2 SOAP

Objectives

After reading this chapter, you should:

1. Possess a basic understanding of SOAP
2. Be able to describe the structure of a SOAP message
3. Understand how to process a SOAP message
4. Be able to map a SOAP to a transport protocol
5. Be able to write a simple SOAP client using SOAP with Attachment for Java (SAAJ)

SOAP was an improvement of XML-RPC (Remote Procedure Call) that made it possible to use XML to represent data between two systems. Initially, SOAP message structure was relatively simple. Since SOAP used HTTP protocol, it was tunneled through firewall using the Internet infrastructure. According to World Wide Web Consortium (W3C), SOAP is defined as follows (http://www.w3.org/ TR/2007/REC-soap12-part1-20070427/):

> *…a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.*

SOAP was implemented as a plug-in to many middleware platforms and enabled data exchange over the Internet in addition to intranet. The initial emphasis on RPC thus allowed SOAP to become widely implemented as a wire-protocol over the Internet. Eventually, the need for interactions other than RPC led to the Documentation type of exchange. Protocols other than HTTP have also emerged over the years (e.g., SMTP and JMS).

SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment (http://www.w3.org/TR/soap12-part1/#intro). SOAP specifications provide a formal set of conventions that governs how SOAP messages are generated and accepted by SOAP nodes (i.e., senders, receivers and intermediaries).

SOAP XML Payload

A SOAP message is enclosed in a SOAP envelope that contains a SOAP header and a SOAP body. The SOAP body is mandatory, whereas the SOAP header is optional. The SOAP message is the basic unit of communication between SOAP nodes. A SOAP node can transmit, receive or relay a SOAP message. The responsibility of the ultimate destination is to process SOAP messages according to the standard. A SOAP message traverses between a SOAP sender and a SOAP receiver. The message path may have SOAP intermediary nodes in a distributed computing environment. These SOAP intermediaries are thought of as 'men-in-the-middle', which can cause significant security problems.

The two major message exchange patterns that SOAP supports are 1) request-response (in-out) and 2) request (in-only). The first pattern is often used where a SOAP request is processed and a SOAP response is returned to the SOAP sender node. The second pattern is used when the SOAP sender has no interest in receiving a response (e.g., notification). A SOAP response may be returned in an asynchronous mode. For example, a server may take a long time to process a SOAP request. Instead of waiting for this request to be completed, a SOAP client may receive a callback when the server completes the processing.

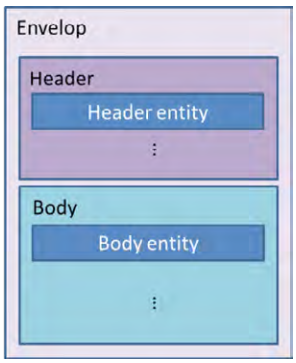

Figure 2-1. *SOAP Message Structure*

2.1 Examples of SOAP messages

The following shows SOAP messages in three forms: requests, responses and faults.

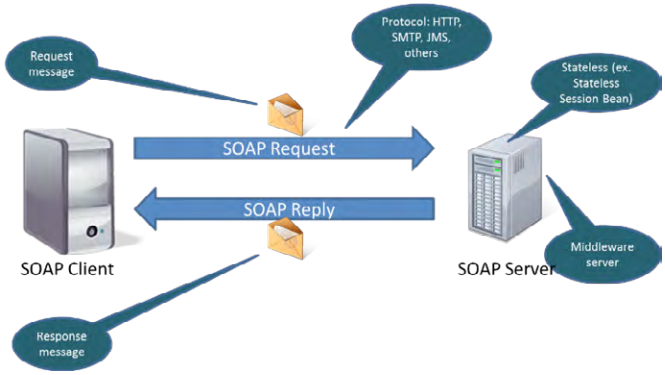

Figure 2-2 *SOAP message exchange*

Messages can be exchanged using HTTP protocol; thus, firewalls would allow the messages to pass through. SOAP messages can also ride naturally over a secure channel using HTTPS protocol as well. SOAP promotes a loose-coupling computing paradigm where the knowledge of software components is limited to interface of the service.

2.1.1 SOAP request message

The request represents an invocation of getEmployee method with an emplNo parameter of 100011. Note that the body contains a single body block with one XML element, <emp:getEmployee>. A request is marshalled (serialized) into an XML document prior to transport across the network to a remote SOAP server. When the message arrives at the server, it is 'un-marshalled' by a SOAP engine and the targeted method is invoked.

- A SOAP header is not used in this case.
- The SOAP body consists of a single element, emp:getEmployee.
- getEmployee: To get an employee record, an employee number (emplNo) is required.

