About a year ago, IBM, BEA, and Microsoft put out a new specification called WS-Addressing (see the Related topics section below for a link). At only 14 pages, it doesn't claim to do anything more than to help standardize the way people specify the location of a web service. However, WS-Addressing has the potential to radically change the SOAP processing model. But how? And, more importantly, why? To better answer these questions, let's first examine what the specification itself actually says. (It should be noted that this discussion won't go into the details of actually implementing the specification; rather, I'll be discussing why its role in SOAP will be so critical in the future.)

If you get right down to the facts, after all the extra verbiage, WS-Addressing really only introduces two new concepts: an endpoint reference (EPR) and message information (MI) headers to the structure of SOAP.

**Endpoint references**

Before WS-Addressing came along, to find out where a web service was located, you had to ask for the URL of the endpoint or the WSDL. This sounds simple enough, so why do you need WS-Addressing? As web services evolve, it's only natural that they start to become more complex in order to support enterprise-level solutions. For example, it might be necessary to include additional information in a SOAP request that helps uniquely identify the web services instance that you're talking to. This would be akin to including a session identifier or instance ID on a request. Doing that wouldn't be a very big deal.

However, using a session identifier by appending something like ?resourceID=123 to the URL, would not be very SOAP-ish. A SOAP purist would more likely rather go back to the extensibility model defined in SOAP and use additional SOAP headers to include this new information to process the message; for example:
Technically, either method -- altering the URL of the request or including identification information in SOAP headers -- works fine for these purposes. However, if you use a SOAP header instead of encoding this extra information in the URL, you are no longer bound to one specific transport. Imagine a case where a SOAP message actually travels over more than one transport in order to get to its final destination. If both transports are HTTP, then the query string part of the URL can easily be reused. But what about some other transports, like SMTP? Where does the resourceID=123 go? Does it become an SMTP header, maybe? What if the transport doesn’t support user-defined fields? Or, even if it does, how would the ultimate receiver of the SOAP message (a SOAP engine) know how to pass on these user-defined fields to the web service?

The other problem with URL encoding is that it now tightly binds web service-specific data with the transport -- in other words, web service-specific data is now outside of the SOAP envelope and kept at the transport level.

So, how does WS-Addressing help? Well, it defines something called an endpoint reference (EPR). In its simplest form, an EPR is nothing more than a URL wrapped by some XML elements. Listing 1 shows an example.

**Listing 1. A simple EPR**

```xml
<widget:myLocation>
  <wsa:Address>http://localhost/WidgetService</wsa:Address>
</widget:myLocation>
```

Here, the widget:myLocation is the EPR, and inside of it is a WS-Addressing-defined element called wsa:Address which specifies the URL that should be used when trying to talk to the WidgetService.

**Listing 2. An EPR with an added ReferenceProperties element**

```xml
<widget:myLocation>
  <wsa:Address>http://localhost/WidgetService</wsa:Address>
  <wsa:ReferenceProperties>
    <widget:resourceID>123</widget:resourceID>
  </wsa:ReferenceProperties>
</widget:myLocation>
```

This is easy enough: you can add a new XML element called ReferenceProperties, and inside of it there is an element called resourceID (also known as a reference property) that contains your web service-specific data. This ReferenceProperties element encapsulates all web service-specific data. But now what do you do with this chunk of XML? Since you're using something called resourceID, continue with it and assume that the reason that you have this additional information is because there are multiple instances of Widgets used by the WidgetService. You can imagine that at some point you either created an instance or asked the service to return a pointer to one. Assume you asked for a new one to be created by invoking some createWidget() operation. But
how does the WidgetService return a reference to this new widget? As you might have guessed, it does so by returning an EPR, as shown in Listing 3.

Listing 3. Using an EPR to return a reference

```xml
<soap:Envelope...>
  <soap:Body>
    <widget:createWidgetResponse...>
      <widget:widgetReference>
        <wsa:Address>http://host/WidgetService</wsa:Address>
        <wsa:ReferenceProperty>
          <widget:resourceID>123</widget:resourceID>
        </wsa:ReferenceProperty>
      </widget:widgetReference>
    </widget:createWidgetResponse>
  </soap:Body>
</soap:Envelope>
```

Here, `widgetReference` is an EPR. The WS-Addressing specification requires the `Address` element, but `resourceID` is specific to the WidgetService.

