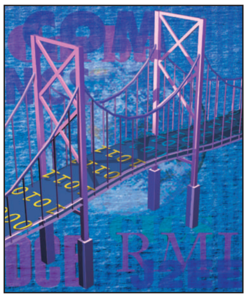

Web Service References

Steve Vinoski • *IONA Technologies*

I wrote about notification-based Web service systems one year ago ("Web Services Notification," Mar./Apr. 2004, pp. 86–90). To properly address the topic, I devoted a significant portion to Web service references, which are, conceptually, like network handles that refer to Web services. Service references are particularly important for notification systems, which usually require subscribers to register service references to receive notification messages. Given that they appear in one form or another in almost every distributed system, service references are nothing new. Nevertheless, no standardized service reference exists for Web services.

Among other things, the WS-Addressing specification,[1] which entered the W3C (www.w3.org) standardization process in October 2004, specifies a construct called an *endpoint reference* (EPR), which is much like a service reference. I'm a member of the WS-Addressing working group (WG), and we've made good progress toward turning the specification into a standard — the process might even be complete by late 2005. In this column, I discuss a few issues that I raised in the WG regarding the features needed to turn the EPR into a flexible and useful Web service reference.

The Nature of Services

The excessiveness of the media hype surrounding the service-oriented architecture (SOA) during the past year or two notwithstanding, several companies have used it to develop effective enterprise computing systems that provide measurable returns on investment. SOA's recently increased popularity is due to Web services, but developers have successfully deployed it for many years using older technologies such as the Distributed Computing Environment (DCE), messaging systems, Corba, and Java 2 Enterprise Edition.

In enterprises that have the knowledge and capabilities to successfully apply service-oriented approaches, it's rare to find only homogeneous SOAP-based Web services. Instead, SOA deployments typically comprise a variety of implementation platforms and technologies. Because successful business computing systems tend to remain tied to the technologies on which they were launched (rather than changing every time a new technology comes along), implementation technology mixtures are inevitable.

An enterprise's inherent heterogeneity means that services often must span multiple technologies. When services don't do so seamlessly, the result is a collection of technology stovepipes, which are often joined through combinations of cumbersome gateways or slow, expensive, and proprietary enterprise application integration (EAI) systems.

Multitechnology services — those accessed via multiple protocols, transports, or middleware systems — avoid technology stovepipes. Thus, references for such services must convey all means of access that a service wishes to make known to its consumers. Let's consider typical Internet services. As Rich Salz of Datapower Technology (www.datapower.com) points out, Internet servers are often implemented as multiport applications in which each port handles a different protocol.[2] In general, a service might make itself available over multiple ports to offer different qualities of service (QoS) to different consumers. For example, it might accept compressed messages over one port, encrypted messages over another, and, perhaps, management messages over a third. Another service might accept Corba messages on one port and SOAP messages on another.

A real-world analog for a multiport service reference is your business card. Along with your name and job title, your business card lists various ways to contact you: phone, fax, and cell phone numbers, email address, mailing address, homepage, and, per-

continued on p. 94

continued from p. 96

haps, your instant messenger ID. Someone who wants to reach you can choose any of the methods listed on your card, and despite the different operational details and QoS each contact method offers, all of them lead to the same "service": you.

EPR Shortcomings

As currently defined, the WS-Addressing EPR allows access to a service over only a single port. That's like mandating that a business card specify only one means of reaching a person, thus forcing you to have multiple business cards to allow others to reach you via multiple access means. Imagine attending a business meeting where you had to give business cards to five people, each of whom you wanted to be able to reach you via your desk phone, cell phone, or email. You'd have to hand out 15 cards to do so.

There are at least four approaches to dealing with this problem for services.

