Web services: SOAP, WSDL, UDDI

A Web service is a software system that supports interoperable machine-to-machine interaction.

- **Interaction** means that more than one application is involved.
- **Interoperable** means that applications can operate with one another without sharing the same platform, operating system, programming language, etc.

Web services are largely delivered by a troika of protocols: SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery, and Interoperability).

These three build on XML (the metalanguage for the representation) and HTTP, the transport protocol.

In overview,

- SOAP defines a uniform way of passing XML-encoded data.
- WSDL allows service providers to specify what a web service can do, where it resides, and how to invoke it.
- UDDI provides a mechanism for clients to dynamically find other Web services.

In SOAP messages, the name of the service request and the input parameters take the form of XML elements.

WSDL describes the data types and structures for Web services, and tells how to map them into the messages that are exchanged.

UDDI provides a repository for Web-services descriptions. An UDDI registry can be searched on various criteria to find all kinds of services offered by businesses.
SOAP

Let’s first take a look at SOAP. Take a very simple search request:

http://www.google.com/search?q=laptop+trackball

This invokes Google with the search terms “laptop” and “trackball”. Google returns lists of pages that entirely depend on matching the given text strings.

XML provides a lot of advantages over simple text-based search. It allows the preceding request to be refined and contained in an XML document:

```xml
<SOAP-ENV:Body>
  <s:SearchRequest
    xmlns:s="www.xmlbus.com/SearchService">
    <p1>laptop</p1>
    <p2>trackball</p2>
    <p3>maxWeight 4.0</p3>
    <p4>OS MacOS</p4>
  </s:SearchRequest>
</SOAP-ENV:Body>
```

What are some of the advantages of sending the request in XML?

- 
- 

This example also illustrates the use of an XML namespace (xmlns:).

A SOAP message\(^1\) contains four parts:

- The *envelope*—the top-level XML element
- The *header*—metadata about the SOAP message

\(^1\) This example is taken from Eric Newcomer's *Understanding Web Services: XML, WSDL, SOAP, and UDDI*
• The **body**—the data that is being sent for the receiver to process.
• The **attachment**—data, usually binary, that shouldn’t be represented in the body but is needed for processing.

Here is an example SOAP message.

```xml
<env:Envelope
xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    Headers
  </env:Header>
  <env:Body>
    Body
  </env:Body>
</env:Envelope>
```

Here is an example of a SOAP header:

```xml
<env:Header>
  <n:broadcastService xmlns:n="http://www.xmlbus.com/broadcastServices">
    <n:list>PDA, Cell, Email, VoiceMail, IM</n:list>
  </n:broadcastService>
</env:Header>
```

The header block is optional; it defines attributes of the message.

The body block contains the message itself:

```xml
<env:Body>
  <m:Function
    xmlns:m="http://www.xmlbus.com/broadcastServices/send">
    <m:message>
      Eric, you are late for the concall again!
    </m:message>
  </env:Body>
```
(Note: The Web-service integration package XMLBus is being replaced by IONA’s Artix, http://www.iona.com/products/artix/xmlbus.htm.)

In this example, the meeting reminder message will be broadcast to the device list by the send service located at the address defined by the URL: http://www.xmlbus.com/broadcastServices.

Separate namespaces (broadcastService and function) are used to qualify the element and attributes for each part of the message.

This simple one-way message can be bound to HTTP using POST:

Request header:
POST /broadcastService HTTP/1.1
Host: www.xmlbus.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<?xml version='1.0' ?>
... SOAP request document

There’s no response in this case, but if there were one, it would look like this:

Response header:
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<?xml version='1.0' ?>
... SOAP request document

The HTTP header indicates that the document is being sent to the broadcastService at the indicated URL.

When the program implementing the broadcastService receives the message, it will decode the body block and execute the send
Although SOAP is usually used over HTTP, it can be used over other protocols as well, e.g., SMTP/POP.²

SOAP messages are exchanged in various Message Exchange Patterns (MEPs).

- Request-Response: A request SOAP message is sent, and a response SOAP message is returned.
- Request: No response is needed.
- Response: Response is to a non-SOAP request of some sort.

**SOAP Attachment**

An attachment is data in the SOAP message that isn’t part of the XML body.

An example would be a JPEG photo of an accident that must be submitted as part of an insurance claim. A SOAP attachment can take various forms.

- Encoded as part of an HTTP POST (SOAP with Attachments (MIME), Direct Internet Message Exchange DIME))
- A URI that points to an external resource that be accessed with HTTP GET.

