SOAP Web services and XML Binding

CS 571
Web services

- **Web service**: "a software system designed to support interoperable machine-to-machine interaction over a network“ – W3C
- In short, web services are entities on a network (which including the internet) that provide functions for other entities to call remotely.
How does SOAP web services work?

- Web services use XML as a transport to get requests to the service from the client and responses to the client from the service.
- This message is wrapped in a XML envelope called SOAP.
What is SOAP

- A SOAP envelope consists of a header and body
- The body is mainly used to carry the payload, the actual data to be sent (whether it be a request or response)
- The header is mainly used to carry metadata and other relevant information about the message, like security, MIME type (binary or text) and other information about the data.
SOAP Connection

- A SOAP connection is required to make the actual transaction.
- The connection is generated by a connection factory, provided by the Java environment.
Implementing SOAP (Pt. 1)

- To implement a SOAP connection, you must first have all the proper connections. To do that you need to download and install all library files, found in the “lib” directory from the root of the package, from an Axis installation (http://ws.apache.org/axis/)

- Since SOAP is a specification, we need an implementation... and that’s what Axis is.
You first need a message factory to generate a blank SOAP message for you:

```java
MessageFactory messageFactory = MessageFactory.newInstance();
SOAPMessage message = messageFactory.createMessage();
```
Implementing SOAP (Pt. 3)

- You then need to extract the different parts of a SOAP envelope:

```java
SOAPPart soapPart = message.getSOAPPart();

SOAPEnvelope envelope = soapPart.getEnvelope();

// extracting the body of the envelope
SOAPBody body = envelope.getBody();

// extracting the head of the envelope
SOAPHeader head = envelope.getHeader();
```
Implementing SOAP (Pt. 4)

- **SOAP Body**
  - Anything can go in here, as long as it is a DOM Document, or a Document Object Model representation of an XML document.
  - You can add whatever using:
    - `body.addDocument(<document>);`

- **SOAP Head**
  - This can have many attributes added to them like MIME types, security, etc. as mentioned earlier
  - Read up on its documentation for more information
Implementing SOAP (Pt. 5)

- To connect to the server to make a SOAP call, you first need a connection factory:
  - `SOAPConnectionFactory soapConnFactory = SOAPConnectionFactory.newInstance();`

- You can then obtain a connection from that factory:
  - `SOAPConnection connection = soapConnFactory.createConnection();`
Implementing SOAP (Pt. 6)

• After you have setup the connection, you can make the actual call and get the response back, a SOAPMessage:
  • SOAPMessage reply = connection.call(message, <URL>);

• You can retrieve the XML payload in that response by getting to the message body:
  • SOAPBody response = reply.getSOAPBody();

• What you do with this response is up to you.

• Oh, and don’t forget to close the connection!
  (connection.close();)
More SOAP tutorials

- Here’s a nice little tutorial for how to make a SOAP connection (don’t worry about the XML part for now):
  - [http://www.informit.com/guides/content.aspx?g=xml&seqNum=130](http://www.informit.com/guides/content.aspx?g=xml&seqNum=130) (read up until the part about making the connection, stop at `connection.close()`.)
  - **WARNING**: You will need to install all the files in the “lib” folder of an Axis ([http://ws.apache.org/axis/](http://ws.apache.org/axis/)) install into your classpath for you to be able to run this code. You still get a warning but it doesn’t matter unless you’re attaching files to the body of a SOAP message.
Web service definitions

- So how do you know how to make a web service call? You can find out how to use a web service by looking a Web Service Description Language (WSDL) document.

- WSDL is really a form of XML. It has special tags for the purpose of defining a web service.

- A WSDL document describes the ports that are open in a Web Service (WS), the functions that it offers, and the data types that you need to use when making the calls as well as how to decrypt data types sent back.
Another useful utility that is out there is the Universal Description, Discovery and Integration (UDDI)

The UDDI works just like a DNS server, it lists a bunch of web services that you can use, their address and their published WSDL(s)
WSDL data types

- The WSDL document that each web service publishes is pretty complete with information about the web service... but for the purpose of this class, all you guys need to do is look at the type definition of the WSDL.
- The type definitions are usually defined as complex types and simple types in the WSDL.
- Things that you also want to pay attention to are the “message” tags. They define the messages that you can write to the functions within the web service. Reading this usually will abstract you from reading up on the actual interface of the web service.
Once type definitions in a WSDL are determined we can try to “bind it” to a client.

Binding the types to an application is the process of creating classes that restrict your Java (or other language) code into following the XML schema so that no mistakes are made while creating them.

This service is done for you by a variety of binding tools, you don’t always have to hack it yourself.
XML Beans

- XML Beans is an XML binding utility that we will use in this class.
- XML Beans compiles Java classes from the data types in any XML schema, including WSDL documents.
- Each type has a Document object that is the “root” of the XML document. It has a Factory inner-class associated with each Java class it generates that will create a blank instance for each type (without worry of screwing it up).
- Each Document object has an actual object in there that is the actual data.
- XML Beans objects all inherit from XmlObject, which can all be converted into DOM Nodes (needed by SOAP for the body of a SOAP document).
Using XML Beans Tutorials

- Compiling a JAR file from a schema is made easy with this tutorial from the official site (check out the example):
  - [http://xmlbeans.apache.org/docs/2.0.0/guide/antXmlbean.html](http://xmlbeans.apache.org/docs/2.0.0/guide/antXmlbean.html)

- Here’s an official getting started (XML to Java code) tutorial for XML Beans (pay attention to the naming of objects in the XML and how it maps back to code):
Future of web service

- The current day model and most familiar web services used are SOAP ... but many have and are still transitioning to Representational State Transfer (REST) web services.
- REST is drastically different from SOAP in that instead of using an XML envelope to convey the message, plain HTTP messages are used to convey the messages... using simple GET, POST, PUT and DELETE requests to deliver the messages.
REST (cont.)

- REST is relatively new, conceived in Roy Fielding’s PhD dissertation in 2000 with only the first commercial book on it in 2007 (http://oreilly.com/catalog/9780596529260/).
- Most people are still confused about REST and use SOAP messages to weigh it down, defeating the purpose of REST.
- Much more needs to be done in the area before it hits true maturity.