Published on XML.com http://www.xml.com/pub/a/2001/04/04/webservices/index.html
See this if you're having trouble printing code examples

A Web Services Primer

By Venu Vasudevan

Table of Contents

Introduction

Looking back over the last six years, it is hard to imagine networked computing without the Web. The reason why the Web succeeded where earlier hypertext schemes failed can be traced to a couple of basic factors: *simplicity* and *ubiquity*. From a service provider's (e.g. an e-shop) point of view, if they can set up a web site they can join the global community. From a client's point of view, if you can type, you can access services. From a service API point of view, the majority of the web's work is done by 3 methods (`GET`, `POST`, and `PUT`) and a simple markup language. The web services movement is about the fact that the advantages of the Web as a platform apply not only to information but to *services*.

By "services", I don't mean monolithic coarse-grained services like Amazon.com, but, rather, component services that others might use to build bigger services. Microsoft's *Passport*, for instance, offers an authentication function exported on the Web. So hypothetically, an electronic newspaper like the Washington Post can avoid creating its own user authentication service, delegating it to Passport.

Oracle's dynamic services whitepaper provides other examples of component services that are reusable building blocks: currency conversion, language translation, shipping, and claims processing, A more formal definition of a web service may be borrowed from IBM's tutorial on the topic.

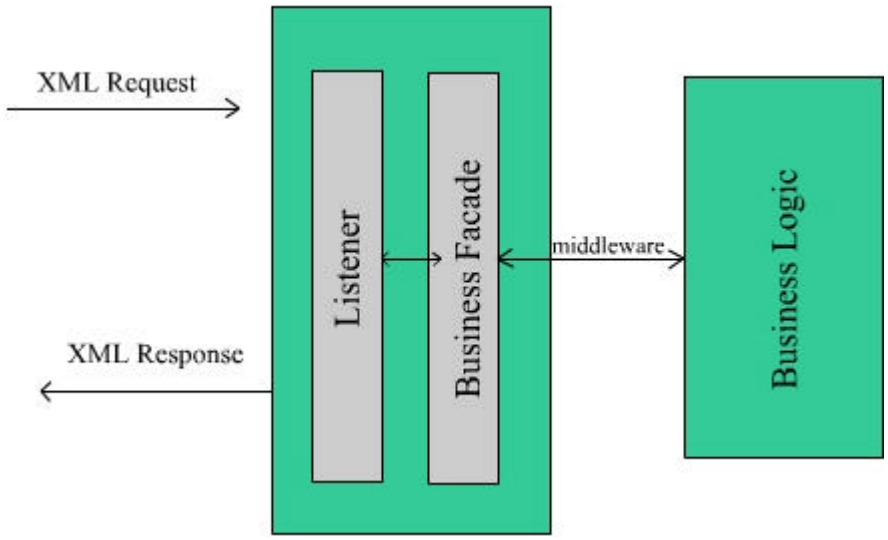
> *Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes...Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.*

IBM's web services tutorial goes on to say that the notion of a web service would have been too inefficient to be interesting a few years ago. But the trends like cheaper bandwidth and storage, more dynamic content, the pervasiveness and diversity of computing devices with different access platforms make the need for a glue more important, while at the same time making the costs (bandwidth and storage) less objectionable.

Why bother with the Web, you say, when I've got my favorite middleware platform (RMI, Jini, CORBA, DCOM etc.)? While middleware platforms provide great implementation vehicles for services, none of them is a clear winner. The strengths of the Web as an information distributor, namely simplicity of access and ubiquity, are important in resolving the fragmented middleware

world where interoperability is hard to come by. The Web complements these platforms by providing a uniform and widely accessible interface and access *glue* over services that are more efficiently *implemented* in a traditional middleware platform.

Viewed from an n-tier application architecture perspective, the web service is a veneer for programmatic *access* to a service which is then *implemented* by other kinds of middleware. Access consists of service-agnostic request handling (a listener) and a facade that exposes the operations supported by the business logic. The logic itself is implemented by a traditional middleware platform.



