Accessing and Interacting with Remote SOAP-enabled Services

Reference:

1. Articles by Qusay H. Mahmoud, http://www.sun.com

SOAP

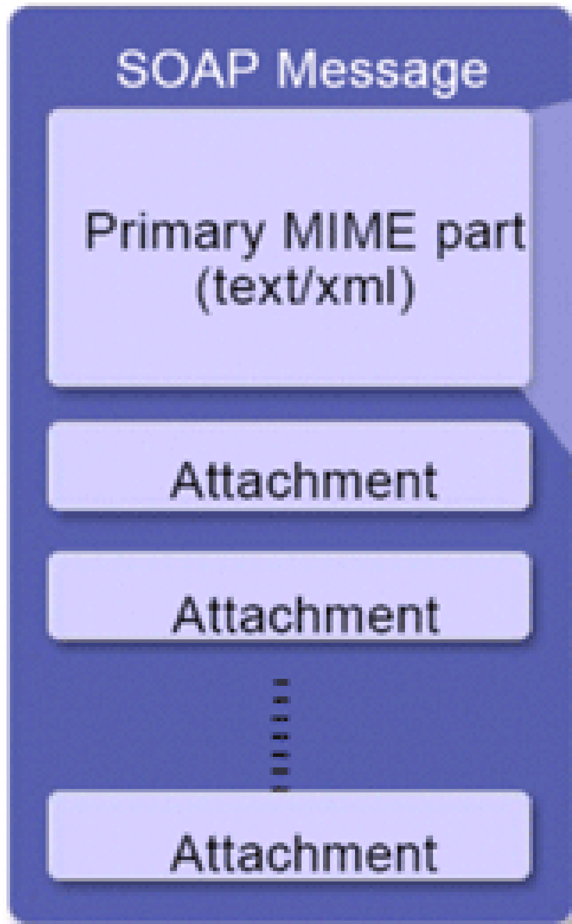
- SOAP can be used on the server-side and client-side. In other words, it is useful for low-level as well as high-level developers.
- It is useful for service-oriented infrastructure, allowing low-level distributed enterprise systems developers to turn applications into SOAP-enabled services that can be accessed, remotely, from any device.

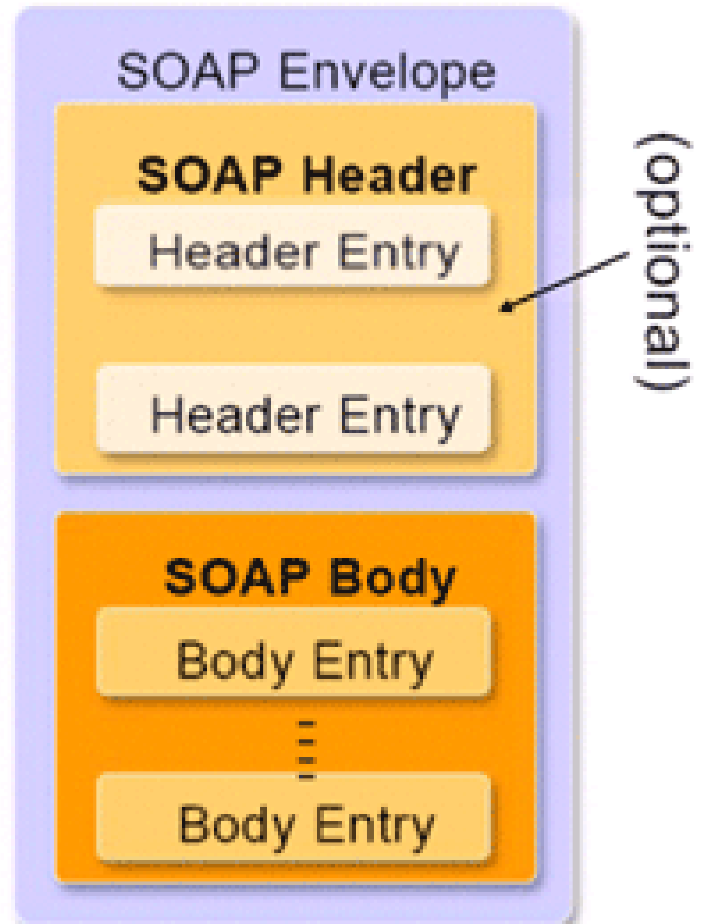
SOAP

- This can be accomplished by introducing a web service that understands SOAP, and the end result would be that applications can interoperate.
- It is useful for high-level application developers who wish to develop clients that can access and interact with SOAP-based services...such as the ones offered by XMethods, Google, and Amazon

SOAP Messages

- The basic unit of interaction between a SOAP client and a SOAP-enabled service is a message.
- A SOAP message is basically an XML document that consists of an *envelope* enclosing any number of optional *headers*, a *body*, and any optional MIME attachments as shown in the next figure.


