Service Oriented Architectures

Much media attention is currently being devoted to Service Oriented Architectures (SOAs). Bob Brown examines the major technologies that underlie what has been called 'the grown-up Internet'

Bob Brown, director of Transentia Pty Ltd, has about 15 years' experience as a software researcher and developer and in tertiary-level lecturing throughout the world. Brown is a regular presenter at conferences and events and is co-author of the Prentice-Hall book JAVA Thin-Client Programming for a Network Computing Environment**. Brisbane-based Transentia Pty Ltd provides specialist consulting, development and training services in such technologies as J2EE, Java, Linux, CORBA and XML. Visit:** www.transentia.com.au **E-mail:** bob@transentia.com.au

An SOA is a three-legged stool built on three rapidly developing technologies: the Simple Object Access Protocol (SOAP), WebServices and Universal Discovery, Description and Integration (UDDI). Each of these technologies, in turn, relies on XML, the eXtensible Markup Language, which is being promoted by the World Wide Web Consortium (W3C[1]) as a standard mechanism for facilitating data interchange.

The promise is great: once they mature, these technologies should make it possible to build self-describing services that can be accessed in a protocol and language-independent way, and that can be advertised in a searchable and browseable directory.

All the major vendors, including Microsoft, IBM, Oracle, Sun and BEA, are (at the moment, at least) lining up behind this group of technologies, and so it is possible to hope for a future free of the vendor-driven COM/ CORBA/Java 'object wars' that have crippled all previous attempts at building large-scale, open interoperable systems.

Let us take a look at each of the legs of the stool in turn.

SOAP

At first sight, SOAP is an unimpressive and, some might say, obvious idea. Where in the past we built systems that interacted using carefully designed and specified binary-formatted data, SOAP interchanges simple textual messages formatted using a relatively straightforward XML-based schema. Where previously we carried our data over optimised, carefully controlled protocols, SOAP is not similarly constrained: SOAP has been designed to make it possible to exchange SOAP-encoded data using standard HTTP. If security is important, SOAP can use the standard secure HTTP protocol to protect its data exchanges. SOAP can even exchange data using SMTP, if asynchronous interchange is useful.

This ability to utilise common, well-known transport mechanisms is one of SOAP's strengths and means that we do not need to worry about infrastructural issues like firewalling and routing new protocols.

Because the SOAP interchange format is simply XML-formatted text, it can be manipulated in a straightforward way by almost any programming language. Implementations exist for many languages, including Java, C/C++, C#, Ada, COBOL and PERL. In stark contrast to earlier technologies like EDI, SOAP messages are text and can be debugged very easily.

Since SOAP is still a young technology, it has its share of teething problems, primarily a perceived lack of security, performance problems and interoperability between implementations.

The most common answer to questions about SOAP's security shortcomings is: "It's not its job." Recall that SOAP delegates the transport of messages to one of the existing standard Internet protocols. SOAP messages are perfectly at home being carried across links secured by SSL/TLS or other secure infrastructure. Much work is being done to permit the use of Digital Signatures with SOAP. In these ways, the designers of SOAP have clearly followed their number-one tenet: "First invent no new technology." Performance is one area where much remains to be done. Experiments done at Indiana University[2] show that, since SOAP messages rely on XML, they tend to suffer from low data density. An eight-byte double may require up to 80 bytes when carried as payload in a SOAP message. Moreover, IBM has estimated that an average SOAP message will run to about 60K. Clearly, the 'grown-up Internet' is going to require high-bandwidth links! Memory and processing power will be needed for SOAP as well. Indiana University's experiments have shown that encoding and decoding SOAP data can take up to 10 times the amount of memory and be up to 100 times slower than for an equivalent Java Remote Method Invocation application.

Interoperability is an issue that also needs to be examined. There are currently about 70 different implementations of SOAP from as many organisations. There are bound to be many problems cropping up as the various developers get their implementations 'bedded down' and ready for serious work.