Listing 2-1. A SOAP Message Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:emp="http://employees.ws.bemach.com/">
   <soapenv:Header />
   <soapenv:Body>
      <emp:getEmployee>
         <emplNo>100011</emplNo>
      </emp:getEmployee>
   </soapenv:Body>
</soapenv:Envelope>
```

2.1.2 SOAP response message

When the SOAP server successfully processes a request, a response is then marshaled into an XML document and returned as a part of HTTP response to the client. The client SOAP engine then unmarshals the message to further process the result of the call.

- A SOAP header is not used.
- The SOAP body has a single element, ns2:getEmployeeResponse. Note that namespace, ns2, is used and referenced.
- Within the SOAP body, an employee record is returned.

Listing 2-2. A SOAP Message Response

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getEmployeeResponse xmlns:ns2="http://employees.ws.bemach.com/"
         xmlns:ns3="http://bemach.com">
         <return>
            <emplNo>100011</emplNo>
            <firstName>Shmuel</firstName>
            <lastName>Birge</lastName>
            <birthDate>1956-07-20T00:00:00-04:00</birthDate>
            <gender>M</gender>
            <hireDate>1989-11-23T00:00:00-05:00</hireDate>
         </return>
      </ns2:getEmployeeResponse>
   </S:Body>
</S:Envelope>
```

2.1.3 SOAP fault message

If a request contains an invalid employee number, then a SOAP fault message can be optionally returned. A SOAP fault may contain messages that help the client to resolve an error or unexpected condition. It must include a fault code, fault string, and a detailed error message.

Listing 2-3. A SOAP Fault

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>No such employee!</faultstring>
         <detail>
            <ns2:SOAPException xmlns:ns2="http://employees.ws.bemach.com/"
               xmlns:ns3="http://bemach.com">
               <message>No such employee!</message>
            </ns2:SOAPException>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

The basic assumption is that all data types are exchanged between SOAP nodes in XML format. In reality, not all data should be translated into XML (e.g., binary image, proprietary data formats). Thus, a proposed solution is to use SOAP attachments similar to the email protocol. There are problems with this approach, though, in that scalability can become an issue. Furthermore, not all SOAP engine implements SOAP attachment; thus, compatibility becomes an issue between SOAP nodes. This does not discount the overhead of an attachment.

2.2 Mapping SOAP to HTTP

The SOAP message exchange maps nicely into HTTP protocol. In this example, an HTTP POST is used for invocating a SOAP method on a server:

Listing 2-4. An HTTP Post

```
POST /java-ws/hello HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
Content-Length: 298
Host: localhost:9999
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:hel="http://hello.ws.bemach.com/">
   <soapenv:Header/>
   <soapenv:Body>
      <hel:say>
         <!-Optional:->
         <name>Johnny B. Good</name>
      </hel:say>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 2-5. Another HTTP Post

```
POST /rpc/employees HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
Content-Length: 290
Host: localhost:9999
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:emp="http://employees.rpc.ws.bemach.com/">
   <soapenv:Header/>
   <soapenv:Body>
      <emp:getEmployee>
         <emplNo>10002</emplNo>
      </emp:getEmployee>
   </soapenv:Body>
</soapenv:Envelope>
```

In this case, a return of an HTTP POST for a SOAP response is relatively simple:

Listing 2-6. An HTTP Response

```
HTTP/1.1 200 OK
Transfer-encoding: chunked
Content-type: text/xml;charset="utf-8"

5e
<?xml version="1.0" ?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body>

88
<ns2:sayResponse xmlns:ns2="http://hello.ws.bemach.com/"><return>Hello,
Johnny B. Good!</return></ns2:sayResponse></S:Body></S:Envelope>

0
```

Listing 2-7. Another HTTP Response

```
HTTP/1.1 200 OK
Transfer-encoding: chunked
Content-type: text/xml; charset=utf-8
Date: Thu, 11 Apr 2013 10:20:49 GMT

5e
<?xml version="1.0" ?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body>

167
<ns3:getEmployeeResponse xmlns:ns2="http://bemach.com"
xmlns:ns3="http://employees.rpc.ws.bemach.com/"><return><emplNo>10002</
emplNo><firstName>Bezalel</firstName><lastName>Simmel</
lastName><birthDate>1964-06-02T00:00:00-04:00</birthDate><gender>F</
gender><hireDate>1985-11-21T00:00:00-05:00</hireDate></return></
ns3:getEmployeeResponse></S:Body></S:Envelope>

0
```

2.3 SAAJ Client

A SOAP with Attachment API for Java (SAAJ) is more complex to write. It allows the developers to gain direct access to methods of creating SOAP messages. You can manipulate XML directly using Java APIs. Unlike the example in Chapter 1 of writing the client code that uses JAX-RPC, writing an SAAJ client can be laborious. SAAJ allows a small footprint and support for binary data; however, it is difficult to work with in various binary data formats.

The steps for creating a SOAP client using SAAJ are as follows:

1. Create a SOAP message that includes a SOAP header and SOAP body.
2. Add necessary elements to the SOAP header and SOAP body.
3. Create a SOAP connection (URL).
4. Send the SOAP message to the SOAP server.
5. Process the response message.

A SOAP client can be written in Java with a basic understanding of SOAP messages. A client code of the HelloWorld service is as follows.

Listing 2-8. HelloWorldSOAPClient.java class Using SAAJ

