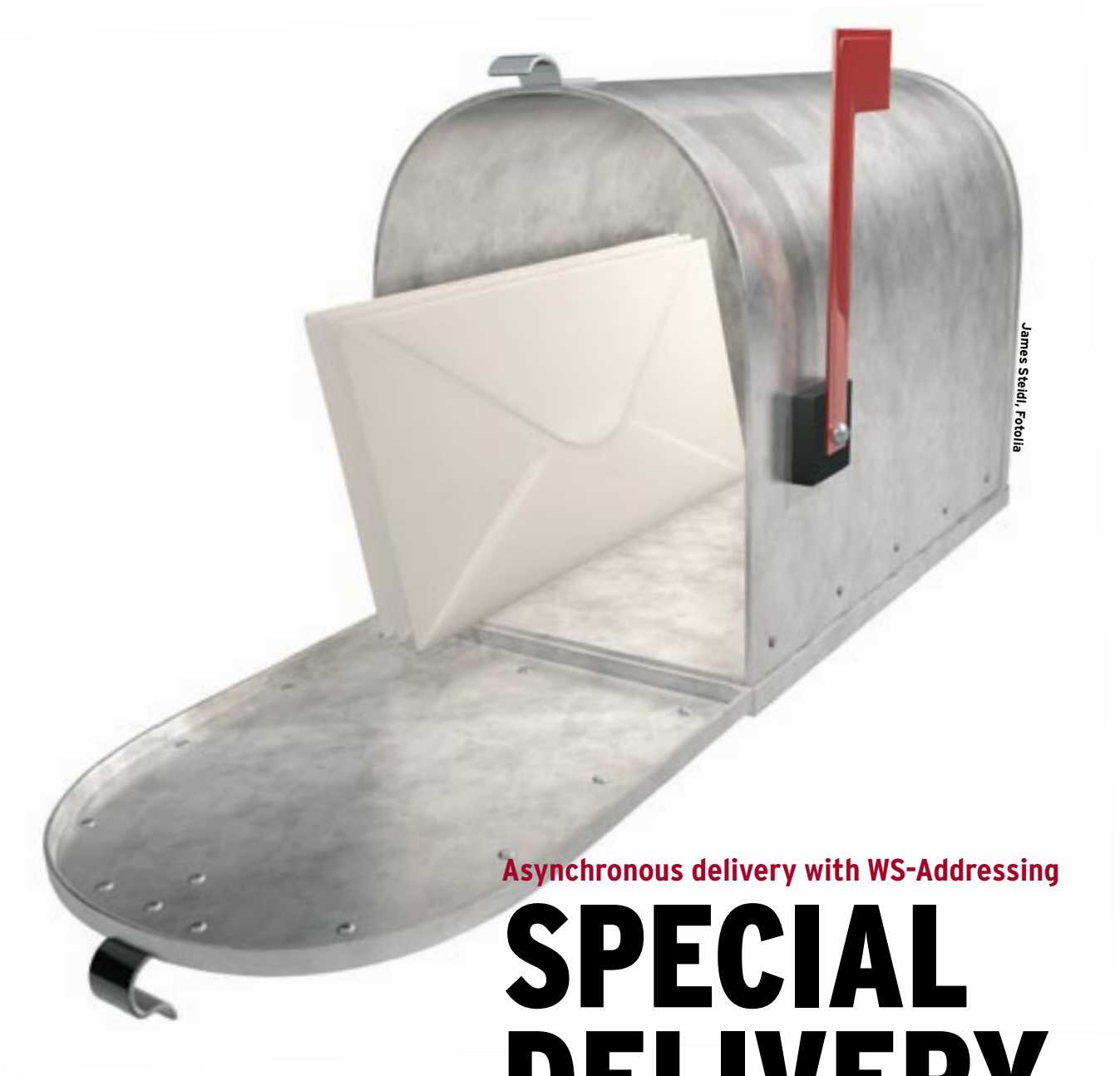

James Steidl, Fotolia

Asynchronous delivery with WS-Addressing

SPECIAL DELIVERY

WS-Addressing is a standard that enables flexible communication between web services. **BY DAVID HULL**

Two of the major standards bodies, OASIS and the W3C, recently released a flurry of standards dealing with web services. Because most of these standards have names that start with WS, the standards are loosely known as WS-* or (less formally) WS-splat.