1. Augment the EPR structure with additional port information, perhaps by adding optional fields to the EPR.
2. Standardize a policy assertion that could convey additional optional port information within an EPR. Sanjiva Weerawarana, research staff member at IBM T.J. Watson Research Center, suggested this approach, based on WS-Policy,[3] in an email to the WS-Addressing WG in November 2004. This approach could work in theory, but because WS-Policy isn't an actual standard, the WS-Addressing standard can't reference it.
3. Create a separate structure that can hold one or more EPRs. Someone could argue that the concept of a "multiport EPR" is flawed because endpoint and port are somewhat synonymous. Creating a separate multi-EPR structure would leave the existing EPR as it is and create a separate composite reference structure capable of holding one or more EPRs. A composite reference would be similar to a Web Services Description Language (WSDL)[4] service element and could be passed around in messages just like an EPR.
4. Use WS-MetadataExchange (WS-MEX).[5] Here, a consumer would use a service's advertised EPR — containing information for only a single port — to retrieve the service's metadata, which would provide any additional port information.

Another shortcoming with the single-address EPR relates to service availability.

Ultimately, it boils down to whether a service can carry multiple port information in its EPR by value, or whether it must advertise an address that an application can use to indirectly obtain information about alternative ports. WS-MEX promotes the indirect approach, whereas the by-value approach is reminiscent of Corba interoperable object references (IORs). The IOR approach, which has been in use for about a decade, proves the viability and usefulness of having a service reference structure that can carry connectivity details for any number of protocols and transports.

The WS-MEX approach would work, but it would require network operations (which can be fairly expensive) to retrieve additional port information, and it would be tough on legacy services, which can't be rebuilt and redeployed to support WS-MEX. For each legacy service, we could introduce a separate WS-MEX service process, but that would add production system overhead in the form of extra hardware provisioning, system management, and maintenance.

Another shortcoming with the single-address EPR relates to service availability. In the current WS-Addressing specification, EPRs can hold only a single service address. In fact, the name EPR implies that it is just that — a reference to a single endpoint. One question this design evokes is fairly obvious: how does an application access the service when the address indicated by its EPR fails?

If an application using a failed EPR originally retrieved it from some lookup or directory service, the application could presumably go back, do another lookup, and get a new or different EPR. This assumes that the EPR-referenced service is available at multiple endpoints, the service has advertised multiple EPRs in the directory service, the directory service is capable of storing multiple EPRs under a single service name or description, and that the directory service hands out EPRs in a round-robin, random, or other process that prevents calling applications from getting the same EPR each time they ask. Those are a lot of assumptions.

What if the application received the failing EPR from some other application, perhaps as part of some message? Unless that message provided multiple EPRs to try for each service, the application would be stuck with a failed EPR, and would be unable to reach the service.

We might argue that the machinery for high availability is somewhere in the Web services stack below the level of the address contained within an EPR, such that EPRs need not account for such failures. For example, we might use the same approaches that distribute loads across multiple Web servers to try to ensure that at least one service instance is alive and ready at the endpoint indicated by the EPR. Indeed, this is one viable approach, but it shouldn't be the only implementation choice the standard imposes.

A composite or multiport EPR could refer to a single logical service and hold physical addresses for multiple endpoints. Thus, an application encountering a failure using one physical address could go back to the same EPR and try a different address. This approach would alleviate the need for other machinery above or below the EPR level to provide for service availability, while obviating the current EPR design shortcomings that force you to use some other approach to allow for such availability.

Purity vs. Practicality

Because WS-Addressing is destined to become a W3C standard, some believe that it should support only standard W3C protocols, specifically SOAP. Of course, WSDL is also a W3C standard. Unfortunately, not everyone wants to focus at that level in the Web services stack. Because of the two camps' different focuses regarding Web services, there is a divide between the SOAP and WSDL supporters.