Generic Web Service Architecture

The Web Services Platform

So what is the web service platform? The basic platform is *XML plus HTTP*. HTTP is a ubiquitous protocol, running practically everywhere on the Internet. XML provides a metalanguage in which you can write specialized languages to express complex interactions between clients and services or between components of a composite service. Behind the facade of a web server, the XML message gets converted to a middleware request and the results converted back to XML.

Wait a minute, you say, access and invocation are only the bare bones, that would be like saying CORBA is only IDL plus remote procedure calls. What about the platform support services -- discovery, transactions, security, authentication and so on -- the usual raft of services that make a platform a platform? That's where you step up to the next level.

The Web needs to be augmented with a few other platform services, which maintain the ubiquity and simplicity of the Web, to constitute a more functional platform. The full-function web services platform can be thought of as *XML plus HTTP plus SOAP plus WSDL plus UDDI*. At higher levels, one might also add technologies such as XAML, XLANG, XKMS, and XFS -- services that are not universally accepted as mandatory.

Below is a brief description of the platform elements. It should be noted that while vendors try to present the emergent web services platform as coherent, it's really a series of in-development technologies. Often at the higher levels there are, and may remain, multiple approaches to the same problem.

- SOAP (remote invocation)
- UDDI (trader, directory service)

- WSDL (expression of service characteristics)
- XLANG/XAML (transactional support for complex web transactions involving multiple web services)
- XKMS (XML Key Management Specification) - ongoing work by Microsoft and Verisign to support authentication and registration

SOAP

SOAP is a protocol specification that defines a uniform way of passing XML-encoded data. In also defines a way to perform remote procedure calls (RPCs) using HTTP as the underlying communication protocol.

SOAP arises from the realization that no matter how nifty the current middleware offerings are, they need a WAN wrapper. Architecturally, sending messages as plain XML has advantages in terms of ensuring interoperability (and debugging, as I can well attest). The middleware players seem willing to put up with the costs of parsing and serializing XML in order to scale their approach to wider networks.

Submitted in 2000 to the W3C as a Note by IBM, Microsoft, UserLand, and DevelopMentor, the further development of SOAP is now in the care of the W3C's XML Protocols Working Group. This effectively means that SOAP is frozen and stable until such time as the W3C Working Group delivers a specification.

See also Don Box's Brief History of SOAP.

UDDI (Universal Description, Discovery and Integration Service)

UDDI provides a mechanism for clients to dynamically find other web services. Using a UDDI interface, businesses can dynamically connect to services provided by external business partners. A UDDI registry is similar to a CORBA trader, or it can be thought of as a DNS service for business applications. A UDDI registry has two kinds of clients: businesses that want to publish a service (and its usage interfaces), and clients who want to obtain services of a certain kind and bind programmatically to them. The table below is an overview of what UDDI provides. UDDI is layered over SOAP and assumes that requests and responses are UDDI objects sent around as SOAP messages. A sample query is included below.

Information	**Operations**	**Detailed information** (supported by lower-level API)
White pages: Information such as the name, address, telephone number, and other contact information of a given business	**Publish**: How the provider of a Web service registers itself.	**Business information**: Contained in a *BusinessEntity* object, which in turn contains information about services, categories, contacts, URLs, and other things necessary to interact with a given business.
Yellow pages: Information that categorizes businesses. This is based on existing (non-electronic) standards	**Find**: How an application finds a particular Web service.	**Service information**: Describes a group of Web services. These are contained in a *BusinessService* object

Green pages: Technical information about the Web services provided by a given business.	**Bind**: How an application connects to, and interacts with, a Web service after it's been found	**Binding information**: The technical details necessary to invoke a Web service. This includes URLs, information about method names, argument types, and so on. The *BindingTemplate* object represents this data. **Service Specification Detail**: This is metadata about the various specifications implemented by a given Web service. These are called *tModels* in the UDDI specification

There is no near-term plan in UDDI to support full-featured discovery (e.g. geography-limited searches or bidding and contract negotiation supported by vendors like eLance). UDDI expects to be the basis for higher level services supported by some other standard. There are plans for UDDI to support more complex business logic, including support for hierarchical business organizations. UDDI has fairly broad support; IBM, Ariba, and Microsoft are driving it. It's not yet an open standard.