SOAP Message
Primary MIME part
(text/xml)
Attachment
Attachment
Attachment


SOAP Envelope
SOAP Header
Header Entry
Header Entry
(optional)
SOAP Body
Body Entry
Body Entry

SOAP Envelope

- It is the root element of the XML document. This element falls under the http://schemas.xmlsoap.org/soap/envelope namespace. An envelope is uniquely identified by its *namespace*, and therefore processing tools can immediately determine if a given XML document is a SOAP message.
- But this capability or convenience comes at an expense -- you cannot send arbitrary XML documents; well, yes you can embed such documents in the Body element, but this requires XML validation with the web services engine.

SOAP Headers and Body

- *SOAP Headers*: They are the primary extensibility mechanism in SOAP. Using headers, SOAP messages can be extended with application-specific information like authentication, authorization, and payment processing.
- *SOAP Body*: It surrounds the information which is core to the SOAP message. Any number of XML elements can follow the Body element. This is a nice extensible feature that can help with the encoding of data in SOAP messages.

Attachments

- They can be entire XML documents, XML fragments, text documents, images, or any other content with a valid MIME type.

SOAP Message Syntax

- To get an idea of what a SOAP message looks like, let's start by looking at the following simple SOAP message:

```
<Envelope
  xmlns="http://schemas.xmlsoap.org/soap/envelope
  /">
<Body>
<getStockPrice/>
</Body>
</Envelope>
```

SOAP Message Syntax

- Here, the XML Namespace (xmlns) was used to identify the Envelope as a SOAP Envelope. Another important thing to note in the above segment of SOAP code is that the getStockPrice element is in the default namespace.
- XML Namespaces can be used to specify which getStockPrice procedure or method should be used, as in the following example:

SOAP Message Syntax

```
<Envelope
   xmlns="http://schemas.xmlsoap.org/soap/envelope
   /">
<Body> <getStockPrice
   xmlns="http://finance.sometradingcompany.com"/>
   </Body>

</Envelope>
```

SOAP Message Syntax

- If you do qualify it, the end result may look as follows, but this doesn't change the real meaning of the message.

SOAP Message Syntax

```
<SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/"
  >
<SOAP-ENV:Body> <m:getStockPrice
  xmlns:m="http://finance.sometradingcompany.com"
  />
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

SOAP Message Syntax

- Input is passed to services through arguments.
- For example, the above SOAP message can be made to look more professional by adding an argument that represents the name of the stock as follows:

SOAP Message Syntax

```
<SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/en
  velope/">
<SOAP-ENV:Body> <m:getStockPrice
  xmlns:m="http://finance.sometradingcomp
  any.com">
<symbol>RIL</symbol>
</m:getStockPrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Message Syntax

- Note that the above SOAP message has no header. An empty header can be added as follows

SOAP Message Syntax

```
<SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/en
  velope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body> <m:getStockPrice
  xmlns:m="http://finance.sometradingcomp
  any.com"/>
<symbol>RIL</symbol>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP with Attachments API for Java (SAAJ)

- The SAAJ API allows you to read, write, send, and receive SOAP messages over the Internet. The examples here show you how to create a SOAP connection, a SOAP message, populate the message, send the message, and receive the reply.
- This API (version 1.2) conforms to SOAP 1.1 and SOAP with Attachments, and supports the WS-I basic Profile.

SAAJ

- The APIs defines the javax.xml.soap package that provides the classes needed for
  - creating and populating SOAP messages,
  - extracting content from SOAP messages,
  - sending SOAP request-response messages, and
  - accessing/adding/modifying parts of SOAP messages.

Creating a SOAP Message and Handling the Response

- In SAAJ, SOAP messages are sent and received over a connection that is represented by a SOAPConnection object.
- The following segment of code shows how to create a SOAPConnection object; it is a good practice to close a connection once you're finished using it, since this will release resources:

Creating a SOAP Message and Handling the Response

