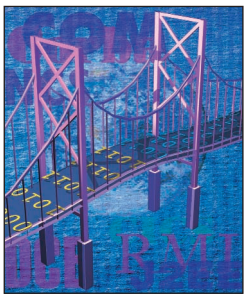

More Web Services Notifications

Steve Vinoski • *IONA Technologies*

In the March/April issue of *IEEE Internet Computing*, I wrote about several Web Services (WS) specifications that define facilities and approaches required to support notification-based and event-driven systems. I had not intended to write a two-part column series, but just after I finished that column, a group of companies, including Hewlett-Packard and IBM, published their WS-Notification specifications (www-106.ibm.com/developerworks/library/specification/ws-notification/). Since then, several readers have asked why I didn't include these new specifications, so I've decided to cover them here.

WS-Notification is actually a family of documents:

- The white paper "Publish–Subscribe Notification for Web Services"[1] serves as a base document for the others. It presents the general notification pattern's basic concepts, WS-Notification's goals and requirements, and defines the terminology used in the related specifications. The paper also contains a detailed and useful discussion of WS-Notification's security aspects.
- The Web Services Base Notification (WS-Base Notification) specification[2] describes the notification-producer and consumer interfaces, along with their associated semantics. It also details the message exchanges required to fulfill these interfaces.
- The Web Services Topics (WS-Topics) specification[3] defines the features required to allow applications to work with topic-oriented notification systems.
- The Web Services Brokered Notification (WS-Brokered Notification) specification[4] identifies the interfaces, message exchanges, and semantics associated with intermediaries known as *notification brokers*. These brokers serve as both consumers and producers of notification messages, and can help increase performance and scalability in notification-based systems.

WS-Notification addresses event-driven systems. Not surprisingly, there are several similarities between WS-Notification and WS-Eventing[4] (which I addressed in last issue's column). For example, both specify interfaces and messages used for event subscription and event delivery. Both rely on WS-Addressing[5] to provide endpoint references for notification consumers, and both supply features supporting automatic subscription timeout. However, there are also significant differences between the two specifications, related mainly to the scope of their coverage.

WS-Base Notifications

The WS-Base Notification document's main contribution is two-fold; defining the `NotificationProducer` interface (supported by applications that accept notification subscriptions and send notification messages) and the `NotificationConsumer` interface (supported by applications that subscribe to notification messages).

WS-Base Notification lets consumers accept "raw" notification messages (which are application-specific) or "notify" messages (which essentially wrap raw messages with notification-specific information). These messages include the topic associated with the message, the dialect used to specify the topic, and an optional endpoint reference to the producer. Notify messages serve as a general notification-delivery approach because they can contain any type of application-specific notification message. A single notify message can also contain multiple notification messages, thus supporting efficient batch-notification delivery.

Producers serve two roles in WS-Notification:

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

producing notifications and handling notification subscriptions. To create a subscription, a subscriber must send specific information to a `NotificationProducer`, including:

- a consumer endpoint reference to which the producer can send notification messages;
- a topic expression, which identifies the topics the consumer is interested in;
- a dialect for that expression;
- an optional indicator for whether the consumer wishes to receive raw notification messages or notify messages (by default, consumers receive notify messages);
- an optional precondition expression, the details of which are specific to each notification producer. The producer does not deliver any message for which evaluation of the precondition expression returns false;
- an optional selector expression, which the producer applies to notification messages that already meet the subscriber's topic expression. The notification producer does not deliver a message if evaluating the selector expression against the message returns false. Selector expressions are thus useful for performing message filtering within the producer; and
- an optional suggested termination time for the subscription. (This time, which is relative to the producer's clock, controls the subscription lifetime.)

The response to a subscription request is an endpoint reference that includes an identifier for the newly created subscription, as well as an address for a `SubscriptionManager` service, which can manage the subscription (when contacted). The `SubscriptionManager` interface allows a consumer to pause and resume subscriptions in order to control message delivery. Depending on the producer's quality of service, a consumer that pauses a subscription might later get all missed messages by resuming the subscription (although the specification doesn't require producers to maintain any such missed messages). The `SubscriptionManager` interface also lets the consumer read and write subscription properties.