In this article, I will examine one of the key pieces, WS-Addressing, and explain how it can enhance existing web service applications and ultimately bring powerful new messaging patterns into the web services world.

The W3C Web Services Addressing standard (or WS-Addressing, or just WSA) attempts to give web clients and servers – particularly those using SOAP – more flexibility in communicating with each other. WSA provides a standard way for defining an *Endpoint Reference* (EPR) – a structure that denotes the address of a service along with any other information needed for delivery.

WSA also defines a standard set of properties called *Message Addressing Properties* (MAPs) that are much like the headers on an email message. MAPs convey important data such as the sender's address, the receiver's address, a unique ID for the message, and addresses for replies and faults.

These basic facilities support a great variety of interactions. In this article, I will focus on one of the main use cases associated with WSA: the asynchronous request-response pattern. The WS-Addressing specifications of interest for this

THE AUTHOR

David Hull is a co-editor of the OASIS WS-BaseNotification standard. He participated in drafting of the W3C WS-Addressing standards and the SOAP one-way MEP. He blogs at http://*fieldnotesontheweb.blogspot.com*.

article are W3C recommendations, as is a "Metadata" document aimed at integrating WS-Addressing with WSDL and WS-Policy.

Asynchronous Request-Response

If you've ever enclosed a self-addressed, stamped envelope with a letter, you've participated in an asynchronous request-response. In other words, you have sent a request that includes a way for your counterpart to send a response back to you later. In the networking world it's the "self-addressed" part that is important. Stamps and envelopes are already provided.

Suppose I want a price quote for some service provided by Example Industries. Their web site at *www.example.com* could provide a front-end for accepting price requests, and it could even provide this as a SOAP service and advertise it using WSDL. If you want a quote, you send a request, and the server at *example.com* sends you back a response with the quote. Toolkits like Apache make it easy to generate the server code and WSDL from a business class you write. In practice, the client uses HTTP to *POST* a SOAP request, and the server sends back its SOAP response in the HTTP response. So far, so good.

In many industries, however, putting together a price quote requires some thought on the part of an estimator. If I'm requesting a price quote, I don't want to have to wait until the person who will respond to the quote is back in the office. Or, even if the estimator is in the office, I don't want to try to keep my HTTP connection open until the estimator is done thinking it over. I would really like to send in the request, hang up the connection, and have the reply come back to me later.

From a certain point of view, this is still very much a request-response operation. If I were using SOAP, I would send a SOAP request, and Example Industries would send me back a SOAP response, just not over the same connection. However, from the point of view of the web services stack (and the committees charged with specifying that stack) this is something different, since the response is no longer coming back on the same connection that delivered the request.

WS-Addressing

For many years after the introduction of SOAP, there was no standard way to imple-

Figure 1: The World Wide Web Consortium (W3C) develops standards for web technologies.

Listing 1: Asynchronous Request with WSA

```
01 POST /Widget HTTP/1.1
02 Host: estimates.example.org
03 Content-Type: application/soap+xml; charset=utf-8
04 Content-Length: nnn
05 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
06             xmlns:wsa="http://www.w3.org/2005/08/addressing">
07    <S:Header>
08     <wsa:MessageID>http://example.com/request-id-1</wsa:MessageID>
09     <wsa:ReplyTo>
10       <wsa:Address>http://example.com/business/client1</wsa:Address>
11     </wsa:ReplyTo>
12     <wsa:To>http://estimates.example.org/Widget</wsa:To>
13     <wsa:Action>http://example.com/EstimateRequest</wsa:Action>
14    </S:Header>
15    <S:Body>
16      <ex:EstimateRequest xmlns:ex=?http://example.com/estimate?>
17        <ex:Item>retro-confabulator</ex:Item>
18        <ex:Quantity>42</ex:Quantity>
19        <ex:Note>I need this in one standard galactic week</ex:Note>
20      </ex:EstimateRequest>
21    </S:Body>
22 </S:Envelope>
```

ment this pattern. Where do I put the return address for the response? In some new HTTP header? In a SOAP header? Called what? Maybe somewhere in the body of the request itself? What should the server send back as an HTTP response, since the real response is going to go over some other connection? Or should the server just close the connection without sending anything? Should I just give the return address as a raw URI? What if there is other information the server needs to know to deliver a response? If I'm sending several price requests to the same vendor, how can I tell which response goes with which request? Another HTTP header? A SOAP header? Something else?

