SOAP LIBRARIES

Introduction:
The following dicussion aims to outline the working of soap libraries the main focus of attention will be Apache's versions of SOAP versin 2 and axis. We will also look at some other SOAP engines available and outline the differences between them. The piece will be accompanied by other copmlimentary pieces such as SAX, jax-rpc which will help to clarify certain aspects of the disscussion.

Apache Axis.

Overview.
Axis is essentially SOAP 3.0 which uses the Simple API for XML SAX (see SAX) rather than DOM. Axis can be described as a framework for constructing SOAP processors such as clients and servers etc.

source:http://ws.apache.org/axis/

The basic Arcitecture of AXIS:
When the central Axis processing logic runs, a series of Handlers are each invoked in order. Order depending on deployment configuration, and whether the engine is a client or a server.

The object which is passed to each handler invocation is a MessageContext.

A messageContext is a structure which contains
a "request message"
a "reponse message"
a bag of properties.

There are two ways axis is invoked.

As a server, a transport listener will create a MessageContext and invoke the Axis processing framework.

As a client, application code usually aided by the client programming model of axis will generate a Message Context and invoke the axis processing framework.

In either case the axis framework's job is simply to pass the resulting MessageContext through the configured set of handlers.

1. The Message Path on the server side

A message arrives at a Transport listener e.g HTTP servlet
The listener packs the protocol-specific data into a message. org.apache.axis.Message
The MessageContext is loaded with various properties by the listener.
The transport Listener also sets the transportName String on the MessageContext e.g to http.
Once ready the Transport listener hands it over to the AxisEngine.

-
-

The Axis Engine

Firstly look up the transport by name.
The transport is an object which contains a request chain and or a reponse chain.
A chain is a handler consisting of a sequence of handlers which are invoked in turn.

If a transport request chain exists.
It will be invoked, passing the MessageContext into the invoke() method.
This will result in calling all the handlers specified in the request Chain configuration.

After the transport request Handler, the engine locates a global request chain, if configured, it then invokes any handlers specified therin.

At some point during the message processing the serviceHandler field of the messageContext will have been set by some handler.

This determines the Handler to invoke to execute service-specific functionality.
Service in Axis are typically instances of " SOAPService" class which may contain request and response Chains but lust contain a provider.
A provider is simply a Handler responsible for implementing the actual back end logic of the service.

RPC-style requests. (see jax-rpc)

The provider is the org.apache.axis.providers.java.RPCProvider class.
This is another Handler, when invoked, attempts to call a backend Java object whose class is determined by the " className" parameter specified at deployment time.
It uses the SOAP RPC convention for determining the method call, and makes sure the types of the incoming XML-encoded arguments match the types of the required parameters of the resulting method.

1.1. The Message Path on the Client side.

The serviceHandler is called there is no provider since the service is being provided by a remote node.
The service request and response chains if any perform any service-specific processing of the request message on it's way out of the system and any reponse messages on it's way back to the caller.

After the service request chain, the global request chain, if any is invoked.
Followed by the transport.
The transport sender, a special Handler to perform whatever protocol-specific operations are necessary to get the message to and from the target SOAP server is invoked to send the message.

The response is placed into the responseMessage field of the messageContext, and the MessageContext then propagates through the response chains first the transport then the global and finally the service.

1.2 Other details in brief.

Handlers and Chains.

Handlers are invoked in a sequence to process messages.
At some point in the sequence a Handler may send a request and receive a response or else process a request and produce a response.
Such a handler is called is known as a pivot point of the sequence.
Handlers are either transport, service specific or global.
The handlers of each of these three different kinds together are combined into chains.

A web service does not necessarily send a response to each request.
Response handlers may alternativly be used as stop timers and clean up resources.

Message Contexts

Each message context may be associated with a request message and or a response message.
Each message has a soap part and an attachements object, both of which implement the part interface.

