Web Services Overview

Web Services

Web Services are a very simple idea based on a situation and chain of events that many of us use every day without thinking. To look at the basics of Web Services, we need to look at the basic way in which a Web request works.

When a client requests a page from the server, it opens the network connection and sends a request (the URL of the page or object the client wants to download), and the server responds with the data.

When accessing a dynamic page—that is, one that's made up of either embedded request components or the result of a form—the client sends the request for the script or application that will process the request, along with the data contained in the form that will be processed by the application. The result, again, is the data the client requested.

In both cases, the information returned is usually in the form of HTML, but it also includes graphics, video, applications, Zip files, and a whole host of other formats used to make up the content of a Web site.

Web services play on this basic premise—the supply of data and the neutrality of the response (that is, it doesn't have to be HTML) to send requests directly to a remote application.

Instead of requesting a specific page or supplying a script with a form full of data, a Web Service client sends the data and the name of the function that you want to be performed embedded within an XML document. The XML is still sent to an application, but instead of the application

responding with HTML, an image, or other information, it responds back with an XML document.

This exchange mirrors the typical exchange of information that happens when a function or procedure is executed within an application. The difference is that it's translated into XML. XML is architecture neutral—we can use it to exchange information between machines without having to worry about the technicalities of file formats, line termination, or the architecture of the CPU.

The benefit of Web Services is that we can achieve the same basic ability of running a function, but we can do it on a remote server, using accepted open standards in an open and standard way. This means we can build applications that use this functionality in a way that doesn't rely on building custom solutions (and therefore locks us into specific architectures) and also doesn't rely on the client running a Web browser or the server always responding with HTML.

Because we're using XML, HTTP, and other open standards, the architecture of the client and server don't matter. It could be a Mac OS X client querying a Windows Server 2003 machine or a Windows client querying a Unix server. Most significantly, it doesn't have to be a Web browser; it can be any application that has been written with Web services capabilities.

There are two different Web Services solutions—both developed independently of each other but that provide a number of similarities. Both are compatible with a wide range of platforms and tools, and both can be easily integrated into an existing Web server environment.

Choosing which solution to use is not an exact science, and it's as much about the personal choice of the programmer as anything else.

XML-RPC

XML-RPC, short for *XML Remote Procedure Call*, is a relatively simple protocol that enables a client to execute a remote procedure, sending parameters and receiving a response.

XML-RPC is a relatively straightforward protocol ideal for the sort of database querying, updating, and other simple integration tasks.

Depending on the programming solution you are using, the XML-RPC system can be as transparent as using a local version of the function.

SOAP

The *Simple Object Access Protocol (SOAP)* operates in a similar fashion to XML-RPC, but has the advantage that it was specially designed with the idea in mind of manipulating objects remotely. This means that instead of calling a simple function or procedure on a remote

machine, we can use SOAP to create a remote object and call methods directly on that remote object.

Since a lot of software is now based on the object-oriented model, SOAP fits better into this type of system. It also enables us to work with remote objects in an identical way to a local object while allowing us to use the horsepower, connectivity, and functionality of the remote server.

Security

One of the problems with typical client/server solutions is that the security of the systems can be complex to organize. If working with an internal system but allowing external access, say with an extranet, you have to configure additional ports on the firewall and have additional potential worries about secure communication and authentication of users to the system.

With Web Services, you can use standard Web ports, which are probably already open, in combination with standard HTTP-based authentication, security, and, if necessary, encryption with the standard SSL system.

This reduces the time required to develop the software while ensuring the maximum security benefits—all with very few changes to your existing infrastructure.

Future Directions

In reality, we've come full circle again. In the early days, the future of computing was seen as the massive mainframe in the corner with individual clients. Then we moved to client/server technology and put powerful machines on the desktop as well as in the server room. Then we went the 'thin client' way, using Web browsers and hefty Web servers to provide custom solutions.

With Web Services, we can return to a client/server model, but remove all the restrictions and limitations of the previous solutions that made it so difficult to make solutions that worked across a variety of platforms.