DISCO) (see Resources), which is in now in limbo, outside Microsoft's official .NET strategic plan. DISCO describes how to find ("discover") SCL descriptions of services for a particular requirements. Frankly, reading the DISCO spec, it is hard to make heads or tails of the value it was supposed to provide, but whatever it had of use has since been sprinkled into UDDI and WSDL.

Parallel to Microsoft's efforts on SCL, IBM was creating the Network Accessible Service Specification Language (NASSL) (see Resources). One can also see that IBM threw its ideas fully into WSDL, and certainly its NASSL editors. IBM also got into the services discovery act with their Advertisement and Discovery of Services (ADS). There doesn't appear to have ever been a formal specification of ADS, though the Web Services Toolkit from IBM's alphaWorks project has a reference implementation of it (see Resources).

If you are thoroughly confused by now, you're in good company. More than one wag has quipped that XML is a specification with no other use but to spawn scads of other specifications. The formation of the UDDI group is supposed to help in the area of service description. Out of the current spaghetti should emerge a simple order, creating an overall protocol for deployment of web-based services. This will probably be in the form of separate but linked formats for service discovery, description, request/response protocol, request structure and data-typing, semantic discovery, and, of course, transport protocol. Figure 1 offers a suggested diagram representing this order and placing the various specifications I've mentioned accordingly. Hopefully, it will help clear up the landscape. Within this picture, WSDL handles the specific purpose of a description mechanism for services.

**Figure 1. Service roles and interactions**

![Service roles and interactions diagram](image)

**Sample WSDL document**

Let's look at how WSDL works with SOAP through the following example. Let us say we are the entrepreneurs behind the imaginary company snowboard-info.com, an intrepid snowboarding industry database providing a service that allows others to query endorsements from snowboard manufacturers. A client can send a request to retrieve this information from a server using a SOAP request like the one in **Listing 1**. In natural language, **Listing 1** encapsulates the question "Which professional snowboarder endorses the K2 FatBob?"
Listing 1. A SOAP 1.1 Request

```
POST /EndorsementSearch HTTP/1.1
Host: www.snowboard-info.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 261
SOAPAction: "http://www.snowboard-info.com/EndorsementSearch"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:GetEndorsingBoarder xmlns:m="http://namespaces.snowboard-info.com">
            <manufacturer>K2</manufacturer>
            <model>Fatbob</model>
        </m:GetEndorsingBoarder>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In response, the server can send the SOAP 1.1 response (sans HTTP header) message for the foregoing request as shown in Listing 2. In natural language, it encapsulates the simple string response "Chris Englesmann".

Listing 2. A SOAP 1.1 Response

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:GetEndorsingBoarderResponse xmlns:m="http://namespaces.snowboard-info.com">
            <endorsingBoarder>Chris Englesmann</endorsingBoarder>
        </m:GetEndorsingBoarderResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Now the overall structure of requests, the relevant data types, the schema of the XML elements used, and other such matter are left to the trading partners by the SOAP specification itself. WSDL provides a standard for service specification that unites the types of requests and the requirements needed to process them.

In order to get all the hot snowboarding portals and discussion sites hooked up to our system, we might want to define WSDL communications ports. We do so by releasing the WSDL description of our point of service as shown in Listing 3. A WSDL description for a Snowboarding endorsement query

First, a bit of reassurance. Listing 3 may seem long, but WSDL is actually quite simple. Our sample WSDL document not only uses nearly every facet of WSDL, it also has a hefty chunk of XML Schema and also takes advantage of the SOAP binding to WSDL. This last portion, though presented in the same service description, is technically an extension to the standard service description.

The whole thing is enclosed in the `<definitions>` element that describes a set of related services. The `<types>` element allows the specification of low-level data-typing for the message or procedure contents. Different mechanisms are permitted through namespace extensibility, but XML schemas are likely to be the choice for most users, and is used in our example. This specifies a simple element content model that you can see matches the sample exchange in Listing 1 and Listing 2.
WSDL provides a system for importing data-type specifications located as separate resources, and there could be several such resources in cases of complex messages in multiple usage domains.

The `<message>` element defines the data format of each individual transmission in the communication. In our case, one message represents the `EndorsingBoarder` request and the other the response. In our example, this is a simple statement that the body of the message is a particular element from the schema in the types section. The breaking of a transmission into message parts depends on the logical view of the data. For instance, if the transmission is a remote procedure call, the message might be divided into multiple parts, one of which is the procedure name and meta-data and the rest of which are the procedure parameters. There is naturally a relationship between the granularity of the data-typing and the break-down of the message into parts.

The `<portType>` element groups messages that form a single logical operation. For instance, in our case, we can have an `EndorsingBoarder` request which triggers an `EndorsingBoarder` response, or in case of error or exception, an `EndorsingBoarderFault`. This particular exchange is grouped together into a WSDL `port type`. As you can see, the relationship to messages is made by qualified name reference.

There are only four forms of operations with built-in support in WSDL: `one-way`, `request-response`, `solicit-response`, and `notification`. The latter two are simply the "inverse" of the first two, the only difference being whether the end point in question is on the receiving or sending end of the initial message. Basically, WSDL supports unidirectional (one-way and notification) and bidirectional (request-response and solicit-response) port types. Faults are only supported in the bidirectional port types, unlike the CORBA model -- I'll leave the inevitable controversy between the two approaches right there for now.

The WSDL document so far has moved from the concrete and physical (data typing) to the abstract and logical (messages and port types), with some reference between the two. The `<binding>` element is the bit that firmly provides the connection between logical and physical model. In this case, it takes the operation we have defined through the abstract port type and connects it to a concrete description of how it is transmitted through SOAP. Here is where we come to the SOAP extensions to WSDL I mentioned earlier. WSDL also provides bindings to bare-metal HTTP and MIME, and full extensibility to other protocols.

Our sample binding specifies the `GetEndorsingBoarderPortType` as having the SOAP "style" of `Document`. The style can be `RPC` or `Document`, the former indicating a more procedural bent to the communication and the latter a message-exchange direction. Of course, the dividing line between these is quite broad, and I can imagine much fruitless discussion over whether a given port type is one or the other. My bias in this debate is to use `Document` nearly everywhere.

Our binding also specifies the network transport as HTTP -- SOAP can be transmitted by other means, such as SMTP. The `<soap:operation>` elements get down to the grit, mapping the individual messages in the port type to definition of the SOAP transmissions that actuate them. Note that we specify a SoapAction, which is required for SOAP over HTTP. The given value
must be used in the HTTP headers of the actual messages in order to signal the "intent" of the message. This will supposedly allow intelligent proxying and firewalling of SOAP traffic some day.

The final element, `<service>`, defines a physical location for a communication end-point. It uses the port type and binding specified earlier, and basically gives the web address or URI for a particular provider of the described service. Naturally, in our example, it is the address where we have set up our SOAP server to traffic in snowboard product endorsement queries.

However, what if once having launched the service, it turned out to be a hit with users, and traffic begins to overwhelm our server? We might decide to set up a mirror, perhaps in Europe. In this case, the service is exactly the same, but we provide a separate URI from which it can be obtained. In the WSDL scheme of things, all we’d have to do to make this happen is modify our WSDL document to add another `<service>` element such as in Listing 4.

**Listing 4. An alternative `<service>` element for handling multiple sites.**