Answers

WSA is aimed directly at addressing these questions. An EPR specifies a destination URI, together with other information needed for delivery, such as a set of SOAP headers to insert into the response message. This extra information provides a place to put cookies like transaction IDs, as well as policy assertions about security and reliability. EPRs act very much like function pointers or callback objects in programming languages, and they provide a similar degree of expressive power.

MAPs, which are represented in SOAP as header elements, carry the return address, a message id for correlation, and similar information. MAPs look quite a bit like the *reply-to* and *message-id* headers in email, and for good reason. The email headers work well in practice, so why not steal from the best?

The WS-Addressing core specification gives rules for the server to follow when it gets a message with MAPs attached. Basically, send the reply to the *reply-to* address (called the *[reply endpoint]* in the spec and *ReplyTo* on the wire) and mark the response with the *[message id]* of the request. The committee also drafted a note saying how to handle HTTP requests with no immediate response – namely by sending back a dummy message with HTTP status code 202. This is what most pre-standard implementations already did.

The Example in Action

So far, I haven't explained how the response is supposed to get back if not as the HTTP response. There are any number of ways the server might send the response. For instance, the server might use email or an instant messaging service to respond to the request.

I will assume the server is going to use a second HTTP connection. That is, when the response is ready, the server will open a connection back to the HTTP server I've designated in the *[reply end-*

Listing 2: POSTed Response to Request

```
01 POST /business/client1 HTTP/1.1
02 Host: example.com
03 Content-Type: application/soap+xml; charset=utf-8
04 Content-Length: nnn
05 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
06             xmlns:wsa="http://www.w3.org/2005/08/addressing">
07    <S:Header>
08     <wsa:RelatesTo>http://example.com/request-id-1</wsa:RelatesTo>
09     <wsa:To>http://example.com/business/client1</wsa:To>
10     <wsa:Action>http://example.com/EstimateResponse</wsa:Action>
11    </S:Header>
12    <S:Body>
13      <ex:EstimateResponse xmlns:ex=?http://example.com/estimate?>
14        <ex:Item>retro-confabulator</ex:Item>
15        <ex:Quantity>42</ex:Quantity>
16        <ex:Price type=?each? currency=?USD?>1000000</ex:Price>
17        <ex:Note>We have only 17 retro-confabulators in stock.  The
   rest will be shipped directly from the manufacturer</ex:Note>
18      </ex:EstimateResponse>
19    </S:Body>
20 </S:Envelope>
```

point] and *POST* the answer to me. The server will not be expecting a full-fledged HTTP response but will expect me to send back a dummy message with status 202 as it did when I *POST*ed my request. Listings 1 and 2 show what the resulting exchange might look like on the wire. Note the WSA header elements *ReplyTo* and *MessageId* in the request (Listing 1) and *RelatesTo* in the response (Listing 2).

At the application level, this exchange can be seen as two one-way messages: a request from me to the server, and a response from the server back to me. On the other hand, as far as HTTP is concerned, there are two request-response operations. Note that in neither of these messages is the HTTP server acting as a server in the usual sense. It is not serving some resource to the server at *example.com*. It is merely receiving a message and sending back a dummy response.

Mistakes Happen

Suppose I make a mistake in putting my request together. Maybe I leave out some element that's required for a price request. In such a case, I would expect a SOAP fault to come back.

There is no need for the estimator to get involved, or for me to wait for the estimator to do anything. The server can detect the error immediately and send me back an error directly on the HTTP response.

By default, WS-Addressing requires that faults be sent to the same destination as replies, that is, the *[reply endpoint]*. (The full story is a bit more involved, but the details aren't important here.) This behavior is not what I want in this case. If there's a fault, I want to know about it right away.

WS-Addressing provides a property just for this purpose, namely, the *[fault endpoint]* property, seen on the wire as *FaultTo*.

If I want faults sent differently from normal replies, all I have to do is give a different destination for them, and the server will know to send the faults to that destination instead of sending the faults as replies. The URI for "use the HTTP response" can't be the address I gave for the reply. Using that would require the server to open up a second connection for the fault, which is just what I don't want. WS-Addressing provides a special URI just for such cases, called the anonymous URI.