```
package com.bemach.ws.hello.client;

/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.net.MalformedURLException;
import java.net.URL;
import java.util.Iterator;
import java.util.logging.Logger;

import javax.xml.soap.MessageFactory;
import javax.xml.soap.Name;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPFactory;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;

/**
 * The following code is a complex way of go about calling a
 * Web services using Java code.
 *
 * There are times this is necessary.
 *
 */
public final class HelloWorldSOAPClient {
   private static final Logger LOG = Logger.getLogger(HelloWorldSOAPClient.
class.getName());
   /**
    *

      <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:m="http://hello.ws.bemach.com/">
         <soapenv:Header/>
         <soapenv:Body>
            <m:say>
               <arg0>Johnny</arg0>
            </m:say>
         </soapenv:Body>
      </soapenv:Envelope>
```

```
      <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
         <S:Body>
            <ns2:sayResponse xmlns:ns2="http://hello.ws.bemach.com/">
               <return>Hello, Johnny!</return>
            </ns2:sayResponse>
         </S:Body>
      </S:Envelope>

    */

   private HelloWorldSOAPClient() {}

   public static void main(String[] args) {
      try {
         // 1. Create a SOAP Message
         SOAPFactory sf = SOAPFactory.newInstance();
         MessageFactory mf = MessageFactory.newInstance();
         SOAPMessage sm = mf.createMessage();
         SOAPHeader sh = sm.getSOAPHeader();
         SOAPBody sb = sm.getSOAPBody();
         sh.detachNode();

         // 2. Add necessary elements to header and body.
         Name bodyName = sf.createName("say","m","http://hello.ws.bemach.com/”);
         SOAPBodyElement be = sb.addBodyElement(bodyName);
         Name name = sf.createName("name");
         SOAPElement arg0 = be.addChildElement(name);
         arg0.addTextNode("Johnny");

         // 3. Create a SOAP connection
         URL ep = new URL(String.format("http://localhost:%s/java-ws/
hello?WSDL",args[0]));
         SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
         SOAPConnection sc = scf.createConnection();

         // 4. Send a SOAP message using the connection.
         SOAPMessage response = sc.call(sm, ep);
         sc.close();

         // 5. Process the response message
         SOAPBody respBody = response.getSOAPBody();
         Iterator it = respBody.getChildElements();
         SOAPBodyElement el = (SOAPBodyElement)it.next();
         LOG.info("resp="+el.getNodeName());

         it = el.getChildElements();
         SOAPElement ret = (SOAPElement) it.next();
         LOG.info(String.format("%s=%s",ret.getNodeName(),ret.getTextContent()));
      } catch (UnsupportedOperationException e) {
         LOG.severe(e.getMessage());
      } catch (SOAPException e) {
         LOG.severe(e.getMessage());
      } catch (MalformedURLException e) {
         LOG.severe(e.getMessage());
      }
   }
}
```

SAAJ has many more capabilities that you can use to work with SOAP messages. For more information on this topic, see the references at the end of this chapter.

SOAP messages can be structured in two ways: document and RPC styles. At first, only RPC style was supported. For the RPC model, the SOAP body defines a specific method with associated parameters that the client can invoke. Thus, the message exchange between client and server can be restricted. Method calls are tied directly with the in and out parameters.

Unlike the RPC style, however, the document style enables the client and server to exchange messages in whatever formats they choose. The SOAP body contains messages that do not follow any SOAP formatting rules. The body can be validated against a schema.

2.4 Summary

SOAP is the foundation of Web Services. SOAP remains a dominant message exchange protocol used for B2B integration. Note that no security concern is deliberately mentioned in SOAP. SOAP is mainly about exchanging messages between two systems. It promotes loose-coupling computation. Security and other capabilities are left out to avoid complexity in the protocol.

In the next chapter, we will discuss how Web Services use SOAP as a transport protocol to promote the service-oriented paradigm.

2.5 References

Additional information can be found in the following documents, which are available online:

Albrecht, C.C. (2004). "How clean is the future of SOAP?" Commun. ACM **47**(2): 66–68.

SOAP Version 1.2 Part 0: Primer (Second Edition), Retrieved August 27, 2007 from: http:// www.w3.org/TR/2007/REC-soap12-part0-20070427/

Bong, G. (2002). Apache SOAP type mapping, Part 1: Exploring Apache's serialization APIs, Retrieved August 27, 2007 from: http://www-106.ibm.com/developerworks/webservices/ library/ws-soapmap1/

Bong, G. (2002). Apache SOAP type mapping, Part 2: A serialization cookbook, Retrieved August 27, 2007 from: http://www-106.ibm.com/developerworks/webservices/library/ws-soapmap2/

Cohen, F. (2001). Myths and misunderstandings surrounding SOAP, Retrieved August 27, 2007 from: http://www-106.ibm.com/developerworks/library/ws-spmyths.html