```
SOAPConnectionFactory scf =
   SOAPConnectionFactory.newInstance();
   SOAPConnection sc = scf.createConnection(); //
   close the connection sc.close();
```

Creating a SOAP Message and Handling the Response

- Note that an application can send SOAP messages directly using a SOAPConnection object, or indirectly using a messaging provider such as JAXM.
- A message can be created using the MessageFactory object as follows:

Creating a SOAP Message and Handling the Response

```
MessageFactory mf =
  MessageFactory.getInstance();
  SOAPMessage msg = mf.createMessage();
```

Creating a SOAP Message and Handling the Response

- The message created, msg, already contains empty basic parts (envelope and header), and these can be retrieved as follows.
- Note that the SOAPPart contains the envelope, which in turn contains the header and the body.

Creating a SOAP Message and Handling the Response

```
SOAPPart sp = msg.getSOAPPart();
SOAPEnvelope envelope = sp.getEnvelope();
SOAPHeader header = envelope.getHeader();

SOAPBody body = envelope.getBody();
```

Creating a SOAP Message and Handling the Response

- Another way to access the parts of the message is by retrieving the header and the body directly as follows:

Creating a SOAP Message and Handling the Response

```
SOAPHeader header = msg.getSOAPHeader();

SOAPBody body = msg.getSOAPBody();
```

Creating a SOAP Message and Handling the Response

- As mentioned earlier, the header is optional and thus, if you are not using it, you can delete it as follows:

Creating a SOAP Message and Handling the Response

```
header.detachNode();
```

Creating a SOAP Message and Handling the Response

- The next step is to populate the SOAPBody with the actual message to be sent. In doing so, you simply create an element specifying the message to be invoked and its argument(s).
- Here is an example:

Creating a SOAP Message and Handling the Response

```
Name bodyName =
  sf.createName("getStockPrice", "m",
  "http://finance.sometradingcompany.com"
  );
SOAPBodyElement bodyElement =
  body.addBodyElement(bodyName);
Name name = sf.createName("symbol");
SOAPElement symbol =
  bodyElement.addChildElement(name);
symbol.addTextNode(“RIL");
```

Creating a SOAP Message and Handling the Response

- This segment of code will create a SOAP message that looks as follows:

Creating a SOAP Message and Handling the Response

```
<SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/"
  >
<SOAP-ENV:Body> <m:getStockPrice
  xmlns:m="http://finance.sometradingcompany.com"
  />
<symbol>RIL</symbol>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Creating a SOAP Message and Handling the Response

- Before a message can be sent, you should specify its destination, which can be specified as a URI string such as:

Creating a SOAP Message and Handling the Response

```
String destination = "http://...";
  or
URL destination = new URL("http://...");
```

Creating a SOAP Message and Handling the Response

- Now, the message is ready to be sent and this is done using the call() method, which blocks until it receives the returned response represented in SOAPMessage.

```
SOAPMessage response = sc.call(msg, destination);
```

- The returned response, response, is a SOAPMessage object and therefore has the same format as the one sent.

Handling Faults

- Things do not always work according to plan.
- For example, a client may fail to authenticate with an application (consider a developer that doesn't have a Google key to interact with Google services).
- SOAP defines a mechanism for error handling that is capable of identifying the source and cause of the error.
- In addition, it allows for error-diagnostic information to be exchanged by the parties involved in the interaction.

Handling Faults

- This is accomplished through the notion of a SOAP *fault*.
- As an example, consider the following segment of a response message containing a fault caused by the authentication failure:

Handling Faults

```
... <SOAP-ENV:Body>
<SOAP-ENV:Fault>
  <faultcode>Client.AuthenticationFailure
  </faultcode>
<faultstring>Failed to authenticate
  client</faultstring>
<faultactor>urn:X-
  SomeService:SomeGatewayProblem</faculta
  ctor>
</SOAP-ENV:Fault> </SOAP-ENV:Body> ...
```

Handling Faults

- The body of the response contains a single Fault element.
- This informs the client that an error has occurred, as well as some diagnostic information.
- The faultcode, which must always be present, provides information that is helpful in identifying the error -- this is not for human consumption however.