Web services aren't as much about SOAP as they are about messages and contracts. SOAP is a good way to encode Web service messages, but it's not the last good way we'll ever see. WSDL is a very flexible way of describing and abstracting services that span multiple message protocols, formats, and transports because it's a step removed from the specifics of those technologies. WSDL supports SOAP, but doesn't require it. It can support a wide variety of protocols, transports, and message formats because, unlike the current version of the EPR, WSDL's designers explicitly included extensibility to support multiprotocol services.[6] This enables WSDL to support older protocols and message formats that are still important in today's enterprises. As underlying technologies like SOAP inevitably come and go, WSDL's extensibility can accommodate them, and the end result is a strong and clean separation between business application logic and the technologies that implement it.

An older successful service abstraction language example is the Corba Interface Definition Language, which the Object Management Group (OMG) originally defined in the early 1990s. IDL's definition came several years before that of the Internet Inter-ORB Protocol, which Corba object request brokers (ORBs) now generally use to communicate. Before IIOP, IDL worked with a variety of vendor-specific proprietary protocols and transfer syntaxes. Interestingly, IDL didn't change at all when IIOP came along; it worked for the then-new IIOP-accessible systems, and yet continued to work for all the older vendor-specific protocols. This let vendors add support for the IIOP interoperability standard to their products while maintaining support for their own protocols, all without requiring their customers to make any code changes. If not for this, Corba would likely have died in the mid-1990s. Yet, telecommunications, manufacturing, finance, and other industries are still heavily using it a decade later.

One counter to this argument is that WSDL is not IDL. Note that in my earlier statements, I wasn't talking about objects versus services, statefulness versus statelessness, or any of the other positions that typically accompany the WSDL versus IDL debate. Instead, as with the business-card analogy, what matters is the service and its contract, and both WSDL and IDL support that view. There might be multiple ways to access a given service, and how you access the service could change over time. What ultimately provides business value is the service, not the pipes leading to it.

In many ways, it comes down to which of the two words in the phrase "Web services" you focus on. SOAP advocates focus on the Web part, which supports their general opinion that SOAP and HTTP are all that matter. We who focus on WSDL tend to address the services part, because most Web services work today occurs not on the Web but in the enterprise, where a variety of protocols and message formats are facts of life, and will continue to be for quite some time. I know of no compelling reason that the WS-Addressing EPR, appropriately augmented to allow for multiport services, couldn't suffice as a standard Web service reference and serve both viewpoints equally well.

Acknowledgments

I thank my IONA colleague Rebecca Bergersen for her feedback.

References

1. D. Box et al., *Web Services Addressing (WS-Addressing)*, W3C member submission, Aug. 2004; www.w3.org/submission/2004/subm-ws-addressing-20040810.
2. R. Salz, "WSDL 2: Just Say No," *XML.com*, Nov. 2004; www.xml.com/pub/a/2004/11/17/salz.html.
3. S. Bajaj et al., *Web Services Policy Framework (WS-Policy)*, joint specification by IBM, BEA Systems, Microsoft, SAP, Sonic Software, and VeriSign, Sept. 2004; www-106.ibm.com/developerworks/library/specification/ws-polfram/.
4. *Web Services Description Language 2.0*, W3C working draft 3, Aug. 2004; www.w3.org/tr/wsdl20.
5. K. Ballinger et al., *Web Services Metadata Exchange* (WS-MetadataExchange), joint specification by BEA Systems, Computer Associates, IBM, Microsoft, SAP AG, Sun Microsystems, and WebMethods, Sept. 2004; www-106.ibm.com/developerworks/library/specification/ws-mex.
6. S. Vinoski, "Integration with Web Services," *IEEE Internet Computing*, vol. 7, no. 6, 2003, pp. 75–77.

Steve Vinoski is chief engineer of product innovation for IONA Technologies. He's been involved in middleware for 17 years. Vinoski is coauthor of *Advanced Corba Programming with C++* (Addison Wesley Longman, 1999), and he has helped develop middleware standards for the Object Management Group (OMG) and World Wide Web Consortium (W3C). Contact him at vinoski@ieee.org.