The typing of Message Contexts needs to be carefully considered in relation to the Axis Architecture. A message Context appears on the Handler interface, it should not be tied or biassed in favour of SOAP.

Class SOAPMessage

public abstract class SOAPMessage extends java.lang.Object

The root class for all SOAP messages. As transmitted on the " wire" , a SOAP message is an XML document whose first body part is an XML/SOAP document.

A SOAPMessage object consists of a SOAP part and optionally one or more attachement parts. The SOAP part for a SOAPMessage object is a SOAPPart object, which contains information used for message routing and identifivation, and which can contain application-specific content. All data in the SOAP Part of a message must be in XML format.

A new SOAPMessage object contains the following by default
- A SOAPPart object
- A SOAPEnvelope object
- A SOAPBody object
- A SOAPHeader object

The SOAP part of a message can be retrieved by calling the method SOAPMessage.getSOAPPart()
The SOAPEnvelope object is retrieved from the SOAPPart object.
The SOAPEnvelope object is used to retrieve the SOAPBody and Soap Header objects.

```
SOAPPart sp = message.getSOAPPart();
SOAPEnvelope se = sp.getEnvelope();
SOAPBody sb = se.getBody();
SOAPHeader sh = se.getHeader();
```

In addition to the mandatory SOAPPart object, a SOAPMessage object may contain zero or more AttachementPart objects, each of which contains application-specific data.. The SOAPMessage interface provides methods for creating AttachementPart objects.

Unlike the rest of a SOAP message, an attachement is not required to be in XML format and can therefore be anything from simple text to an image file. Consequently, any message content that is not in XML format must be in an AttachementPart object.

A MessageFactory object creates new SOAPMessage objects. If the MessageFactory object was initialised with a messaging Profile, it produces SOAPMessage objects that conform to that profile. For example, a SOAPMessage object created by a MessageFactory object initialised xith the ebXML profile will have the appropriate ebXML headers.

Constructor

public SOAPMessage()

Methods

abstract void addAttachementPart (AttachementPart attachmentpart)
Adds the given AttachmentPart object to this SOAPMessage object.

abstract int countAttachments()
Gets a count of the number of attachments in this message

public AttachementPart createAttachmentPart(java.lang.object content, java.lang.String contentType)

Creates a new empty AttachmentPart object.

Other Soap LIbraries

SOAP::Lite for Perl

SOAP::Lite for Perl is a collection of Perl modules which provides a simple and lightweight interface to the Simple Object Acess Protocol both on Client and server side.

This verson of SOAP::Lite supports the SOAP1.1 specification

Overview of SOAP::Lite's Classes and Packages.

SOAP::Lite.pm	
SOAP::Lite	Main class provides all logic
SOAP::Transport	Supports transport architecture
SOAP::Data	Provides extensions for serialisation architecture
SOAP::Header	Provides extensions for header serialisation
SOAP::Parser	Parses XML file into object tree
SOAP::Serialiser	Serialises data structures to SOAP package
SOAP::Deserialiser	Deserialises results of SOAP::Parser into objects
SOAP:: SOM	Provides access to deserialised object tree
SOAP::Constants	Provides access to common constants
SOAP::Trace	Provides tracing facilities
SOAP::Schema	Provides access and stub(s) for schema(s)
SOAP::Schema::WSDL	WSDL implementation for SOAP::Schema
SOAP::Server	Handles requests on server side
SOAP::Server::Object	Handles objects-by-reference
SOAP::Fault	Provides support for Faults on server side

2. NuSOAP:PHP SOAP Library

Not much information exists about it except it's size is amost 140 KB long which is a bit too big to be of any real interest to us.
source http://dietrich.ganx4.com/nusoap/APIDoc/

Classes

XMLSchema
nusoap_base
soap_fault
soap_parser
soap_server
soap_transport_http
soapclient
soapmsg
soapval
wsdl

3. Jamsterdam SPAX

Note:
Because of the apparent similarities between SPAX and SAX I initially concentrated on understanding SAX a basic summay of which is available on the CDS twiki site.