UDDI Example

Query: The following query, when placed inside the body of the SOAP envelope, returns details on Microsoft.

```
<find_business generic="1.0" xmlns="urn:uddi-org:api">
       <name>Microsoft</name>
</find_business>
```

Result: detailed listing of <businessInfo> elements currently registered for Microsoft, which includes information about the UDDI service itself.

```
<businessList generic="1.0"
 operator="Microsoft Corporation"
 truncated="false"
 xmlns="urn:uddi-org:api">
 <businessInfos>
  <businessInfo
      businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3">
   <name>Microsoft Corporation</name>
   <description xml:lang="en">
          Empowering people through great software -
          any time, any place and on any device is Microsoft's
          vision. As the worldwide leader in software for personal
          and business computing, we strive to produce innovative
          products and services that meet our customer's
   </description>
    <serviceInfos>
     <serviceInfo
        businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
        serviceKey="1FFE1F71-2AF3-45FB-B788-09AF7FF151A4">
      <name>Web services for smart searching</name>
     </serviceInfo>
    <serviceInfo
        businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
        serviceKey="8BF2F51F-8ED4-43FE-B665-38D8205D1333">
      <name>Electronic Business Integration Services</name>
    </serviceInfo>
    <serviceInfo
```

```
          businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
          serviceKey="611C5867-384E-4FFD-B49C-28F93A7B4F9B">
            <name>Volume Licensing Select Program</name>
      </serviceInfo>
          <serviceInfo
          businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
          serviceKey="A8E4999A-21A3-47FA-802E-EE50A88B266F">
            <name>UDDI Web Sites</name>
      </serviceInfo>
      </serviceInfos>
    </businessInfo>
  </businessInfos>
 </businessList>
```

by Venu Vasudevan

WSDL (Web Services Definition Language)

WSDL provides a way for service providers to describe the basic format of web service requests over different protocols or encodings. WSDL is used to describe *what* a web service can do, *where* it resides, and *how* to invoke it. While the claim of SOAP/HTTP independence is made in various specifications, WSDL makes the most sense if it assumes SOAP/HTTP/MIME as the remote object invocation mechanism. UDDI registries describe numerous aspects of web services, including the binding details of the service. WSDL fits into the subset of a UDDI service description.

WSDL defines services as collections of network endpoints or *ports*. In WSDL the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions of messages, which are abstract descriptions of the data being exchanged, and port types, which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A port is defined by associating a network address with a reusable binding; a collection of ports define a service. And, thus, a WSDL document uses the following elements in the definition of network services:

- Types -- a container for data type definitions using some type system (such as XSD).
- Message -- an abstract, typed definition of the data being communicated.
- Operation -- an abstract description of an action supported by the service.
- Port Type -- an abstract set of operations supported by one or more endpoints.
- Binding -- a concrete protocol and data format specification for a particular port type.
- Port -- a single endpoint defined as a combination of a binding and a network address.
- Service -- a collection of related endpoints.

So, in plain English, WSDL is a template for how services should be described and bound by clients.

In what follows, I've described a stock quote service advertisement and a sample request/response pair for the service, which seeks the current quote on Motorola (ticker: MOT).