You might be looking at this example and wondering why you don't just return the string "123" instead of an entire EPR -- why do you need the added complexity? There are actually a couple of reasons why the added abstraction actually makes life easier in the long run. First, the message's receiver does not need to know anything about the contents of the EPR. If you returned something as simple as "123," then the receiver would have to know not only that this is a string, but also what the string means and how it's supposed to use the string later on. The receiver could, potentially, be required to interpret the return value differently, depending on the type of application it's talking to. By using EPRs, you've standardized the format for referencing a web service and, more specifically, an instance of a web service as well.

The other important aspect of the EPR is the required element: `Address`. Because EPRs have this `Address` field, the field could contain a URL that is radically different from the URL of the original request. For example, the `createWidget()` call's URL could end with `.../createWidgetService`, while the subsequent service calls' URL could be `.../WidgetService`. Regardless of the specific values, the point is that the receiver of this EPR doesn't need to know anything about how to interpret the data; it just needs to know that the `Address` field is the URL that it should use for future messages, and that all of the reference properties should be SOAP headers on those messages. So, the SOAP envelope of future requests to this EPR would take the form shown in Listing 4.

Listing 4. SOAP envelope for requests to our sample EPR

```xml
<soap:Envelope...>
  <soap:Header>
    <wsa:To> http://host/WidgetService </wsa:To>
    <widget:resourceID>123</widget:resourceID>
  </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```
This envelope will be sent to http://host/WidgetService (I'll talk about the wsa:To header in a moment). Now, all of the information that the WidgetService needs to successfully process this SOAP message is available inside of the envelope, and nothing is transport-specific -- and, most importantly, the client-side application is totally unaware of the extra information that the message's receiver needs (such as the reference properties) to successfully process the message. All the client-side infrastructure needs to know is how, in a generic sense, to handle EPRs.

So, you've now standardized the way in which you pass around references to web services. Standardization is a good thing, but what you've seen so far hardly merits the claim that WS-Addressing will change SOAP itself. You'll get a better picture in the following sections covering the second feature of WS-Addressing: *message information* (MI) headers.

**MI headers**

The WS-Addressing specification defines some additional (and, of course, standardized) SOAP headers that should be used to help convey information about a message. In this section, I'll cover the more interesting ones.

**To**

To is nothing more than the target web service's URL. Typically, this URL is the same as the HTTP request's URL, but it is not required to be.

**Listing 5. To header**

```xml
<wsa:To> http://host/WidgetService </wsa:To>
```

The To header should be the same value as the `<wsa:Address>` element when using an EPR -- see the examples in the previous sections for more details.

**From**

From is an EPR of the message's sender. If the message's receiver needs to send a message back to the endpoint that sent the message, then it should use this EPR. Note that there's also a ReplyTo header, which indicates to where the response message should go. The From header would be used in cases where, like those governed by the WS-ReliableMessage specification, an Acknowledgement needs to be sent back to the sender. **Listing 6** shows an example of the From header in action.

**Listing 6. From header**

