**Remote Data Exchange**

No matter where you look today, you will find evidence of computers and devices exchanging information. Most of the time you never think about the process, and other times it’s at the forefront of your mind. For example, requesting a document from a website is obviously an example of exchanging information with a remote machine: The client sends a request and receives the document that was requested.

Now think about logging on to computer within a network: When you type in your name and password, the information has to be checked with the network server. That exchange of data is relatively straightforward, but it still requires network communication, and in turn it requires a protocol, a method of data representation, and of course the exchange of information.

Hopefully, you’ve spotted a few key words there. If these systems are exchanging information and data, then they are ideal targets for the use of XML.

In practice, the web browsing example is not an ideal target for transition to XML. Currently, the act of submitting a request and getting a response is largely one-sided; a typical request will be about 100 bytes, but the response could be 2KB, 16KB, or 16MB. However, there’s an argument that says this division of client and server will change as websites and services become more interactive.

The second example, that of logging in to a network, is a more likely target. You are sending very small, discrete pieces of information and getting similarly tiny responses. You’re also running a remote procedure—the one that checks the user database and makes sure that what you supplied matches what’s in the database. All it needs to do is return a true or false result and you’ve achieved your aim.

Remote procedure calls are nothing new—the Sun platform has had a system called *Remote Procedure Call* (RPC) for many years, and many of the services, from data monitoring to printer spooling, have been supported using the RPC system.

The problem with RPC is that although the system was technically cross platform, there was a substantial amount of work involved in translating the data types used in your code into an architecture-neutral format for transmission over the network. The external data representation (XDR) system was very complex, and for anything but the simplest data types you had to perform the process manually.

Using XML you get round all of this—you can convert the request, which is made up of the function or procedure you want to call and any arguments that you supply, into an XML structure. You transfer that XML document over the network to a request handler, which decodes the contents and converts it into a local function or procedure call. The whole process then works in reverse, with the return value being converted into an XML document that is then sent back.

Two such systems that provide this functionality have been produced: Simple Object Access Protocol (SOAP) and XML-RPC. Both work in a similar manner, although they are incompatible with each other.

In both cases you create a server, which can be a CGI script hosted on a web server, a dedicated network service provider (in much the same way as a web or FTP server), or in some cases an e-mail processor that reads the XML request as an attachment to an e-mail. The client then talks to the server, either through a normal HTTP request (in the case of the CGI or network service) or by bundling the request in another packet such as an e-mail, and then waits for a response.

Both SOAP and XML-RPC are easy to use, and it’s likely that you will see an explosion of Internet services being supported using these systems in the future. You can already find public services for converting quantities and temperatures and doing basic calculation.

Because you’re calling remote procedures, the complexity of the request and the data you transfer is not limited by the constraints of HTTP, and you don’t have to worry about creating our own protocol to handle the communication side. In most instances, using the two systems is as simple as specifying the name of the function you want to call and the arguments that you want to supply. Under Perl and Python, these calls can even be as transparent as calling a local function.

The other major benefit of both solutions is that they are both platform *and* language independent. You can call a SOAP object from a Perl client when the object itself is hosted on a Python server. This interoperability means that you no longer have to worry about which language you use to provide each end of the solution.

The capability to provide mixed-language applications in this way has helped to drive Microsoft and its .NET initiative, which is in itself an attempt to blur the distinction between developing an application in one language and doing so in many.

**SOAP**

Simple Object Access Protocol (SOAP) was developed by a consortium of companies that included Userland, IBM, Lotus, and Microsoft. As the name suggests, SOAP was actually designed as a method for accessing and working with objects remotely.

Although you can use SOAP for the simple execution of a remote procedure, its real power is in its capability to manipulate objects, either created on the server side or created and then returned to the client. For example, you can have a server process that provides access to a customer’s account through an object interface. The object and server can be written in Python, but you can create and manipulate the object from Perl or Java, or indeed Python.

SOAP’s power, and its major advantage over XML-RPC, is that you can work with objects over the network. You are not dealing with a simple request and response; once the object has been created, it remains until you delete it. You can therefore use SOAP when state information is useful, such as logging in to a server or making purchases from a catalog.

Because SOAP deals with live objects, it’s frequently seen as the killer application where other attempts have failed. Those with long memories will remember systems such as Common Object Request Broker Architecture (CORBA) and Microsoft’s Distributed Common Object Model (DCOM). It’s unlikely that we’ll see these systems disappear anytime soon, but don’t expect them to last forever, either.

**XML-RPC**

Curiously, XML-RPC actually grew out of the some of the initial work to develop the SOAP standard. However, unlike SOAP, XML-RPC was designed entirely from the procedural point of view, and rather than dealing with objects, it deals with simple requests and responses.

In fact, XML-RPC is best described simply as a method for supporting Remote Procedure Calls. Unfortunately, this makes it useful only in situations where you would normally run a function. You cannot use it for working with objects, and you can’t use it for applications that require state information.

**Limits**

Although I’ve portrayed both SOAP and XML-RPC as solutions to the problem of data exchange between computers and languages, this shouldn’t lead you to believe that these solutions will replace everything that requires network communication.

It’s unlikely that protocols such as HTTP and FTP will be replaced. Neither SOAP nor XML-RPC is a great alternative for transferring large files, and XML itself is not ideal for storing binary data at all.

Instead, SOAP and XML-RPC will replace the sort of solutions that up until now have required either clever use of HTTP or FTP or a whole new protocol of their own.