Table of Contents

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

Chapter 9. SOAP Interoperability and WSDL

All of the examples to this point have used the same SOAP implementation for both client and server. Our Apache SOAP server examples were accessed using Apache SOAP client applications, and our GLUE server examples were accessed by GLUE client applications. In those rare occasions when you have control over the technology used at every node of a distributed system, it's easiest to use the same technology throughout. However, that opportunity doesn't present itself all that often, and it's fundamentally at odds with the web services vision of the computing world. So it's necessary to investigate how SOAP implementations interoperate with each other.

There are dozens of SOAP implementations available right now, and others will be showing up every day. Over time, you'll probably find lots of existing enterprise systems making themselves accessible via a SOAP mechanism over a variety of transports. Some systems will undoubtedly use SOAP under the covers, so you won't have to deal with it at all. And new distributed software frameworks based on SOAP will certainly sprout up. How well can we expect these systems to interoperate with one another? After all, software is still developed largely by companies and individuals that are competing in one way or another, which usually leads to problems with interoperability. Sometimes the problems arise because the specification has one or more sections that are open to interpretation, and developers working for different organizations interpret things differently. The fact that these problems occur doesn't necessarily indicate that the specification is flawed; it only indicates that software development is ultimately a human enterprise, and humans are prone to disagree.

9.1. Web Services Definition Language

One way to avoid interoperability problems with SOAP-based services is to use a structured language to describe the service, its location, the service methods, parameters, data types, and so on. The Web Services Definition Language (WSDL) does just that. WSDL is an XML grammar for describing web services. Systems can determine the programmatic interface of a web service by looking at the WSDL document associated with that service. The document describes the service methods along with their parameters and return types, and may also include the address, or endpoint, of the service. One of the greatest benefits of WSDL is that it is a single, accepted standard[1] for describing web services, which among other things motivates SOAP developers to avoid using their own mechanism for that task.

[1] You can find the WSDL specification at http://www.w3.org/TR/wsdl.

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

Does WSDL eliminate the problems associated with human (or machine) interpretation? Well, not completely. It does mean that service descriptions are far less ambiguous than they would be if there were no standard, but on the other hand, WSDL has a specification of its own, and that specification has the potential to be interpreted differently by multiple parties. WSDL isn't a perfect solution, but it certainly puts those of us working with these technologies in a better place than we'd be in otherwise.

9.2. Calling a GLUE Service from an ApacheSOAP Client

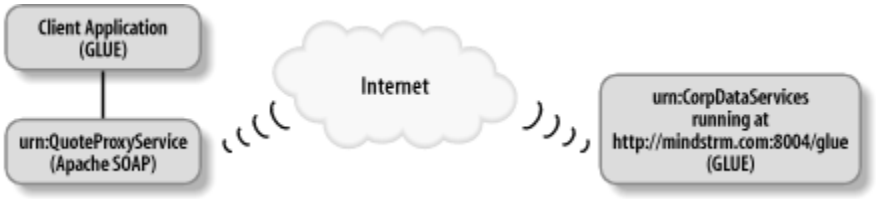
Since we already have a GLUE service running, let's try to access it with a client written using Apache SOAP. It's nice of the service to publish WSDL, but Apache SOAP clients can't do anything with that data. So we'll have to derive whatever information we need from the WSDL ourselves. Before we get into that, though, let's write a quick application that tries to call the `getHeadlines` method of the `urn:CorpDataServices` service.

First we need to determine the proper URL. When we were calling Apache SOAP services, the URL pointed to the rpcrouter servlet. In this case, a GLUE server application is hosting the service, so the URL we want is *http://mindstrm.com:8004/glue/urn:CorpDataServices*.

9.3. A Proxy Service Using Apache SOAP

I'm going to use *http://mindstrm.com:8004/glue/urn:CorpDataServices* as the backend for the services we develop through the rest of this chapter. In this example we'll create an Apache SOAP service called `urn:QuoteProxyService` that gets its data from the GLUE-based service we've been working with. In other words, this new Apache SOAP service acts as a proxy to the quote retrieval part of the GLUE-based service. This setup is depicted in Figure 9-2.

Figure 9-2. A proxy service



9.4. Calling an Apache SOAP Service from a GLUE Client

This time, we'll write a GLUE client that accesses the `urn:QuoteProxyService` we just created. But how do we begin? GLUE creates service bindings dynamically based on a WSDL document. In our GLUE-to-GLUE examples, GLUE automatically generated the WSDL document describing the service, but Apache SOAP doesn't do that. We could certainly write an appropriate WSDL document by hand . . . but that's a lot of work, and the chances of ending up with a WSDL document that's correct are fairly small, unless you

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

want to learn a lot more about WSDL than anyone should have to know. Instead, we'll use the `java2wsdl` tool that comes with GLUE to generate the WSDL.[4] This tool doesn't care that the Java code isn't for a GLUE-based service; it simply reads the Java code and generates the WSDL. `java2wsdl` doesn't give us a perfect WSDL document for the Apache-based service, but it's much less work to modify the document by hand than it is to write the whole thing. So let's generate the WSDL:

[4] Other tools for this purpose are available from other sources. For example, the IBM web services toolkit includes a similar utility.

```
java2wsdl javasoap.book.ch9.services.QuoteProxyService -n
    urn:QuoteProxyService -t urn:QuoteProxyService -s
    -e http://georgetown:8080/soap/servlet/rpcrouter
```

9.5. Accessing .NET Services

Although it's not based on Java, there's no way to avoid a discussion of Microsoft's .NET technology. If you're interacting with web services, you'll certainly run into .NET more than once. To show how to achieve interoperability between .NET and web services written in Java, I've written a .NET web service using the C# language and published it on the Internet at http://mindstrm.com:8199/CorpDataService/Proxy.asmx.[7] For those of you working with .NET, it's necessary to specify that your service style is RPC, since that is not the .NET default service style. To do so, use the `[SoapRpcService( )]` declaration in your C# code. My .NET service acts as a proxy to the `urn:CorpDataServices` service running at *http://mindstrm.com:8004/glue/urn:CorpDataServices*. Figure 9-3 depicts the relationships involved.

[7] The code for the service is located at http://www.mindstrm.com/soap.htm.

Figure 9-3. Interactions between a client, a proxy, and a service

9.6. Writing an Apache Axis Client

The Apache group is currently working on a completely new SOAP implementation known as Axis.[8] It's really too early in the evolution of that project to spend much time on it. It's likely to change quite a bit before it stabilizes, and right now it's not ready for production use. The plan is for Axis to conform to JAX-RPC, an emerging API standard for SOAP RPC. Nevertheless, Axis is out there now and can be used to experiment a bit. So let's take a brief

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

look at Axis and get a feel for how it's likely to work. Later, in Chapter 11, we'll take a look at JAX-RPC.

[8] The actual name is currently being debated, but as of this writing it is still called Axis. The examples in this book are based on the Axis release dated December 5, 2001. Obviously, you can expect changes to the APIs as things develop.

The heart of the client is `Axis2DotNet`. It invokes the `getHeadlines` method on the .NET service that we used before.

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01