Handling Faults

- The faultstring is the human-readable string representing the error message.
- Finally, the faultactor specifies where in the message path the error has occurred.
- Here is an example of how to retrieve this information from a fault element using SAAJ:

Handling Faults

```
// Using SAAJ to work with SOAP faults
   if(responseBody.hasFault())
{ SOAPFault fault = responseBody.getFault();
Name code = fault.getFaultCodeAsName();
String string = fault.getFaultString();
String actor = fault.getFaultActor();
System.out.println("Fault contains: ");
System.out.println("Fault code:
   "+code.getQualifiedName());
System.out.println("Fault string: "+string);
if(actor != null)
{ System.out.println("Actor: "+actor); } }
```

Sample Application 1: (Using XMethods' Barnes & Noble Price Quote Service)

- This section provides a real-world sample application to demonstrate the power of SAAJ. The application is a SOAP client that can be used to interact with the XMethods' Barnes and Noble Price Quote service, which returns the price of a book at bn.com given its ISBN number. To get a feeling of how it works, try.
- To interact with this service, a SOAP request similar to the one shown in Code Sample 1 is used. Note that the ISBN would be different depending on the user's input.

request.xml

```
<SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-
  instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
<ns1:getPrice xmlns:ns1="urn:xmethods-
  BNPriceCheck" SOAP-
  ENV:encodingStyle="http://schemas.xmlsoap.org/s
  oap/encoding/">
<isbn xsi:type="xsd:string">0596002432</isbn>
  </ns1:getPrice>
</SOAP-ENV:Body> </SOAP-ENV:Envelope>
```

Sample Application 1: (Using XMethods' Barnes & Noble Price Quote Service)

- Using SAAJ, the body of the above SOAP request can be populated in a message as follows:

A message

```
Name bodyName = sf.createName("getPrice",
  "ns1", "urn:xmethods-BNPriceCheck");
SOAPBodyElement bodyElement =
  body.addBodyElement(bodyName);
Name name = sf.createName("isbn");
SOAPElement isbn =
  bodyElement.addChildElement(name);
isbn.addTextNode("0596002432");
```

Sample Application 1: (Using XMethods' Barnes & Noble Price Quote Service)

- To execute SoapClient.java:

```
C:\j2eetutorial14\examples\saaj\myapp>C:\apache-
   ant-1.6.1\bin\ant run
Buildfile: build.xml
init:
prepare:
build:
run:
     [echo] Running SoapClient.
     [java] SOAP Request Sent:
```

Sample Application 1: (Using XMethods' Barnes & Noble Price Quote Service)

- Once the message is populated and sent to its destination, the call() method blocks to receive a response.
- The response is received in a SOAPMessage that will have the same structure as the SOAP request, and therefore you can process it to retrieve the information needed (price).
- In this example, a transformer has been created to retrieve the content of the reply and display it as received. The response received is shown

response.xml

```
<SOAP-ENV:Envelope xmlns:SOAP-
   ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-
   instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
<ns1:getPriceResponse xmlns:ns1="urn:xmethods-
   BNPriceCheck" SOAP-
   ENV:encodingStyle="http://schemas.xmlsoap.org/s
   oap/encoding/">
<return xsi:type="xsd:float">34.95</return>
   </ns1:getPriceResponse>

</SOAP-ENV:Body> </SOAP-ENV:Envelope>
```

Sample Application 1: (Using XMethods' Barnes & Noble Price Quote Service)

- A SOAP message can be processed by retrieving the child elements and iterating over them as follows.
- In this example, there is only one child element.

Sample Application 1: (Using XMethods' Barnes & Noble Price Quote Service)

```
SOAPBody responseBody =
  reply.getSOAPBody();
Iterator iterator =
  responseBody.getChildElements();
while(iterator.hasNext())
{ bodyElement = (SOAPBodyElement)
  iterator.next(); String price =
  bodyElement.getValue();
System.out.println("The price for book
  with ISBN: .... is: "+ price); }
```

Reading the Message From a File

```
StreamSource msg = new StreamSource(new
  FileInputStream("c:/request.xml"));

  soapPart.setContent(msg);
```

Sun's JAX-RPC

- Sun's JAX-RPC is a Java API for XML-based Remote Procedure Calls (RPC) that can be used to easily develop Web services and web services clients. The advantage of JAX-RPC is that it hides the complexity of SOAP messages from the developer.
- Using JAX-RPC, the developer doesn't need to worry about constructing SOAP messages on his/her own but instead can concentrate on the application logic.