



# WEB SERVICES

**Rob Nelson**

**TESSELLA SUPPORT SERVICES PLC**

Issue V1.R1.M0

February 2003



## WEBSERVICES

### **What is a Web Service?**

There has been a great deal of talk about Web Services. Much of this has been due to Microsoft, which sees them as central to its latest technology, .NET. There are various definitions of what a web service is, but they all amount to the same thing:

A Web Service is an application component, which is accessible programmatically across the web using standard protocols.

A Web Service enables a company to make a service accessible to programs running elsewhere, in the same way that a web site could make the service available to remote users. A Web Service can contain many operations so, for example, an organisation might offer operations to create an account, check the availability of a facility and to book its use. Equally a Web Service could be internal to the company, for example providing a number of standard calculations. This would ensure that they are always performed in the same way and changes to the calculations would only ever be required in one place.

Web services can be linked together. An example could be environmental monitoring where sensors could make their data available as a web service. A flood prediction system could then use these services to obtain rainfall and river level measurements. If the flood prediction was itself published as a web service, then a second system further down the river could use the prediction as in input to its own prediction.

Another example might be access to online databases such as the various public domain genome databases. These are currently designed to be searched by people and so they are implemented in HTML. If however someone wishes to perform automated searches, URLs must be constructed to fake the process of filling in forms and the resulting HTML must be parsed to separate the information from the formatting and navigation code which often make up most of the web page.

This document describes Web Services from a technical point of view.

### **Web Services and XML**

Web Services rely heavily on XML and on numerous other standards, which are themselves based on XML. In fact the objective of Web Services is closely related to that of XML.

XML was originally intended to simplify data exchange. Different systems store their data in different, incompatible ways, so exchanging data means converting it. This is much easier if data is exchanged in a common format rather than dealing with every possible permutation of source and recipient. HTML is not a suitable common format because it is designed to make data intelligible to people, not to make it readable by programs. Therefore XML was devised as a means for exchanging information in a structured way.

XML is extensible, so for any data, a structure can be devised which preserves both the data fields and their relationships. XML however does not provide a mechanism for transferring the data. A company can convert its information into XML, but a Web Service provides a mechanism by which another company can ask for and receive that information. A Web Service can also enable actions to be performed, such as querying a database to find the information to be returned.

### **Why Web Services and Why Now?**

A Web Service is simply a means for a process on one machine to invoke procedures or methods on another machine. This is not a new problem, but it is one of growing interest. As Companies accumulate software applications, the need to make them interoperate increases, but if the applications are not insulated from each other, changing or re-implementing individual applications becomes harder. As the Internet becomes increasingly pervasive, and companies become used to trading over the Internet, so the attractions of automatically trading with suppliers and partners via the Internet becomes more obvious.

Many solutions to the problem already exist. Sun's RPC (remote procedure call) has existed for many years and Java contains RMI (remote method invocation) but they have never moved beyond specialist applications. CORBA (IIOP) and DCOM are both attempts to make RPC more user friendly. They are widely used for Intranet applications, but seldom across the Internet.

Web Services offer the prospect of creating distributed, loosely coupled applications, using standard protocols (XML, HTTP) and infrastructure (web servers and internet connections). CORBA was seen as complex and expensive and DCOM as proprietary, restricted to the Windows platform, and difficult to maintain. Web services however are being perceived as working across heterogeneous platforms and as small increments to existing knowledge and infrastructure. The resulting much wider take up will lead to Web Services becoming the de-facto standard for building loosely coupled applications.

Whether Web Service become a significant resource in the internet is still an open question, that they will become a useful tool in integrating applications across the enterprise seems certain.

### *HTTP and Firewalls*

A simple obstacle to using CORBA and DCOM across the Internet is the perceived rather than actual risk. The default action of most firewalls is to allow through only a small number of protocols regarded as useful and safe, such as email and HTTP. Other protocols are blocked, especially if the purpose of the protocol its to allow applications to run remotely. Thus anyone trying to offer a CORBA or DCOM based service has first to change the firewall protecting the server on which it runs. Then all potential customers must change their firewalls to match. In a large company the risks of incorrectly setting the firewall will be large and getting a new protocol allowed through can be extremely difficult. Worse, the potential user of the service will be unable to try out the service before asking for firewall changes.

The web however is accessible from almost everywhere so Web Services, built on top of HTTP, will not be blocked by firewalls. While HTTP is not secure, it is trivial to switch to the secure version: HTTPS.

For the service provider, using HTTP does not make Web Services any more secure than CORBA, DCOM or anything else. A web service could be created that allowed anyone to run any command on the web server, but this is not a new risk, the same could be done using cgi scripts for example. From the client point of view there *is* a real difference because HTTP is asymmetric – a web browser cannot act as a web server, and in the same way it is impossible for the client to accidentally provide a web service when trying merely to use one.