Notification Topics

Applications that use notifications typically receive only those notification messages that fulfill certain criteria. As an analogy, consider a public discussion site for home-theater enthusiasts. The site moderator likely wants to see all messages in order to ensure that the site contains only home theater-specific messages that conform to site etiquette and to keep spam from getting through. Site subscribers, on the other hand, almost certainly want to see only those messages that address specific subtopics they're interested in, rather than reading all the messages published to the site. Presumably, such a discussion site could be organized to allow subscribers to sign up for only those messages that cover topics they specify.

WS-Notification supports specific topics that help consumers receive only those notification messages of specific interest. The topic-based approach has been used for years in message-oriented middleware, and is thus well understood. When the producer has a notification message to send, it verifies that the message's topic overlaps with the consumer's subscription. If there is no such overlap, the producer does not deliver that message to that consumer.

The WS-Notification topic approach is flexible and powerful. Topics are arranged within *topic spaces*, which use XML namespaces to avoid topic-definition clashes. Topics are named and might have child topics, thus allowing for the definition of topic hierarchies or trees. The specification supports several *topic expressions*, which are used to specify topics in subscribe and notify messages, as well as to indicate the topics that notification producers support. The WS-Topics document specifies several topic expressions, including a simple approach that refers only to root topics within a given topic space, and a full approach that uses XPath-like expressions to refer to topics.

The WS Notification topic-based approach allows message filtering within the producer, the consumer, or a combination of both, which is important for scalability.

Because of its flexibility, the WS-Notification topic-based approach allows message filtering within the producer, the consumer, or a combination of both, which is important for scalability. Filtering occurs in the producer based on the topics specified for a consumer's subscription, along with any selector expressions and precondition expressions associated with that subscription. Consumers can then apply further criteria to filter messages that arrive from a producer. In event-driven systems that do not support message filtering, every consumer receives every message. If the producer performs the filtering, it eliminates wasted network bandwidth and consumer processing cycles. However, it also adds overhead to the processing that the producer performs for each message. Thus, if implemented poorly, performing all the fil-

tering could leave the producer unable to keep up with the desired message-delivery rate. WS-Notification topics, together with precondition and selector expressions, are flexible enough to avoid locking applications into undesirable low-performance filtering approaches.

Notification Broker

I have implied that notification producers always send notification messages directly to notification consumers. Such a direct connection approach is useful in closed systems in which there are relatively few producer and consumer applications — and those applications already know each other and are unaffected by the coupling introduced by direct connections. Most event-based systems, however, seek to completely decouple producers and consumers.

A notification broker is designed to provide highly scalable notification handling that eliminates coupling between producers and consumers. Notification brokers operate as intermediaries between producer and consumer applications, such that producers and consumers each know about the broker but do not know about each other. Because applications that produce or consume notifications are not normally designed to also fulfill large-scale notification requirements, brokers can improve system scalability by offloading the difficult aspects of notification handling (such as dealing with subscriptions, filtering, efficient delivery to multiple consumers, and message persistence). By doing so, brokers help keep infrastructure concerns separate from application concerns.

Conceptually, a notification broker is somewhat similar to a router: it accepts notification messages on the incoming side and sends them out again on the outgoing side. On the incoming side, the broker fulfills the `NotificationConsumer` interface; on the outgoing side, it fulfills the `NotificationProducer` interface. To a notification producer, the broker appears as a consumer, but to a consumer, it appears as a producer. One broker can even subscribe to another, which can be useful for distributing the processing load associated with topic filtering.

WS-Notification can support everything from simple event-driven systems to complex enterprise-scale multibroker notification systems.

Final Impressions

The WS-Notification family is extensive, making it hard to describe all its features in this column space. I've therefore tried to touch on the most important parts. Overall, though, the WS-Notification family is a winner. The specifications are well written and relatively easy to understand — largely because they borrowed tried-and-true approaches from previous notification and messaging systems rather than reinventing the wheel. WS-Notification can therefore support everything from simple event-driven systems to complex enterprise-scale multibroker notification systems.

Compared to WS-Notification, WS-Eventing is a simple subset, roughly equivalent to WS-Base Notification. Given this equivalence, I don't know why the companies that authored these specifications couldn't agree on that subset and avoid producing competing specifications. I hope WS-Base Notification and WS-Eventing can be combined sometime in the future.

Shifting Gears: Middleware Blogging

Before closing out this column, I want to touch on Web logging, or *blogging*. As most of you probably know, a blog is an online journal that not only allows writers, or *bloggers*, to easily publish content to the Web but also allows readers to add comments to each posting. Each blog is thus similar to a discussion board. Far more interesting, however, is that blogs tend to get linked together to form distributed discussion boards when various bloggers comment about postings they've read in other blogs. It is amazing how quickly blog-facilitated discussions can progress, and how widely they can spread — even across seemingly unrelated blogs.