```xml
<service name="EndorsementSearchEuropeanService">
  <documentation>snowboarding-info.com Endorsement Service European Mirror</documentation>
  <port name="GetEndorsingBoarderPort"
    binding="es:EndorsementSearchSoapBinding">
    <soap:address location="http://www.snowboard-info.co.uk/EndorsementSearch"/>
  </port>
</service>
```

Notice the different service name and address. Now any users who find this WSDL document through whatever means of service discovery will have two options for where to make the actual request.

A few general comments on our WSDL example. You can see that WSDL leans heavily on XML namespaces. The XML namespaces given in the `<definition>` element's `targetNameSpace` attribute is by default attached to all the names used for the other top-level WSDL elements. Developers can use qualified names to refer to these elements using prefixes from the particular namespace declarations in scope. Note that the default namespaces are not applied to un-prefixed names within WSDL attributes. This is consistent with other places where the XML namespaces mechanism has been borrowed for use in disambiguating names in the character data of XML specifications. XML namespaces are also used to connect WSDL elements (and elements from binding extensions) to the data-typing provided in the `<types>` element. In our example, we use the default namespace, `http://schemas.xmlsoap.org/wsdl/`, to indicate the official elements of WSDL. However, the spec explicitly leaves wide open the option of extending the core elements using elements in other namespaces.

Overall this example is quite simple. It describes communication consisting of short SOAP transmissions, with two input strings and one output in each operation. WSDL could just as easily define multiple port types consisting of a myriad of messages that use the full, extraordinary range of XML Schemas. Then again, at least in the short term, more simple communications between XML service providers and users is more likely to succeed.
**Wrap-up, and the next topic**

WSDL does a good job of covering its ground. It might seem modest, but is an important building-block for XML-based commerce. WSDL is quite simple, which is no mean accomplishment considering the companies that worked on it and the political dynamics in which it emerged. IBM, Microsoft and Ariba seem to have provided a good example of how things can be done correctly between multiple vendors when building a specification.

WSDL borrows very thoughtfully from other efforts to avoid re-inventing the wheel. It provides strong extensibility through XML namespaces, and gives many tools to improve its usefulness for describing quite complex services. In all, even though one is always skeptical of specifications that spring into the public's hands fully formed as "1.0", it does seem that there is little to nit-pick about.

One thing WSDL leaves unmentioned is its potential for integration with RDF. As the W3C's oft-mentioned "semantic web" expands, on-line service description could benefit from all that RDF has to offer in the form of networks of rating and trust. In the next article on WSDL, I shall explore how to use RDF with WSDL.
Resources

- Take a look at the Web Distributed Data eXchange (WDDX) by Allaire; it provides a simple protocol for XML-message-based communications.
- Get familiar with Information Content Exchange (ICE), a protocol for the description of distributed content.
- The RDF Site Summary (RSS) provides a very simple specification for describing the content in on-line resources.
- Read the W3C Simple Object Access Protocol (SOAP) 1.1.
- Go to the WC3 compilation of XML protocol proposals
- Read the complete Web Interface Definition Language (WIDL) document.
- Review the WSDL specification.
- Visit the Universal Description, Discovery and Integration (UDDI) site to get a complete understanding of the project.
- Learn about using for Discovery of Web Services (DISCO).
- Download the PDF document specifying IBM's Network Accessible Service Specification Language (NASSL).
- The Web Services Toolkit package from alphaWorks, implements Advertisement and Discovery of Services (ADS).
About the author

Uche Ogbuji

Uche Ogbuji is a consultant and co-founder of Fourthought Inc., a consulting firm specializing in XML solutions for enterprise knowledge management applications. Fourthought develops 4Suite, the open source platform for XML middleware. Mr. Ogbuji is a Computer Engineer and writer born in Nigeria, living and working in Boulder, Colorado, USA. You can reach him at uche@ogbuji.net.

© Copyright IBM Corporation 2000
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)