

## Understanding Web Services

By Steven Smith

This article will explain the complexities of Web Services, using Geometry's implementation of a Mapping Web Service into Forestry Tasmania as an example.

### Introduction

Web Services were designed as a level of abstraction between potentially different computer systems. It allows a programming interface to be defined without being tied to any particular programming language, operating system or computer architecture. This provides freedom for client and server-side developers to choose the most appropriate tool for their particular job. There is no need for either to know anything about what the other is using.

For example, server-side applications are generally better written in a cross-platform language such as Java to maximise portability for the end-user. They can deploy the server on a Windows system if required, or any other Java enabled platform such as the various Unix flavours.

However, client-side applications can be a different story. Most client workstations run one of the Windows operating systems, so it sometimes makes sense to use a Windows tool to create client-side applications. The .Net framework provides excellent tools for this task, as-well as server-side components. Client-side applications can still be developed in many other languages, such as Java, PHP, Perl or C/C++.

### Application Programming Tools

There are many tools available to create Web Services, both server-side and client-side. Geometry chose the Apache Axis project as the basis for the Web Service implementation into Forestry Tasmania as Apache is a strong player in the open-source community and has provided many excellent programming tools over the last few years. Geometry uses many Apache tools in our development and has come to appreciate the well designed and implemented projects the Apache foundation delivers.

Apache is currently implementing two versions of the Axis system: one in Java and the other in C/C++. We used the Java version for the server-side application and client-side demonstration application. In the long term, you can use whatever tools you desire to create your clients for the Web Service.

### Application Programming Interface

Every Web Service is defined by a WSDL (Web Service Descriptor Language) file. WSDL is an XML file that describes what services a Web Service provides, how requests are to be made to it and where it can be found to make those requests. Even though the WSDL is written in XML, it is

not really meant for humans to read. The long term goal of Web Services is for applications to read the WSDL and then dynamically generate client applications to utilise the services found within it.

However, this is a complex task that will not come into fruition for many years. The easier way to develop client applications for a Web Service is for a human to find and understand the WSDL for a Web Service and then program to it directly.

Being the only piece of information in between the server and client, the WSDL file needs to be well designed. It must provide all of the functionality the client desires. It must present that functionality in a clear and concise manner. The WSDL file actually defines requests and responses (and exceptions for when errors happen) as XML. An element structure is defined for each request/response/exception so that both the client and server understand each other clearly.

Even though the inter-computer communication is performed through XML for maximum interoperability, the client application will most likely not deal with XML for these purposes. Most Web Service programming tools provide a means to generate objects within your particular programming language for the developer to use and communicate with the Web Service. This provides the developer with a comfortable place to develop their application. They know and understand the nuances of their own language and do not need to be concerned with the XML conversions usually required when dealing with XML documents. The programming tools take care of the conversion process for you.

For example, suppose we have a Web Service written in Java and a client for that service is written in .Net. Once the client has found the service, it creates the required objects written in, say, VB and makes a call to the Web Service to perform some operation. The .Net Web Services framework takes those .Net objects (VB, C/C++, C#, etc) and converts them into XML to send to the Web Service. On the server side, the server decodes the XML into Java objects for the Web Service to understand and use to fulfil the request. So even though our initial objects were in VB, the client framework converted them into XML before sending them to the server. Once on the server, it converted them into Java objects that represent the same information.

At first, the OpenGIS GML Schema was to be used for spatial elements at Forestry Tasmania. Unfortunately, this could not be used as it contains abstract elements and most Web Service Frameworks find it hard to deal with this abstract nature. More importantly, it becomes very hard to implement cross-language communication between the server and client with abstract elements.

## The Communication Process

In order to publish a Web Service, a number of modules must come together for a client to establish communication with that service. The first step for a client is to find the Web Service. This is usually done by querying a UDDI (Universal Description, Discovery and Integration) server. A list of available Web Services and their descriptions is presented for the client to choose from. The client then asks the UDDI server for the WSDL file for any Web Services it is interested in.

At this point, the client may actually be a developer looking for Web Services for a client application. As mentioned above, it is easier to develop a client that knows what Web Services it requires and how to find and use them.

From the WSDL file, the client has all of the information it requires to find and communicate with the Web Service. However, at this point, the client has not sent or received anything from the Web Service itself. The UDDI server may be a public server hosted in a different location to the Web Services that it publishes. It is even possible to write the client so it implicitly understands the Web Services to avoid the need for a UDDI server all together. If a UDDI server is required, the Apache Axis system includes a UDDI server as part of its infrastructure.

---

If the client intended to dynamically consume this Web Service, it would read the WSDL file and create some language specific objects to encapsulate the required information for the service. Otherwise, the client developer will create these objects for the client to use.

The client then fills the objects with information for the Web Service. Calls to the Web Service framework are made to invoke the Web Service and pass all of this information along. When the Web Service returns, it may need to pass information back to the client as a response to the given request. This response will also be translated into objects the client can understand and it will proceed based on the returned data.

From the perspective of the client, dealing with a Web Service is quite trivial and can actually appear as a local object. It takes surprisingly little work by the developer to call a Web Service. All of the hard work is taken care of by the Web Service Framework being used, whether this is .Net, Apache Axis or a host of other frameworks that are currently available.

## **Server-Side Components**

The target platform for the server-side components is Java using the Apache Axis Web Service Framework. Axis requires Java 1.4.x or above and is based upon the Java Servlet architecture. This means Axis requires a Java Servlet Container implementation such as Apache Tomcat 4.x or above or Oracle Application Server in order to run. The development team used the Apache Tomcat Servlet Container in Forestry Tasmania, chosen because it has proven to be very reliable and is the Reference Implementation for Servlet Containers.

Given this environment, the Forestry Tasmania Mapping Web Service is deployed to the Servlet Container. Behind the Web Service, implementations of the OpenGIS WMS and WFS specifications are required to actually perform the operations published by the Web Service.

## **Client-Side Components**

On the client side of things, you can use whatever Web Service framework you desire. As a demonstration of a small client, Geometry used the Apache Axis framework. For the client, Axis only requires Java 1.4.x or above.

Geometry created client-side objects for the client application to use for communication with the Web Service. These objects were created using the Apache Axis framework.

## Forestry Tasmania Architecture Diagram

This diagram shows the different pieces of the Web Service architecture and how they fit together. Processes shaded in red were developed and delivered as part of the Forestry Tasmania Mapping Web Service project.