Service Advertisement

```
<?xml version="1.0"?>
  <definitions name="StockQuote"
               targetNamespace="http://example.com/stockquote.wsdl"
               xmlns:tns="http://example.com/stockquote.wsdl"
               xmlns:xsd1="http://example.com/stockquote.xsd"
               xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
               xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/1999/XMLSchema">
     <element name="TradePriceRequest">
      <complexType>
       <all>
        <element name="tickerSymbol" type="string"/>
       </all>
      </complexType>
     </element>
     <element name="TradePrice">
      <complexType>
       <all>
        <element name="price" type="float"/>
       </all>
      </complexType>
     </element>
    </schema>
   </types>

  <message name="GetLastTradePriceInput">
     <part name="body" element="xsd1:TradePrice"/>
  </message>
  <message name="GetLastTradePriceOutput">
     <part name="body" element="xsd1:TradePriceResult"/>
  </message>

  <portType name="StockQuotePortType">
   <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
   </operation>
  </portType>

  <binding name="StockQuoteSoapBinding"
           type="tns:StockQuotePortType">
  <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
     <soap:operation
      soapAction="http://example.com/GetLastTradePrice"/>
       <input>
         <soap:body use="literal"
          namespace="http://example.com/stockquote.xsd"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
       </input>
      <output>
       <soap:body use="literal"
         namespace="http://example.com/stockquote.xsd"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
     </operation>
  </binding>

  <service name="StockQuoteService">
   <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
     <soap:address location="http://example.com/stockquote"/>
    </port>
```

```
  </service>

 </definitions>

  <binding name="StockQuoteServiceBinding"
   type="StockQuoteServiceType">
    <soap:binding style="rpc"
     transport="http://schemas.xmlsoap.org/soap/http"/>
         <operation name="getQuote">
          <soap:operation
       soapAction="http://www.getquote.com/GetQuote"/>
        <input>
         <soap:body type="InMessageRequest"
          namespace="urn:live-stock-quotes"
          encoding="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
        <output>
         <soap:body type="OutMessageResponse"
          encoding="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
      </operation>
    </binding>
    <service name="StockQuoteService">
     <documentation>My first service
         </documentation>
      <port name="StockQuotePort"
       binding="tns:StockQuoteBinding">
       <soap:address location="http://example.com/stockquote"/>
      </port>
     </service>
</definitions>
```