```xml
<wsa:From>
  <wsa:Address> http://client/myClient </wsa:Address>
</wsa:From>
```

**ReplyTo**

As mentioned, any response from the web service should be sent to the ReplyTo EPR. Because From and ReplyTo can be two distinct EPRs, the message's sender might not be the endpoint that is meant to receive the response. I'll discuss the implications of this later. **Listing 7** illustrates an example of the ReplyTo header's use.
Listing 7. ReplyTo header
<wsa:ReplyTo>
  <wsa:Address> http://client/myReceiver </wsa:Address>
</wsa:ReplyTo>

FaultTo
If the response to a message is a SOAP fault, the fault should be sent to the EPR in the FaultTo header. Listing 8 shows an example.

Listing 8. FaultTo header
<wsa:FaultTo>
  <wsa:Address> http://client/FaultCatcher </wsa:Address>
</wsa:FaultTo>

MessageID
The MessageID is nothing more than a URI that uniquely identifies a message. Listing 9 shows an example.

Listing 9. MessageID header
<wsa:MessageID>uuid:098765</wsa:MessageID>

Action
The Action header shown in Listing 10 is the in-envelope version of the SOAP HTTP Action header.

Listing 10. Action header

RelatesTo
RelatesTo will typically be used on response messages to indicate that it is related to a previously-known message and to define that relationship. Listing 11 shows an example.

Listing 11. RelatesTo header
<wsa:RelatesTo RelationshipType="wsa:Response">
  uuid:098765
</wsa:RelatesTo>

The RelatesTo header in Listing 11 indicates that this is a Response message for a previously-known Request message whose MessageID was uuid:098765. This header is critical in an asynchronous messaging scenario -- the response message’s receiver must be able to associate it with the original request message. The RelatesTo header provides a standard mechanism to do so.

Using headers
WS-Addressing defines all these special headers for use, but you might ask yourself: “Who cares?” Initially, it might seem like you’ve actually added a level of complexity that, up until now,
has not been needed. For simple request/response HTTP message flows, that is probably true. However, once you start to use asynchronous message flows or send messages across multiple transports, these headers become very important.

Take the simple case of an asynchronous message exchange (even over HTTP) where the response message is sent back over a new connection. How does the server know where the next response message needs to go? One obvious solution would be to include the return address (or EPR) as part of the application data (that is, in the Body). However, if you use WS-Addressing, you now have a standard location for this data: the ReplyTo MI header/EPR. Likewise, if you're working with a specification like WS-ReliableMessaging (see Related topics), one where there are potentially three different response messages (a traditional Reply, a possible Fault, and an Acknowledgement), WS-Addressing offers a standardized mechanism through which all web services can express this information. This becomes especially critical when these three messages might need to be sent to three different endpoints. It's not unlike the way there is a standard location on a physical envelope for the mailing address, return address, and stamp. Standardization is a good thing.

Anonymous URIs

In all of the headers that either contain an Address element or are, like the To header, an address themselves, you can use a special anonymous URI:

http://schemas.xmlsoap.org/ws/2003/03/addressing/role/anonymous

When you use this URI, you are indicating that there is no real endpoint available for this address. This means that a connection to that endpoint cannot be opened. This actually has some very interesting side effects on message flow. For example, let's say you use the anonymous URI in the ReplyTo header's address. What does this mean? Clearly, the client doesn't want a new connection opened -- it's not a real URI. Using the anonymous URI is equivalent to not using the ReplyTo header at all -- in other words, the response must flow back in the HTTP response message.

While you might initially view this example as a mildly interesting use of ReplyTo and the anonymous URI, it actually has a very large impact on the message processing model. Consider the case where a request message looks like the one in Listing 12.
Notice that this message’s sender is asking for the response message to be sent back over the HTTP response flow (as the anonymous URI indicates in the \texttt{ReplyTo} header). This means that the client who sent this code is assuming a synchronous processing model, right? Maybe. Look at the \texttt{FaultTo} header -- that’s a real, non-anonymous, endpoint. So, if this message’s receiver generates a SOAP fault, it is supposed to send that fault to \texttt{http://client/myReceiver} and not over the HTTP response flow -- and that’s an asynchronous processing model. Think about what this really means: the client, in certain cases, might not control whether a message is sent back synchronously or asynchronously. This could have a very large impact on SOAP engines. No longer can they assume one processing model per call -- they must be able to switch between the two based on what they get back from the server. So, in this case, the client might wait for a response on the HTTP response flow. Failing to get one, it will then wait for the response to arrive asynchronously at the \texttt{http://client/myReceiver} endpoint. This dynamic switching might not be a trivial task for some current SOAP processors.

Of course, it’s not just the client side that is affected. The receiver (or server) must be able to dynamically switch between synchronous and asynchronous modes as well, based on the values in the various headers and the response message generated.

There’s another interesting side effect of this. In most services, the WSDL describes a very simple request/response synchronous message pattern. But with WS-Addressing, any web service, whether it was defined asynchronously or not, will implicitly have asynchronous support simply through the correct use of the WS-Addressing headers. That is a very powerful tool now available to SOAP applications. Certain specifications, such as WS-Transactions (WS-Coordination and WS-AtomicTransaction -- see \textit{Related topics}), go out of their way to define two different port types -- one for synchronous and one for asynchronous messaging patterns. This distinction now becomes obsolete.

The Emerging Technologies Toolkit (ETTK; see \textit{Related topics} for a link) contains quite a few demos use the WS-Addressing specification. In particular, the web-based WS-ReliableMessaging demo lets you easily change the WS-Addressing headers to see how the message flow changes.
Conclusion

The most common reason to support a WS-* specification is because of the level of standardization it brings to the web services world. WS-Addressing definitely does this through the definition of EPRs and MI headers. This definition provides a single mechanism through which people are able to specify both the location of a web service (or instance of a service) and the ways in which services use those EPRs in SOAP messages (via MI headers). But it's the implied changes in the message processing model/flow that have even more important implications for SOAP. WS-Addressing's importance will grow over time -- so much so that it will be viewed as one of those specifications that should have been part of the core SOAP specification itself.
Related topics

- You can download the various web services specifications discussed in this article:
  - WS-Addressing
  - WS-Coordination
  - WS-AtomicTransaction
  - WS-ReliableMessaging
- Check out the IBM Emerging Technologies Toolkit.

© Copyright IBM Corporation 2004
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)