### **How Web Services Work**

For Web Services to work, there are a number of related problems, which must be solved. To take a trivial example, suppose company A produced a program for adding two numbers together and company B wanted to use this as part of its calculator program. At the lowest level, company A must have a protocol for receiving numbers and for returning the answers. At the next level up, company B needs to be able to find out what this protocol is. Before it can do that however, company B needs to find out that company A offers the service. Finally if such services are going to be created and used, it must be as easy as possible for company A to create and publicise the service and for company B to create a client able to use it.

To achieve this Web Services are assembled from a number of technologies which each addresses a part of the overall problem:

Service Discovery	UDDI		
Service Description	WSDL		
Message Formating	SOAP		
	XML		
Message Transport	HTTP	HTTPS	SMTP/JMS

HTTP and HTTPS are the standard web protocols. SMTP (Simple Mail Transfer Protocol) and JMS (Java Messaging Service) are standard protocols, which may be used for notification type services. The higher levels are specific to Web Services and are described below:

### SOAP

Web Services exchange information using HTTP, but they need to do so in an agreed way. The Web normally consists of pages written in HTML but for transferring information between programs, XML is preferred because it is more generic and easier to parse. By its nature XML is completely flexible, so some rules are needed for messages used in Web Services. SOAP (Simple Object Access Protocol) is a standard for wrapping up method names and parameters as XML.

A service for adding two numbers together might take a SOAP message like:

```
POST /webservices/maths/maths.ashx HTTP/1.1
```

```
Content-Type: text/xml
```

```
SOAPAction: urn:math#Add
```

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body>
    <s:add xmlns:s="http://www.nowhere.zz">
```

```
<x>2</x>
<y>2</y>
</s:add>
</soap:Body>
</soap:Envelope>
```

The reply would be a standard HTTP response containing XML, which would include the answer: 4.

```
HTTP/1.1 200 OK
Content-Type: text/xml
```

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body>
    <s:add xmlns:s="http://www.nowhere.zz">
      <answer>4</answer>
    </s:add>
  </soap:Body>
</soap:Envelope>
```

If required, the SOAP messages can have MIME attachments allowing documents, graphics etc to be passed.

Clearly this allows a service to be accessed as though it were a web page. However when producing the client, the developer has to know the format of the message it should send and how to interpret the reply. Supplying this information is the province of another XML based standard, WSDL.

## WSDL

WSDL (Web Services Description Language) is used to describe a Web Service. Given a WSDL document it is in theory possible to automatically generate a client capable of using the Web Service. In other words, a WSDL file is the Web Services equivalent of the IDL files used by DCOM or CORBA.

A WSDL document is written in XML. WSDL alone is not complete because although it defines methods and their parameters, it does not say how to actually invoke them. Instead WSDL can be extended to include any binding scheme, but the only extension currently defined is the bindings for SOAP.

To illustrate WSDL, the following is a WSDL document for the addition server described above.

A WSDL document contains a hierarchy of information. It starts with the normal XML header including definitions of namespaces:

```
<?xml version='1.0'?>
<definitions name='Math' xmlns='http://schemas.xmlsoap.org/wsdl/'
xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/' xmlns:xsd='http://www.w3.org/
2001/XMLSchema' xmlns:tns='urn:math' targetNamespace='urn:math'>
```

The types used in the SOAP message are defined in the types section:

```
<!-- XML Schema type definitions -->
<types>
  <xsd:schema targetNamespace="urn:math" elementFormDefault="unqualified">
    <xsd:complexType name="RequestType">
      <xsd:sequence>
        <xsd:element name="x" type="xsd:double"/>
        <xsd:element name="y" type="xsd:double"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ResponseType">
      <xsd:sequence>
        <xsd:element name="answer" type="xsd:double"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="Add" type="tns:RequestType"/>
    <xsd:element name="AddResponse" type="tns:ResponseType"/>
  </xsd:schema>
</types>
```

The messages section identifies the types sent in messages:

```
<!-- messages -->
<message name="AddMsg">
  <part name="parameters" element="tns:Add" />
</message>
<message name="AddResponseMsg">
  <part name="parameters" element="tns:AddResponse" />
```

```
</message>
```

The portType section names the operations available and says which messages they receive and return:

```
<!-- portType (interface) definitions -->
<portType name="Math">
  <operation name="Add">
    <input message="tns:AddMsg" />
    <output message="tns:AddResponseMsg" />
  </operation>
</portType>
```

