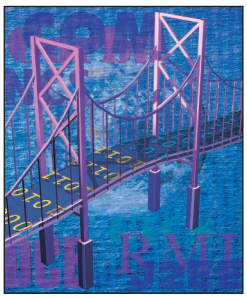

Web Services Notifications

Steve Vinoski • *IONA Technologies*

It's an asynchronous push-style world out there. We communicate via email, instant messages, pagers, faxes, and voice mail, pushing messages to each other without waiting for immediate responses. This style of communication lets us prioritize, save, or ignore incoming messages, delaying responses until necessary or appropriate. If sequentially issuing and handling synchronous requests and responses were our only mode of interpersonal and inter-application communication, our business and social networks would operate at considerably lower efficiencies.

Asynchronous push is also common in distributed computing systems. For example, publish–subscribe (pub–sub) systems let applications communicate through pushed messages. Publishing applications push messages into the system, and from there they're pushed out to subscribed applications. Pub–sub systems support a variety of subscription approaches, such as "send me all messages sent to this communication channel" or "send me only those messages containing field X with value Y." For example, a system-monitoring application might post all error, warning, and informational messages it receives from a set of distributed applications. Other applications interested in seeing the messages, such as those running on system-operator desktops, subscribe to those messages — usually with the ability to filter them according to type and content. (Most pub–sub systems also provide for polling, or pull-style, subscriptions, but designers usually eschew this scheme in favor of the push approach because polling is much less efficient.) Pub–sub systems also support different qualities of service, typically revolving around the lifetimes of messages and whether they are stored persistently for later retrieval by intermittently connected consumers.

As Web services have matured, they've also acquired the need for asynchronous push capabilities. Early Web services, such as those for getting traffic or weather reports or for performing currency exchange-rate calculations, were seemingly all remote procedure calls-oriented. In an RPC system, the receiver typically performs the requested service and sends a response back to the caller over the same connection on which the request arrived, which means the details required by the service for sending a reply are implicit in the RPC network connection. Unfortunately, these kinds of implied communication details are wholly inadequate for asynchronous push Web services.

Service References

First, consider a simple case: the callback. Let's say you have a Web services implementation connected to a stock-market feed, and you have a desktop application that wants to register itself with the service to receive a callback whenever a given stock crosses a certain price threshold. To receive such a callback, the desktop application also must be a Web service, and it must inform the stock service of its callback endpoint. In other words, it must send the stock service some kind of network-communications handle or reference to itself.

However, Web services currently have no standard concept for such service references. You might think that the "Web" part of the Web services name implies that service references are simply URLs. For various reasons, this is not the case.

- A URL specifies only a single protocol, but a service could be reachable via multiple protocols. For example, a service might accept messages over both HTTP and SMTP, but any URL for the service can specify only one of those access methods.
- URLs can't adequately describe some transport mechanism types. For example, message queues typically are defined by a series of para-

 Published by the IEEE Computer Society 1089-7801/04/$20.00 © 2004 IEEE

meters that describe the queue name, queue manager, `get` and `put` options, message-expiration settings, and message characteristics. It isn't practical to describe all of this information in some form of message-queue URL.
- URLs do not necessarily convey interface information. This is especially true of HTTP URLs, because Web services generally tunnel SOAP over HTTP. Given a URL such as http://computer.org/internet/SomeService/, you therefore have no idea whether the resource it represents is a Web page or a Web service. Many Web services development kits automatically return the service's WSDL definition in response to an HTTP `get` on the service URL. Thus, you can indirectly discover the service's interface, but that's only a convention, not a standard, and it's not universally supported.

Several proposals aim to add service references to Web services and thus fix these shortcomings. If adopted, these proposals will eliminate a glaring inequality currently present in Web services — namely, they treat a service reference like a first-class citizen for invoking that service, but it can't appear as a parameter or part of any messages sent for any service invocation. Among these proposals are the WS-Addressing specification,[1] and efforts within the W3C Web Services Description working group to add service references to the Web Services Description Language (WSDL; www.w3.org/TR/wsdl).

WS-Addressing

BEA Systems, IBM, and Microsoft jointly published the proprietary WS-Addressing specification in March 2003. It defines two transport-neutral entities: *endpoint references* and *message-information headers*. Normally, underlying protocols or transports provide the information carried by these types of definitions, but the authors sought to define constructs that messages could carry regardless of whether they were delivered via HTTP, a message queue, or some other means.