Listings 3 and 4 show what the result looks like on the wire. Note the addition of the *FaultTo* header, and note also that there is now only one connection, with the fault coming back as it always would have even without WS-Addressing.

Normal reply messages will still go to my HTTP endpoint, but faults come directly back.

Apache Axis

Implementing this technique with Apache Axis is simple. Axis now generates WSA-aware server code, so you don't need to do anything new on the server side.

If the Apache server code sees WSA SOAP headers, it follows the WSA rules. Otherwise, it behaves as it always has. You need to add two lines of code when

Listing 3: Request with FaultTo anonymous

```
01 POST /Widget HTTP/1.1
02 Host: estimates.example.org
03 Content-Type: application/soap+xml; charset=utf-8
04 Content-Length: nnn
05 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
06             xmlns:wsa="http://www.w3.org/2005/08/addressing">
07    <S:Header>
08     <wsa:MessageID>http://example.com/request-id-2</wsa:MessageID>
09     <wsa:ReplyTo>
10       <wsa:Address>http://example.com/business/client1</wsa:Address>
11     </wsa:ReplyTo>
12     <wsa:FaultTo>
13       <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</
   wsa:Address>
14     </wsa:FaultTo>
15     <wsa:To>http://estimates.example.org/Widget</wsa:To>
16     <wsa:Action>http://example.com/EstimateRequest</wsa:Action>
17    </S:Header>
18    <S:Body>
19      <ex:EstimateRequest xmlns:ex=?http://example.com/estimate?>
20        <ex:Item>retro-confabulator</ex:Item><!-- Oops, no quantity!
   -->
21      </ex:EstimateRequest>
22    </S:Body>
23 </S:Envelope>
```

setting up the client, but everything else works as before.

"Please don't post responses to the list. Email them to me at … and I'll summarize." From one point of view, the asynchronous request-response pattern is just a small variant on the usual request-response. I still send a request, and I still get a response – just delivered to me a different way.

From another point of view, however, the WSA and the asynchronous request-response pattern provide the jumping off point for a whole family of interactions. To implement the request-response variant, we have to introduce some fairly powerful machinery, including the equivalent of a function pointer. This machinery can be put to other uses, too.

One Response

In standard request-response, I know I will get exactly one response for each request (counting faults as responses). This will generally be true of asynchronous request-response as well, though it's now possible for the response to fail independently of the request.

The "one-request, one-response" rule doesn't have to hold in general. For example, I could broadcast a request message to any number of recipients, and they could respond – or not – as they see fit. I would then get zero or more responses for a given request. This approach would be good for soliciting bids, or for any number of "discovery" scenarios where I would like to find something by asking a group of possible providers.

Another important example is publication/subscription scenarios. One or more publishers sends information on a "topic," and zero or more subscribers could listen to that topic. For subscribers to get notifications sent to the topic, something will have to say where to deliver them. EPRs fit the bill perfectly – both the OASIS WS-Notification standard and Microsoft's WS-Eventing use EPRs for this purpose.

Conclusions

Apache Axis provides for a number of powerful request-response scenarios through its support of WS-Addressing and "one-way" messaging. Naturally, programmers have been writing code for responding to messages long before Axis or even HTTP.

The advantage of WSA is that now these responses can take a standard form using conventional Internet protocols and off-the-shelf software. ■

Listing 4: Response to faulty request

```
01 HTTP/1.1 400 BAD REQUEST
02 Content-Type: application/soap+xml; charset=utf-8
03 Content-Length: nnn
04 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
05             xmlns:wsa="http://www.w3.org/2005/08/addressing">
06    <S:Header>
07     <wsa:RelatesTo>http://example.com/request-id-2</wsa:RelatesTo>
08     <wsa:To>http://example.com/business/client1</wsa:To>
09     <wsa:Action>http://example.com/EstimateRequestFault</wsa:Action>
10    </S:Header>
11    <S:Body>
12      <S:Fault>
13       <S:Code>
14         <S:Value>MissingQuantity</S:Value>
15       </S:Code>
16       <S:Reason>
17         <S:Text xml:lang="en">No quantity given in request</S:Text>
18       </S:Reason>
19     </S:Fault>
20   </S:Body>
21 </S:Envelope>
```