

Notes on SOAP

March 11, 2003

1 Intro

This document is designed to give you an overview of SOAP, focusing on Apache Axis (a particular implementation of a SOAP API), and the things you'll need for your project. For comprehensive coverage of SOAP, check out (among many others) :

- <http://www.w3.org/TR/SOAP/> - The W3C's page on the ongoing efforts to develop a SOAP standard.
- <http://soap.weblogs.com/> - Hosted by Userland. Forums and articles about SOAP.
- Programming Web Services with SOAP. Kulchenko, Snell, and Tidwell. O'Reilly Books.

2 What is SOAP?

SOAP stands for Simple Object Access Protocol. It's designed to do two things: provide a communications protocol for distributed computation, and handle the marshalling and exchange of data across platforms.

SOAP is an XML-based protocol. What that means is that SOAP messages are XML documents, parsed and processed just like any other XML document.

3 What is Axis?

Axis is a set of SOAP-aware tools that make the development of applications that use SOAP easier. Axis hides the details of a SOAP message from the programmer. It also can generate stub code from a WSDL file (more on that in a bit). It also includes several other pieces, including a standalone server and the ability to deal with EJBs and JavaBeans, that we won't deal with directly; check out the Axis documentation for more info.

4 What is a SOAP message?

Here's a simple SOAP request for a remote stock-quote service, taken from the O'Reilly book mentioned above:

```
<s:Envelope
  xmlns:s="http://www.w3.org/2001/06/soap-envelope">
  <s:Header>
    <m:transaction xmlns:m="soap-transaction"
      s:mustUnderstand="true">
      <transactionID>1234</transactionID>
    </m:transaction>
  </s:Header>
  <s:Body>
    <n:getQuote xmlns:n="urn:QuoteService">
      <symbol xsi:type="xsd:string">
        IBM
      </symbol>
    </n:getQuote>
  </s:Body>
</s:Envelope>
```

Notice that it's a standard XML document- the root element is the Envelope, which has a namespace called 's'. The envelope contains a Header, which indicates the type of transaction and the ID, and a Body, which indicates the requested service. In this case, the service is to get a Quote for the symbol IBM. A standard reply would look like this:

```
<s:Envelope
  xmlns:s="http://www.w3.org/2001/06/soap-envelope">
  <s:Body>
    <n:getQuoteResponse
      xmlns:n="urn:QuoteService">
      <value xsi:type="xsd:float">
        98.06
      </value>
    </n:getQuoteResponse>
  </s:Body>
</s:Envelope>
```

Here the server returns the quote 98.06 in the Body of a SOAP Envelope. (There's no Header here, since this is a direct response.)

For our purposes, this is all we'll need to know about the structure of SOAP messages. Luckily, we won't have to deal with SOAP directly; instead, we can use the classes provided by Axis.

5 Discovering Services

So far, we've seen how SOAP provides a protocol for distributed communication, but we haven't talked about how a client knows what services are available. How did the requester in the above example know that he should use QuoteService to get a quote?

Providers of Web Services can publish their services using an XML-based (what else?) language known as WSDL, for Web Services Description Language. A WSDL file specifies the data structures that a server recognizes, the sequence of input-output messages corresponding to different types of requests, and the sorts of requests that are allowed. Here are a few snippets from Amazon's WSDL.

```
<xsd:complexType name="ProductInfo">
  <xsd:all>
    <xsd:element name="TotalResults" type="xsd:string" minOccurs="0"/>
<!-- Total number of Search Results -->
    <xsd:element name="TotalPages" type="xsd:string" minOccurs="0"/>
<!-- Total number of Pages of Search Results -->
    <xsd:element name="ListName" type="xsd:string" minOccurs="0"/>
<!-- Listmania list name -->
    <xsd:element name="Details" type="typens:DetailsArray" minOccurs="0"/>
  </xsd:all>
</xsd:complexType>
```

This specifies that the ProductInfo data structure will have four fields: a String TotalResults, a String TotalPages, a String listName, and an array of Details nodes called Details.

```
<message name="KeywordSearchRequest">
<!-- Messages for Amazon Web APIs -->
  <part name="KeywordSearchRequest" type="typens:KeywordRequest"/>
</message>
<message name="KeywordSearchResponse">
  <part name="return" type="typens:ProductInfo"/>
</message>
```

This specifies the messages sent for a Keyword Search and the namespaces they will use. The client will send a KeywordRequest data structure, and receive back a ProductInfo data structure.

```
<portType name="AmazonSearchPort">
  <!-- Port for Amazon Web APIs -->
  <operation name="KeywordSearchRequest">
    <input message="typens:KeywordSearchRequest"/>
    <output message="typens:KeywordSearchResponse"/>
  </operation>
  ...
```

```

</portType>
<binding name="AmazonSearchBinding" type="typens:AmazonSearchPort">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <!-- Binding for Amazon Web APIs - RPC, SOAP over HTTP -->
  <operation name="KeywordSearchRequest">
    <soap:operation soapAction="http://soap.amazon.com"/>
  <input>
    <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://soap.amazon.com"/>
  </input>
  <output>
    <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://soap.amazon.com"/>
  </output>
  </operation>
</binding>

```

this last snippet tells a SOAP client the messages that will be sent for different operations (in this case a Keyword Search, the URL that the message must be sent to, and how the message and reply will be encoded. With this information, the client now knows all about the services that a Web Service provider has exposed.

So what to do with all this? Well, fortunately, XML is a very regular, well-defined language. This means that it's easy to take this description and transform it into Java code. And that's just what Axis does.

6 Using Axis to generate Java code

Axis provides a class called WSDL2Java that will take as input a WSDL file and generate Java classes that provide the analogous data structures and messages. If you've run the client.axis.sh script provided with the Amazon toolkit, axis has already done this. The relevant Java command is:

```

java org.apache.axis.wsdl.WSDL2Java -v -p com.amazon.soap.axis
AmazonWebServices.wsdl

```

this assumes that you've already downloaded Axis and have the Axis jar in your CLASSPATH. It will produce the package com.amazon.soap.axis from the file AmazonWebServices.wsdl.

At this point you can now use the Java classes to build your SOAP messages; the code included with the Amazon toolkit shows how to build, send, and process SOAP requests. At this (Java) level, the fact that there's any sort of distributed computing going on is completely hidden from you, the programmer. Everything just looks like normal library calls.

You just create and fill in the parameters of your request, create an AmazonSearchService and an AmazonSearchPort, and submit the request to the port. Amazon will return a reply, wrapped inside a SOAP object. (typically inside the SOAP object.)