Endpoint references comprise the following components:

- a mandatory *endpoint address*, in the form of a URI that might be either a logical or physical address;
- any number of *reference properties*, which help identify the resource at the given endpoint address;
- optionally, the endpoint's WSDL *port type*;
- optionally, the endpoint's WSDL *service name*; and
- any number of policy elements, as described by the WS-Policy specification.[2] Such policies help describe the service's requirements and capabilities at the given endpoint address.

Several proposals aim to add service references to Web services and thus fix these shortcomings.

When an endpoint reference is transmitted as part of a SOAP message, the message header includes the endpoint address and any associated reference properties. We could use bindings other than SOAP, as long as they were described in the WSDL service definition that the endpoint reference refers to or in the endpoint reference policy elements. Regardless of whether other bindings are possible, support for SOAP bindings is mandatory to help ensure interoperability.

These elements make up the message-information header contents:

- a mandatory destination URI, which identifies the intended message recipient;
- an optional recipient endpoint reference;
- an optional message-source endpoint reference;
- an optional endpoint reference, which provides a destination for any replies to this message;
- a mandatory action URI, which indicates this message's intended semantics. Specifically, the action URI lets the message specify the intended action even when sent to a catchall service endpoint. For example, the service URL might always be http://computer.org/internet/SomeService/, regardless of the operation invoked, so the action URI adds the missing information to indicate the actual target operation. The specification recommends that the action URI should somehow correspond to the matching WSDL definition for the intended action;
- an optional message-identification URI, which identifies this message; and
- an optional relationship indicator for how this message corresponds to another. For example, if this is a response to message X, the relationship indicator will say that it's a response and will include the URI that identifies the original message X.

As you probably can discern from these descriptions — together with endpoint references — a message-information header's net effect is to

augment messages with relatively detailed information about sender, receiver, intent, and relationship to other messages. For more details about endpoint references and message-information headers, including examples, please refer directly to the WS-Addressing specification.

Unfortunately, despite their utility, WS-Addressing endpoint references have several drawbacks. First, they rely heavily on URIs for addressing, which are ill suited for describing certain types of endpoints, such as message queues. Second, they assume that message recipients will have a priori knowledge of the WSDL definitions that correspond to an endpoint reference, especially when that reference includes the optional port-type and service-name elements. Finally, the specification is unclear whether it supports using endpoint references within WSDL definitions. For example, does it support the use of an endpoint reference as part of a parameter type for a remote operation? The specification states, "Endpoint references logically extend the WSDL description model (for example, port types, bindings, and so on), but do not replace it." Other than this statement, WSDL gets little mention.

The specification contains no description or examples of how to use endpoint references in WSDL. Instead, it focuses mainly on using them in message headers, implying no support for specifying endpoint references as parts of the WSDL messages that make up a service's WSDL interface. Other specifications that build on WS-Addressing, such as WS-Eventing,[3] indirectly answer this question in that they do use endpoint references in their WSDL definitions. It would be best, however, if the WS-Addressing specification itself were clear on this point.

WSDL 2.0

As part of WSDL 2.0's design (the next major version of WSDL is slated for publication in April 2004), the W3C Web Services Description working group has discussed adding *service references*. As of this writing, proposals are in the works for making service definitions accessible as service references within WSDL type definitions.[4]

WSDL supports the notion of a *service*, which ties together Web services' abstract and concrete elements by combining what a service is with how you get to it. Specifically, a WSDL service unites an abstract *interface* (formerly called a *port type* in WSDL 1.x), which defines the messages that can be exchanged with the service, with *bindings*, which define what mechanisms and data formats are used to communicate with the service. The most commonly used Web services binding is, of course, SOAP over HTTP.

In WSDL 1.x, service definitions were not visible to other portions of the WSDL definition. In particular, service definitions couldn't be referenced within the `types` element or message elements; sending a service reference as part of a message was not possible. In WSDL 2.0, it is likely that service and other top-level WSDL definitions will become visible to each other, thus letting service references appear in types and messages.