Everything so far has been abstract. The bindings section gives the additional information needed to encode the messages on the wire, in this case as SOAP over HTTP:

```
<!-- bindings -->
<binding name="MathBinding" type="tns:Math">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Add">
    <soap:operation soapAction="urn:math#Add"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

Finally the services section defines the URL on which the service is available:

```
<!-- services -->
<service name="MathService">
  <port name="MathPort" binding="tns:MathBinding">
    <soap:address location="http://localhost/webservices/math/math.ashx"/>
  </port>
</service>
```

</definitions>

For a site offering a single service with a single operation the WSDL file looks very complex, but where there are multiple services, operations and even encodings, this structure is needed.

## **UDDI**

WSDL describes a service, but before the WSDL document can be used, the user must be able to find it. UDDI (Universal Description, Discovery and Integration) is a directory service, which allows clients to find web services. UDDI can be implemented locally on an intranet, or as a public directory. It provides the following types of data:

- Information on the company offering the service such as contact names.
- Classifications in terms of industry, service or geography.
- Technical information such as the location of the service and its WSDL file

UDDI is implemented and accessed as a Web Service.

## **Implementing a Web Service**

The preceding sections have described the protocols used to implement a Web Service and how they fit together. The final requirement is that Web Services need to be easy to implement and use.

A Web Service communicates using HTTP and therefore providing a Web Service requires a web server. From the point of view of the web server hosting it, a Web Service is just another script and it can be written using any of the technologies that the web server supports, e.g. normal CGI, JSP, Servlets, ASP, ISAPI or a combination of them. The Web Service can also take advantage of any support provided by the web server for things like transactions, sessions state, security, connection pooling etc.

A Web Service could be implemented manually as a collection of CGI scripts on a web server. Likewise a client could be written by hand, but SOAP messages and WSDL files are complex, verbose and do not make for exciting reading. However WSDL was designed to be written and read automatically. Numerous toolkits exist which can take a collection of objects and methods and generate code to turn them into a Web Service complete with WSDL file.

Likewise, it is possible to start from the WSDL file and automatically generate either the client stubs or the server skeleton. Thus the only programming required is the

internal logic of the service and client while all the communications have been handled automatically.

Taking the example WSDL file given above and using a toolkit to create an implementation of the server gives a number of files including the following Enterprise Java Bean:

```
package mypackage1.impl;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class AddBean implements SessionBean
{
    public void ejbCreate() { }
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbRemove() { }
    public void setSessionContext(SessionContext ctx) { }

