Tip: Use XML directly over HTTP for Web services (where appropriate)

Taking the direct approach

Uche Ogbuji

January 15, 2004

SOAP technologies usually exchange XML over HTTP, but SOAP has its pros and cons, and a lot of discussion has focused on how to use XML more directly to communicate between applications. This tip describes the direct approach, and discusses where it is most appropriate. It also discusses how to use WSDL to describe such services.

View more content in this series

SOAP provides a means for systems to package and exchange messages encoded in XML. SOAP is designed to be the base layer of a broad array of capabilities that are required by Web services systems, but as such SOAP is not the most direct way to exchange XML over Internet protocols. The basic Web architecture makes it possible to exchange more than just HTML pages. Obvious items are images and popular formats such as PDF. Other systems can grab XML directly from the Web: RSS, Atom, Creative Commons, XSLT stylesheets, SVG, and more. And of course you can create your own services by placing an appropriate XML file on a Web server. You can even support write operations by allowing HTTP POST, PUT, or DELETE commands. And as I shall demonstrate in this tip, you can use WSDL to describe these simple services. Interestingly enough, the way that most SOAP systems exchange WSDL files by hosting on Web servers shows how such basic Web principles are used to bootstrap the entire world of Web services.

Some experts use the name Rep resentational State Transfer (REST) in describing services that rely on only the core Web architecture. You'll find no hard and fast rules as to when to use SOAP and when to use REST, but it is important to always at least consider REST for Web services tasks, and not just use SOAP. A huge installed base of tools and services understand REST implicitly, and although SOAP is a fast-growing technology, it is still far behind in terms of running code. Web proxies, spiders, crawlers, and other agents, as well as universally deployed browsers and the like, can already process XML delivered plainly over HTTP.

It's also important to consider all the other XML technologies that can process XML retrieved over HTTP, FTP, or other means. You can easily incorporate XML retrieved from a RESTful service using the standard document() function, but the only way to incorporate a SOAP request is through
proprietary extensions. XML documents that are available through RESTful services can be included using XInclude, or linked to using XLink or even basic HTML and XHTML links. None of this is straightforward through SOAP.

**The role of WSDL**

You can still use WSDL even though you are not using SOAP. WSDL has always supported bindings for conventions other than RPC-style SOAP over HTTP, with which it is most closely associated. These alternative bindings, however, are not used as often as they could be. Part of the problem is WSDL's complexity: It can feel like overkill for describing simple Web access. Listing 1 shows a WSDL 1.2 example for representing the usual approach of RSS services: simple download.

**Listing 1. WSDL 1.2 example for describing RSS available RESTfully through the Web**

```xml
<?xml version='1.0' encoding='UTF-8'?>
<definitions name='rss-service'
  targetNamespace='http://example.com/rss-service/
  xmlns='http://www.w3.org/2003/06/wsdl'
  xmlns:http='http://www.w3.org/2003/06/wsdl/http'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <message name='rss-request'>
    <part name='feed' type='xsd:string'/> 
  </message>
  <message name='retrieved-rss'>
    <part name='rss-doc' mimeType='application/rss+xml'/> 
  </message>
  <interface name='rss-interface'>
    <operation name='retrieve-rss'>
      <input name='rss-request'/> 
      <output name='retrieved-rss'/> 
    </operation>
  </interface>
  <binding name='rss-interface-http1' type='rss-interface'>
    <http:binding verb='GET'/>
    <operation name='retrieve-rss'>
      <http:operation location='syndication/{feed}.rss'/>
      <input>
        <!-- no parameters other than the one already worked into the URL -->
      </input>
      <output>
        <!-- no parameters because response is explicitly the HTTP response -->
      </output>
    </operation>
  </binding>
</definitions>
```

The *message* elements represent the actual HTTP request and response. The *part* elements represent the information that provides the specifics over the HTTP exchange. For the request,
I parameterize the name of the RSS feed filename. This way I can use the same WSDL to cover an RSS 0.91 feed as well as an RSS 1.0 feed. The binding is what specifies that this description covers low-level HTTP 1.1. The \texttt{location} attribute on \texttt{http:operation} maps the \texttt{feed} parameter to its role in selecting the file name on the server. The square brackets are a WSDL 1.2 convention not unlike attribute value templates: The contents are replaced by the message part of the given name.

**Keep it simple**

Web services spring from a very grand vision. When the entire vision is realized and all the parts are widely available, much of the complexity will be justified. But as things are now, sometimes it is better to use the simpler conventions that brought the Web explosive growth a full decade before the advent of Web services. If XML served over simple HTTP 1.1 can possibly work in solving a problem, it's well worth considering.
Related topics

- Check out "Deploying Web services with WSDL" by Bilal Siddiqui (developerWorks, November 2001). This article covers an older version of WSDL.
- See the current specification Web Services Description Language (WSDL) Version 1.2, and especially Web Services Description Language (WSDL) Version 1.2 Part 3: Bindings. The bindings include SOAP, HTTP, and IME (e-mail, basically).
- Find a broad array of articles, columns, tutorials, and tips on these two popular technologies at the developerWorks Web services and XML content areas. For a complete list of XML tips to date, check out the tips summary page.
- Learn how you can become an IBM Certified Developer.