Still, WSDL 2.0 service references have a couple of potential problems. First, it's not clear whether WSDL 2.0 service references will support anything like WS-Addressing reference properties, which are generally useful for callback and pub–sub situations. In fact, it's not clear if there are any plans for WS-Addressing to become part of WSDL 2.0. Second, as with WS-Addressing endpoint references, any receiver looking to use a WSDL 2.0 service reference must have a priori knowledge of the service's WSDL definition. This disallows applications that can discover and use services at runtime, such as GUI builders for Web services.[5] Using a service without prior knowledge of its WSDL definition requires the ability to find that definition and to figure out which service and interface port from that definition to use to contact it. Armed with this information, an application can download any unknown service's WSDL definition and use it to properly interact with the service. These three pieces of information enable dynamic applications (those without a priori service knowledge) without penalizing static applications (those with prior knowledge of the service). A static application simply looks up the interface and port in the WSDL definition it already has for the service, gets the service address, and then proceeds with interactions. In fact, at IONA, we have already implemented this approach in our Artix product (see www.iona.com/products/artix/, and see http://schemas.iona.com/references/references.xsd for the definition of Artix service references).

WS-Events

The service-references types just discussed let a Web service pass a reference to itself to another Web service so the latter eventually can call the former. This callback capability lets one service asynchronously push events or notifications to another. Standardized Web services references will provide the building blocks for standardized Web services notification systems.

In July 2003, Hewlett-Packard published version 2.0 of its proprietary WS-Events specification.[6] Unlike WS-Addressing, WS-Events specifically mandates the use of WSDL to describe Web services. It supports notifications between Web services applications in both an asynchronous push style and a synchronous pull style.

To subscribe to an event, a consumer must know the details about the event data it will be receiving. The WS-Events `DiscoveryInterface` provides several operations for discovering what event types a particular producer exposes. The `GetAllEventTypes` operation lets a consumer obtain a list of all event types to which it can subscribe. At the next level of detail, the `GetEventTypeDefinition` operation

takes an event types list and returns a list of XML event type definitions that describe details of each. Finally, the `GetEventInstanceInfo` operation also takes an event types list, but rather than describing them, it returns details about any actual instances of those types that the producer still holds. This operation lets new subscribers synchronously request copies of any existing events.

An event type definition includes the following information:

- a URI representing the event type;
- a URI pointing to the schema defining the event type, allowing consumers to validate incoming events of this type;
- a subscription URL pointing to the WSDL definition of the subscription interface for this event type; and
- a subscription mode indicator that tells how the events are delivered. This indicates whether the event is asynchronously pushed by the producer, synchronously pulled by the consumer, or can operate in either mode.

To subscribe, a consumer passes the following information to the producer's `Subscribe` operation:

- a selector for the event type or types to which it wants to subscribe. This can be one or more event URIs, a regular expression matching the desired event types, or the keyword `all` to match all available events;
- a subscription timeout, indicating a time that the producer should release the subscription, or the keyword `infinite`, indicating that the subscription should never time out;
- an optional filter that prevents the producer from sending undesired event data to the consumer (unfortunately, the specification does not provide any meaningful details regarding filtering); and
- a callback URL, which, if provided, indicates where the producer should push events to. If the URL is not provided, the consumer will pull events from the producer instead.

The result of a successful subscription request is an opaque string that identifies the subscription. Later, the consumer can use this subscription ID to extend or cancel the subscription.

As specifications go, WS-Events has some nice features, but overall it could be improved. In one nice feature, the specification addresses the important issue of subscription cleanup through its automatic subscription-expiration feature. In production systems, one of the biggest problems with notification services is that administrators often take applications serving as event consumers offline or entirely decommission them without properly disconnecting them from event producers, and the stale subscriptions can have negative effects on producer performance and scalability.

On the negative side, the specification completely misses filtering. As with subscription management, filtering is a very important issue for notification systems. When properly applied, filters can help reduce machine and network resource consumption in the overall notification process by keeping the producer from delivering event data that will only be thrown away, and by keeping consumers from processing unwanted events. Another problem is that the specification also allows consumers to synchronously pull events, which implies that producers must have the capacity to buffer events for some time. Such buffering can be difficult to implement and can have adverse effects on producer scalability. Finally, perhaps the worst WS-Events specification element is that it identifies consumer endpoints via URLs rather than service references, which has all the drawbacks I explained earlier.

WS-Eventing

The WS-Eventing specification, a proprietary specification written by BEA Systems, Microsoft, and Tibco Software, also supports the asynchronous push use case. It's similar in content and approach to HP's WS-Events specification, but simpler. Specifically, it lets a Web service (called an *event sink*) register with or subscribe to another (an *event source*) so that the event sink can receive notification messages from the event source. To provide this capability, WS-Eventing builds on the endpoint reference and message-information header capabilities provided by the WS-Addressing specification.