    public double add(double x, double y)
    {
        return x + y; // Implementation added here
    }
}
```

Toolkits for implementing Web Services include Microsoft's .NET architecture, Oracle's JDeveloper and IBM's WSTK (demo) or WebSphere Application Server (product).

### **Problems**

Web Services are a new technology and the various standards on which they depend are in many cases still evolving. As a result there are a few problems with interoperability between different implementations:

#### *SOAP Encoding*

SOAP messages come in two not always compatible flavours.

Parameters passed by a Web Service need to be typed and XML types are specified using XML Schema (XSD). Unfortunately SOAP 1.0 was released when only part of XSD existed. Therefore SOAP had to define its own system known as "rpc/encoded".

When XSD was completed it looked nothing like rpc/encoded so SOAP 1.1 suggested using a new encoding “document/literal” based on XSD. These two are incompatible.

Different toolkits use one or the other (e.g. .NET defaults to document/literal while JDeveloper uses rpc/encoded). The WSDL file states which system a Web Service uses but some toolkits can only work with one system and not the other.

There are also other problems such as early implementations that use parameter order, not parameter name, or that fail to specify namespaces for certain elements.

The result is that seamless interaction between a server generated with one toolkit and a client generated with another is not yet guaranteed.

### *Interaction style*

In theory WSDL allows four types of interaction:

- Request/response (client->server->client) which allows the client to run a method on the server
- One way (client->server) which allows the client to notify the server of an event
- Solicit/response (server->client->server) which allows the server to query the client
- Notification (server->client) which allows the server to notify the client of an event.

There is nothing about SOAP messages to prevent them being passed in these different ways, but the problem comes because the underlying protocol is almost always HTTP and HTTP only allows one type of interaction, request/response. Thus in an environmental monitoring service, the server cannot use HTTP to notify the client when conditions change. Instead the client must poll the server to see if conditions have changed. Notification is possible if the SOAP message is sent using a messaging protocol such as SMTP or JMS but standards for this are still being developed (e.g. ebXML – electronic business XML), so additional coding may be required.

Another area awaiting standardisation occurs when the operations on a service must be used in a certain order. For example a WSDL file can list “login,” “buy” and “getPrice” as operations, but it cannot explain that “login” must be called before calling “buy” but is not required before calling “getPrice”.

### *SOAP action header*

For historical reasons SOAP messages often require an accompanying SOAPAction header containing a specific but arbitrary string. Fortunately WSDL contains a field to specify the action, whatever it may be, thereby making this anomaly invisible to the user.

### *A WSDL Standard*

WSDL is not a formal standard, merely a draft, which is still not synchronised with developments in XSD and SOAP. WSDL 1.2, which is under development, should resolve these problems and become a formal standard. Once that occurs and both toolkits and services adopt the standard, many interoperability problems will disappear.

### *Combining Web Services*

One area, which has still to be resolved, is how to combine Web Services. A Web Service can rely on its web server for things like transactions and user identification, but if the Web Service uses other Web Services, user identity may need to be propagated and all the services with transactions need to commit or rollback together. Some toolkits provide their own solutions to these problems but, until there is a standard mechanism, this will be another source of incompatibility. A number of standards are under development to address these problems.

## **Conclusions**

Web Services offer huge possibilities. In theory they permit the seamless integration of remote applications. They could be in different parts of the same company or Web Services could be used for automated business to business e-commerce. Currently there are some interoperability problems because standards are new and still evolving but the situation will improve.

**Tessella Support Services plc**  
**Creating Software for Science and Engineering**

Tessella's services range from feasibility studies, through system design, development, implementation and ongoing support. Our expertise includes:

Data Analysis Software  
Data Capture  
Simulation Software  
Advanced Graphics  
Systems Support  
Database Applications

**Other Technical Supplements available include:**

- |   |  |
|---|--|
| <input type="checkbox"/> Archiving of Electronic Info   | <input type="checkbox"/> Object Oriented Programming   |
| <input type="checkbox"/> Active Server Pages            | <input type="checkbox"/> Pocket PC                     |
| <input type="checkbox"/> Automated GUI Testing          | <input type="checkbox"/> Portable GUI Development      |
| <input type="checkbox"/> Bayesian Statistics            | <input type="checkbox"/> Printer Technology Guide      |
| <input type="checkbox"/> Beowulf Clusters               | <input type="checkbox"/> Real Time Systems             |
| <input type="checkbox"/> C++                            | <input type="checkbox"/> Regression Testing            |
| <input type="checkbox"/> Client-Server Technology       | <input type="checkbox"/> Security and the Internet     |
| <input type="checkbox"/> COM                            | <input type="checkbox"/> Simulation                    |
| <input type="checkbox"/> Computational Fluid Dynamics   | <input type="checkbox"/> Soft Computing                |
| <input type="checkbox"/> Computer Image Processing      | <input type="checkbox"/> Software Design Methodologies |
| <input type="checkbox"/> Decision Support Systems       | <input type="checkbox"/> Software Development Cycle    |
| <input type="checkbox"/> Electronic Data Capture        | <input type="checkbox"/> Software Documentation        |
| <input type="checkbox"/> Electronic Lab Notebooks       | <input type="checkbox"/> Software Portability          |
| <input type="checkbox"/> Excel                          | <input type="checkbox"/> Software Re-engineering       |
| <input type="checkbox"/> Extending the Life of Software | <input type="checkbox"/> Software Specification        |
| <input type="checkbox"/> Federal Drug Administration    | <input type="checkbox"/> SQL                           |
| <input type="checkbox"/> FORTRAN 90                     | <input type="checkbox"/> UNIX Inter-Process Comms      |
| <input type="checkbox"/> Grid Computing                 | <input type="checkbox"/> UNIX Systems Performance      |
| <input type="checkbox"/> High Throughput Screening      | <input type="checkbox"/> UNIX Workstations             |
| <input type="checkbox"/> Instrumentation                | <input type="checkbox"/> Visual Basic 6                |
| <input type="checkbox"/> Integrated Lab Systems         | <input type="checkbox"/> WAP                           |
| <input type="checkbox"/> J2EE                           | <input type="checkbox"/> Web Services                  |
| <input type="checkbox"/> Java                           | <input type="checkbox"/> Windows 2000 Services         |
| <input type="checkbox"/> Lims                           | <input type="checkbox"/> XML                           |
| <input type="checkbox"/> Linux                          | <input type="checkbox"/> X Windows                     |
| <input type="checkbox"/> Microsoft Net                  |  |



INVESTOR IN PEOPLE

**Tessella Support Services plc**

3 Vineyard Chambers, Abingdon, Oxon, OX14 3PX, England

Tel: (+44) (0) 1235 555511 Fax: (+44) (0) 1235 553301

E-mail: [info@tessella.com](mailto:info@tessella.com) Web Address: <http://www.tessella.com>