Here is an example of a SOAP attachment.

```plaintext
POST /insuranceClaims HTTP/1.1
HTTP headers

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim061400a.xml@claiming-it.com>

<?xml version='1.0' ?>
SOAP Message
```

² The following discussion is adapted from a 2003 guest lecture by Michael Thomas, mdthomas@ibiblio.org.
How SOAP Relates To Other Standards

WSDL is the interface definition for SOAP. UDDI is the registry for SOAP.

There are other standards built on top of SOAP:

- Security
- Transactions
- BPEL4WS (Business Process Execution Language for Web Services)
- WS-Routing

WSDL

The Web Services Definition Language was created to describe the formats and protocols of a Web service in a uniform way.

WSDL elements describe the data and the operations to be performed on it.

Both are described in terms of XML schemas.

WSDL is usually used with SOAP.

In order to communicate, both sender and receiver must have access to the same XML schema.

If both are using the same schema, any implementation of the Web service can be used, e.g., CORBA, COM, EJB.

The following WSDL components exist:
<definitions>
  <types>
    definition of types
  </types>
  <message>
    definition of a message
  </message>
  <portType>
    definition of a port
  </portType>
  <binding>
    definition of a port
  </binding>
</definitions>

<types> refers to data types used in the messages. These may be defined in XML schemas or in some other way.

<portType> is analogous to a function library. Defines the Web service, operations that can be performed and the messages involved. Can be used in multiple transports through various bindings. Examples include …

- One-way operation: No response will be sent.
- Request-response operation: Accept request and send a response.
- Solicit response: Requestor will wait for a response.
- Notification: Operation can send a message but won’t wait for a response.

<binding> is the message format and protocols for the Web services.

Request-response operation

<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>
<message>
<portType name="glossaryTerms">  
<operation name="getTerm">
  <input message="getTermRequest"/>
  <output message="getTermResponse"/>
</operation>
</portType>

One-way operation

<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
<operation name="setTerm">
  <input name="newTerm" message="newTermValues"/>
</operation>
</portType>

WSDL binding

This signifies that the binding is to the SOAP protocol format—envelope, header, and body.

<binding type="glossaryTerms" name="b1">  
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />  
<operation>
  <soap:operation
    soapAction="http://example.com/getTerm"/>
  <input>
    <soap:body use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
  </output>
</operation>
</binding>
UDDI

UDDI stands for Universal Description, Discovery and Integration. It provides a directory of Web-services interfaces (written in WSDL).

It uses SOAP for queries and modifications.

There are global UDDI repositories. The largest is at www.uddi.org.

UDDI is the Yahoo/Google of Web services. An enterprise can also have local repositories (just like you can have local Yahoos/Googles).

How UDDI works.

• The world creates different classifications of services.
• Businesses submit their Web services to the registry.
• A Web service is catalogued.
• Potential web services customers query UDDI for what they need.
• A new business relationship is born.

Core UDDI Data Types

• businessEntity: A business providing a service
• businessService: A service
• bindingTemplate: Invocation details
• tModel: used to specify that a service complies with the constraints of a particular, pre-defined model. The UDDI tModel is crucial for interoperability in services, because without standard, pre-defined models every interface must be separately defined and software independently implemented.
• publisherAssertion: relationships between businessEntities.

UDDI & WSDL

• businessService: created for each Web service
• tModel: Points to the WSDL
• bindingTemplate: the unique URL access points
• You’ll probably use an API (e.g., JAXR, the Java API for XML Registries) to access UDDI.

**UDDI business entity**

```xml
<businessEntity businessKey="35AF7F00-1419-11D6-A0DC-000C0E00ACDD"
    authorizedName="0100002CAL"
    operator="www-3.ibm.com/services/uddi">
  <name>BooksToGo</name>
  <description xml:lang="en">
    The source for all professional books
  </description>
  <contacts>
    <contact>
      <personName>Ramesh Mandava</personName>
      <phone>(877)1111111</phone>
    </contact>
  </contacts>
</businessEntity>
```

**Summary**

Web services is about interoperability and providing SOA services over the Internet.

As with Java, there are better, faster ways to do any particular thing Web services can do. But web services can provide a level of interoperability other systems can’t.

Web services can be over-used. Don’t use Web Services inside distributed apps when you could use EJBs or .NET remoting.

**Best way to learn:** Read the specifications, play with available tools and example.