A SOAP enveloped request to the StockQuote service

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<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:GetLastTradePrice
     xmlns:m="Some-URI">
      <symbol>MOT</symbol>
    </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A SOAP enveloped response to the StockQuote service

```
HTTP/1.1 200 OK Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<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:GetLastTradePriceResponse
   xmlns:m="Some-URI">
   <Price>14.5</Price>
  </m:GetLastTradePriceResponse>
```

```
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

by Venu Vasudevan

XLANG

The traditional notion of a database transaction is an atomic, that is, all-or-none action; either the entire action happens or it doesn't. Providing this kind of guarantee in a distributed infrastructure involves an expensive process called *two-phase* commit. An alternative optimistic model has been proposed in database research (originally called *sagas* and proposed by Hector Garcia-Molina), where actions have explicit *compensatory* actions which negate the effect of the action. In the real world of actions, the existence of compensatory actions is quite common. For instance if I debit a credit card $52, the compensatory action is to credit the credit card $52. If I sent out an e-mail saying "you'll get the product you ordered in seven days", the compensatory action is to send an e-mail saying, "oops, it's going to take longer". XLang is a notation for expressing the compensatory actions for any request that needs to be undone. The web services infrastructure can leverage XLang specifications to perform complex undo operations.

XAML

Transaction Authority Markup Language (XAML) provides a more traditional two-phase commit style transactional semantics over web services. A business-to-business web transaction to purchase benzene follows -- a working example from the XAML specification. XAML does not completely restrict itself to two-phase commits, and it leaves open the possibility that some action "undos" will be compensatory actions like XLang. While two-phase commit is clearly useful in enterprise integration, a number of web transactions (e.g business-to-consumer transactions) are well captured by the computationally cheaper *compensatory action* model. Until XAML makes compensatory actions a first class citizen of their model, it would seem that XLang has ample justification to exist.

Scenario

The following scenario demonstrates a business-level transaction involving a set of web services that would utilize XAML. Consider an industrial company that purchases benzene from a chemical manufacturer on the Web. In order for the buyer to purchase the benzene, she requires additional value-added services provided by third parties, such as shipping with specific delivery terms, payment financing, casualty insurance, and government compliance for safe transport. The buyer will not agree to the purchase of benzene until all of these services are available, and until all of them meet her requirements. She will purchase all of them or none of them. In other words, all of these related requirements need to be satisfied in order for the business transaction to be completed.

The software providing the top-level business transaction needs to coordinate with each of the participating web services. These include (1) the chemical provider's inventory system; (2) an insurance policy service to insure the product being shipped; (3) a financing service to ensure payment according to vendor terms; (4) a transportation service to guarantee timely shipment and delivery of product; and (5) a regulatory service to ensure compliance with government safety requirements.

XKMS (XML Key Management Specification)

XKMS is an effort by Microsoft and Verisign to integrate PKI and digital certificates (which are used for securing Internet transactions) with XML applications. The key idea is to delegate the signature processing to a *trust server* on the Web, so that thin or mobile clients don't have to carry around the smarts to do all this themselves. XKMS relies on the XML Signature specification already being worked on by W3C and on anticipated work at W3C on an XML encryption specification.

XKMS consists of two parts: the XML Key *Information* Service Specification (X-KISS) and the XML Key *Registration* Service Specification (X-KRSS). The X-KISS specification defines a protocol for a trust service that resolves public key information contained in XML-SIG elements. The X-KISS protocol allows a client of such a service to delegate part or all of the tasks required to process `<ds:KeyInfo>` elements (information about the key signer). A chief objective of the protocol design is to minimize the complexity of application implementations by allowing them to become clients, thereby shielded from the complexity and syntax of the underlying PKI used to establish trust relationships. These may be based upon a different specification such as X.509/PKIX, SPKI, or PGP. X-KRSS describes how public key information is registered.

While there are no inviolable ties in these proposals to protocols and transports, the current XKMS specification relies on XML, SOAP, and WSDL.

Other initiatives in this area include S2ML (Security Services Markup Language) and AuthXML, which are being unified under the auspices of OASIS's XML Security Services committee.

Other useful initiatives

The web services platform is an evolving ecosystem in which Darwinian processes are still at work. As with all things Darwinian, there is constant evolution, bio-diversity, competition, and, yes, even confusion. The list below is a small sample of such complementary or competing initiatives.

ADS (Advertisement and Discovery of Services Protocol)

Given the existence of UDDI registries, ADS asks the question "how do I facilitate the building of a UDDI ***crawler*** that can pull UDDI advertisements off the Web, without people having to push ads to the registry?" Furthermore, while ADS accepts WSDL as the XML format for a service, it also wants to deal with discovering services that don't have the XML capabilities to build WSDL descriptions. For the XML world, it standardizes on a *svcsadvt.xml* file placed in the root of a web server, which then collectively advertises all the services available on that web site. This takes away the burden on each service to advertise itself, and it provides service crawlers a single place to look for advertisements. For the mom and pop e-shops of the world that want to advertise their services without the XML overhead, ADS proposes an augmented HTML META tag with `name=<serviceDescriptionLocation >`and `content=<valid URL of document containing service advertisements>`. In case of HTML based service crawling, the crawler makes some conclusions about the service properties based on the traversal context.

XFS

The XMethods filesystem service enables you to post and read files via a SOAP interface. This system enables developers to create services that utilize centralized, persistent data. Ideally, this type of filesystem can be used to centralize the storage of information which can be accessed by multiple nodes. For example, one could use this space to support automatic patch updates. XFS provides a client tool that integrates the XFS web service into a Windows Explorer shell. Windows Explorer is then integrated with the XML-SOAP-based file system. XFS is an open-source

initiative by xmethods.com, the momentum of which is unclear. However, the idea is technically attractive.

Related Reading

- Web Services - The Web's next revolution
- A Platform for Web Services
- .Net has XML on its Menu
- Oracle9i Dynamic Services
- XLANG from Microsoft
- BEA touts support for Web services standards
- Introduction to XSI and XSD
- XKMS
- XMethods - lists publicly available web services and their SOAP access interfaces
- XML Trust Services
- SOAP Developer's Site
- ADS - The Advertisement and Discovery of Services (ADS) protocol for Web services
- ebXML
- UDDI4J: Matchmaking for Web Services
- IBM's Web Services Toolkit
- An introduction to WSDL for SOAP programmers
- W3C's XML protocols matrix
- Understanding ebXML, UDDI and XML/edi
- Jon Bosak's (Sun) vision of a service grid
- W3C mailing list discussion in response to Jon Bosak's comments on ebXML vs SOAP vs UDDI
- XAML

Disclaimer: *The opinions expressed herein are those of the author and do not necessarily reflect the opinion, position, or stance of Motorola, Inc. with regard to this subject.*