Developers are starting to organise 'Interoperathons' to test their implementations against their competitors software and to show the world that SOAP is ready for the big time.

WebServices

SOAP makes it possible for a service to interact with another service, regardless of the underlying network transport, implementation language or operating system. For a fully open, dynamic, grown-up Web, this is not enough. In a global Internet where trading partners need to come together in an ad hoc manner and where systems are assembled from numerous third-party services, applications should be able to self-configure according to the needs of the service with which they are interacting. As the second leg of our stool, WebServices allows for this.

With WebServices, a service that wishes to advertise its existence does so using a 'résumé' written in the XML-based Web Services Description Language (WSDL). A WSDL definition contains various stanzas defining the service, specifying incoming and outgoing message formats, message sequences and which transport protocols are supported by the service.

Since WSDL fully defines a service, it becomes possible to generate automatically the code required to drive the service as and when the need arises. The WebServices toolkits from vendors such as Microsoft and the Apache group already provide such functionality, making it relatively easy to build client applications.

Like SOAP, WebServices are still immature. The main outstanding issues surrounding WebServices include manageability and high-level authentication/authorisation. In addition, when a system is comprised of many WebServices whose location and qualities are potentially dynamic, the issues of testing, debugging, profiling, etc. take on a whole new dimension.

UDDI

The remaining leg of our stool is the most controversial and so far the most under-defined. Universal Discovery, Description and Integration is a common set of SOAP APIs that enable the implementation of a universally accessible service broker. Although not yet fully specified, UDDI aims to provide repositories for WebServices along with tools and APIs, making it possible to publish WebServices, and allowing for various types of searching (white-pages, yellow-pages and green-pages [looking up dynamic configuration information] styles are all defined).

What makes UDDI so potentially controversial is that the organisation that controls the repository also controls the associated marketplace by virtue of being able to control who gets included into a repository and by being able to influence and prioritise the outcome of searches. Similar behaviour is seen in some Web search engines today, but since UDDI is a technology targeted much more towards large-scale business, the stakes are higher. This is, of course, a potentially powerful and highly profitable position. Not surprisingly, the larger organisations such as Microsoft and IBM are all keeping a close watch on this technology!

Critics of UDDI also charge that the technology is not needed and maintain that enterprises do not need to be able to discover trading partners dynamically but instead need to be able to work more efficiently with the partners they already have. Ken Vollmer in InternetWeek[3] has summed up this position nicely: "UDDI's problem: Technology Cannot Replace Relationships."

Current activity

Many vendors are rallying behind the WebServices banner. Microsoft is basing a large amount of its forthcoming .NET infrastructure on this technology. Microsoft has provided impressive support for WebServices in its Visual Studio.NET development tool. IBM (an enthusiastic supporter of WebServices through the Apache group) has an equally useful, though not as polished, set of Java-based toolkits. Oracle is supporting WebServices under the banner of 'Dynamic Services'. Sun Microsystems is late but is catching up quickly: its Java APIs for XML Registries (JAXR) API specification provides a convenient API that developers can use to access registries, including UDDI registries. The W3C is coordinating a lot of standards-oriented activity under the auspices of the XML Protocol Working Group.

Under the .NET banner, Microsoft is planning to introduce a set of Hotmail-like services that it is calling 'Hailstorm'. These services will enable a number of user-centric applications such as myAddress, myContacts, myInbox and myCalendar.

In the Java world, most J2EE application server vendors are providing support for WebServices, permitting Servlets and EJBs to be accessed via SOAP in addition to the more traditional HTTP/RMI protocols.

Conclusion

Although some commentators have dismissed SOAs as being merely the latest fad, they are undoubtedly a technology that is here to stay. The major unanswered question is whether they actually correspond to the real needs of the industry at large or whether they merely represent a vendor-driven mechanism aimed at maintaining control in an increasingly open Internet. ■

REFERENCES:

[1] www.w3c.org

[2] www.extreme.indiana.edu/soap/sc00/paper/paper

[3] www.internetweek.com/columns01/beat062001.htm