Standardized Web services references will provide the building blocks for standardized Web services notification systems.

To subscribe to an event source, an application sends a subscription message that sets the message header's action URI to a special value to indicate a subscription action that conforms to WS-Eventing semantics:

http://schemas.xmlsoap.org/ws/2004/01/eventing/Subscribe

In the message body, the application sends a URI for the intended event

sink, along with optional reference properties, such as a subscription identifier, to be sent with each notification. The message body also can include an event filter specification — a Boolean expression that, by default, is an XML XPath[7] expression.

In response to the subscription request, the event source returns a message with its action URI set to the following special value:

http://schemas.xmlsoap.org/ws/2004/01/eventing/SubscribeResponse

Its message body contains a subscription identifier and a subscription expiry time.

When the event source has an event to send to a particular sink, it checks the event against any filters that the sink established as part of its subscription. If the filter evaluates to `true`, the source sends the event to the sink.

Eventually, a sink's subscription will expire, indicated by the expiry time included in the original subscription reply. A sink can renew its subscription before it expires by sending the source a renewal message that includes the subscription identifier and a time or duration by which to extend the subscription.

Overall, the WS-Eventing design is simple yet elegant. Unlike the HP specification, it includes a well-designed filtering capability, and avoids the synchronous-pull event-delivery model.

Nevertheless, WS-Eventing could be better in several ways. First, if it were based on WSDL service references rather than WS-Addressing, it would be much cleaner, more elegant, and more easily applicable to bindings other than just SOAP over HTTP. Second, WS-Eventing has no event metadata discovery capabilities, as in HP's WS-Events specification. Third, and perhaps most important, the intellectual property rights and royalty issues associated with WS-Eventing, WS-Addressing, and other associated proprietary specifications are far from clear. Some complain about the relatively slow pace of standards development, but proprietary standards such as these serve only to slow technical advances in the Web services market, not speed them up.

Need to Standardize

Web services could clearly benefit from standardizing infrastructure features such as service references, routing, policies, and notifications. Convergence on these fundamental specifications is crucial. Opening the development and evolution of specifications like the ones I described here to a broader set of participants, while leaving the legal complications out, would help advance Web-services technologies to a level suitable for enterprise-quality systems. Open source has shown benefits for applications, middleware, and operating systems — perhaps we should apply the same principles and create open-source Web services specifications?

Acknowledgments

Many thanks to IONA colleagues Des Carbery and Adi Sakala for their assistance in exploring the service reference design space, and for reviewing a draft of this column.

References

1. A. Bosworth, et al., "Web Services Addressing (WS-Addressing)," joint specification by BEA Systems, IBM, and Microsoft, Mar. 2003; www-106.ibm.com/developerworks/Webservices/library/ws-add/.
2. D. Box, et al., "Web Services Policy Framework (WS-Policy)," joint specification by BEA Systems, IBM, and Microsoft, May 2003; www-106.ibm.com/developerworks/library/ws-polfram/.
3. L.F. Cabrera, et al., "Web Services Eventing (WS-Eventing)," joint specification by Microsoft, BEA Systems, and Tibco, Jan. 2004; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/WS-Eventing.asp.
4. R. Chinnici, "Counterproposal for Service References in WSDL 2.0," submission to the W3C Web Services Description working group, Oct. 2003; http://lists.w3.org/Archives/Public/www-ws-desc/2003Oct/att-0345/counterproposal.html.
5. M. Kassoff, D. Kato, and W. Mohsin, "Creating GUIs for Web Services," *IEEE Internet Computing*, vol. 7, no. 5, 2003, pp. 66–73.
6. N. Catania, et al., "Web Services Events (WS-Events), version 2.0," Hewlett-Packard specification, July 2003; http://devresource.hp.com/drc/specifications/wsmf/WS-Events.pdf.
7. J. Clark and S. DeRose, "XML Path Language (XPath), version 1.0," W3C recommendation, Nov. 1999; www.w3.org/TR/1999/REC-xpath-19991116.html.

Steve Vinoski is chief engineer of product innovation for IONA Technologies. He's been involved in middleware for 15 years. Vinoski is the coauthor of *Advanced Corba Programming with C++* (Addison Wesley Longman, 1999) and has helped develop middleware standards for the OMG and W3C. Contact him at vinoski@ieee.org.