Not surprisingly, an active middleware blogging community exists. I generally find the blogs by Werner Vogels (http://weblogs.cs.cornell.edu/AllThingsDistributed/), Sean McGrath (http://seanmcgrath.blogspot.com/), and Phil Wainewright (www.loosely coupled.com/blog/) to be very insightful. You can also read my own blog, "Middleware Matters" — play on words intended, of course — at www.iona.com/blogs/vinoski/. I use mine to post middleware-related commentary that goes above and beyond this column, such as information specific to IONA's products, calls for papers for conferences and workshops, or discussions and ideas that are not "cooked" enough to publish here. I also post pointers to my columns there, and have enjoyed the feedback I've gotten from readers and other bloggers.

When I posted a pointer to last issue's column, several bloggers posted comments about it. Mark Baker, in particular, posted several thought-provoking comments (www.markbaker.ca/2002/09/Blog/2004/03/11#2004-03-vinoski-notifications). He took issue with my assertions that URIs can't easi-

ly represent some transport mechanisms or multiprotocol endpoints. To solve the problem of using URIs to represent problematic transports, such as message queues whose parameters are not easily encoded in URI form, Baker suggested establishing a service that takes the message queue details as input, creates a resource to represent them, and returns an HTTP URI for that resource.

This same approach could be used to solve the multiprotocol issue. While this approach would certainly work, Baker and I agree that in many cases it's a fairly heavyweight addition to make to a production system. It introduces a level of distributed indirection for middleware subsystems that sets up and tears down communications channels, effectively requiring them to set up one temporary channel just to retrieve the information required to set up the channel to the actual target service. In addition, this approach potentially introduces a single point of failure if the communication resource service is not replicated. It also burdens production systems with another distributed moving part that requires additional deployment, management, and maintenance considerations.

Interestingly enough, the Corba Interoperable Object Reference (IOR) format[6] already solves the issue of directly representing all the communication details required for accessing multiprotocol services. Unfortunately, it does so by encoding arbitrary protocol-specific data into an unreadable hexadecimal digit string (at least in its "stringified" form) that generally scares most people away. It would be interesting to take the IOR approach — that is, to encode one or more communication profiles together into a single structure — and try to recast it into a URI scheme.

Baker's points are definitely worth additional consideration, and I certainly appreciate his taking the time to read this column and provide feedback. Other bloggers took issue with particular details of my WS-Addressing and WS-Eventing specifications review, or with my opinion regarding the unclear standardization paths for these specifications. You can find the details of these folks' views in my blog. I look forward to continuing to learn from other bloggers, and also to receiving further feedback on my columns.

Acknowledgments

Thanks to Chris Ferris and Steve Graham for reviewing a draft of this column.

References

1. S. Graham et al., "Publish-Subscribe Notification for Web Services," v. 1.0, joint specification by BEA Systems, IBM, and Microsoft, Mar. 2004; www-106.ibm.com/developerworks/library/specification/ws-notification/.
2. S. Graham et al., "Web Services Base Notification (WS-Base Notification)," v. 1.0, joint specification by BEA Systems, IBM, and Microsoft, Mar. 2004; www-106.ibm.com/developerworks/library/specification/ws-notification/.
3. S. Graham et al., "Web Services Brokered Notification (WS-Brokered Notification)," v. 1.0, joint specification by BEA Systems, IBM, and Microsoft, Mar. 2004; www-106.ibm.com/developerworks/library/specification/ws-notification/.
4. S. Graham et al., "Web Services Topics (WS-Topics)," v. 1.0, joint specification by BEA Systems, IBM, and Microsoft, Mar. 2004; www-106.ibm.com/developerworks/library/specification/ws-notification/.
5. 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/.
6. *The Common Object Request Broker: Core Architecture*, Object Management Group, OMG document no. formal/02-12-06, 2002.

Steve Vinoski is chief engineer of product innovation for IONA Technologies. He's been involved in middleware for 16 years. Vinoski is the coauthor of *Advanced Corba Programming with C++* (Addison Wesley Longman, 1999), and he has helped develop middleware standards for the OMG and W3C. Contact him at vinoski@ieee.org.