Advantages of SPAX.

SPAX is an interface and Java library for hand rolled SOAP message parsing in a way that is very similar to SAX. With SPAX, the processing of the outer SOAP elements is peformed automatically, allowing you to concentrate only on the parsing of the SOAP:Body message payload elements. Instead of implementing SAX callback methods such as " startElement" and " characters" , you implement such SOAP-specific methods as " startEnvelope" ," startHeaderElement" , and " bodyCharacters" , allowing quick, lightweight, and fine-grained handling of the message i.e. it provides a shortcut for parsing ony the contents of the SOAP Header and Body.

so briefly SAX works as follows

1 .
implement org.xml.ContentHandler or subclass.org.xml.sax.helpers.DefaultHandler to create a custom handler for the entire XML document.

This would override one or more of these methods:
startDocument()
endDocument()
startElement(**String uri, String local, String raw, Attributes attr**)
endElement(**String uri, String local, String raw**)
characters(**char ch[], int begin, int len**)

Each of which is declared to throw a org.xml.sax.SAXException.

2.
Create an instance for this custom handler in your application.

3.
Create an instance of org.xml.sax.XMLReader in your application.

4 .
Pass the custom handler to the setContentHandler() method of the XMLReader instance.

5.
Pass the XML document, properly encapsulated in an org.xml.sax.InputSource instance, to the parse() method of the XMLReader.

6.
Take the appropriate actions from results of the parsing by the custom handler.

Spax parsing of soap in contrast.

1 .
Implement org.jamsterdam.spax.SoapHandler or subclass org.jamsterdam.spax.DefaultSoapHandler to create a custom handler for the interior of the SOAP message only

Typically would override one or more of these methods:

startEnvelope()
endEnvelope()
startHeader()
endHeader()
startBody()
startHeaderElement**(String uri, String local, String raw, Attributes attr)**
endHeaderElement(**String uri, String local, String raw**)
startBodyElement(**String uri, String local, String raw, Attributes attr**)
endBodyElement(**String uri, String local, String raw**)
headerCharacter(**char ch [], int begin, int local**)
bodyCharacters(**char ch [], int begin, int local**)

2.
create an instance of your custom SOAP handler in your application

3.
Create an instance of org.jamsterdam.spax.SoapReader in your application

4 .
Pass the SOAP message, property encapsulated in an org.xml.sax.InputSource instance, to the parse() method of the SoapReader.

5.
Take the appropriate actions and send appropriate reponse messages from the results of the parsing by the custom handler.

How does SPAX compare to Axis.

Apache Axis runs as a Java web application, through which you can deploy your existing class. Apache Axis is ideal if you simply want to expose your java classes to remote procedure calls using SOAP. If you want more complex serialisation in th emost efficient manner. Moreover, with SPAX, you can deploy your handlers in your own Java web application, and in your own servlets.

To generate SOAP messages for example using a StringBuffer. There are convenience methods in the org.jamsterdam.xml.Soap class of thr Jamsterdam SPAX download to wrap XML inside an appropriate SOAP shell.

Classes found in the Jamsterdam SPAX Library

Class
org.jamsterdam.spax.SpaxParser

Implements org.xml.saxContentHandler. This is the content handler that provides the filtering of the SOAP Message, in order to make sure the message is a proper SOAP Message.

In most cases, you do not need to extend this class. Instead in your code, you create an instance of this class and pass it to the setContentHandler method of an XMLReader instance.

To provide the actual parsing of the SOAP content, you create a SoapHandler implementation and pass it to the setContentHandler() method of the SpaxParser instance.

Class
org.jamsterdam.spax.SoapHandler

Interface that provides the methods for parsing the interior of the SOAP message. THe work of parsing your SOAP message consists of implementing this interface in a custom class. It is analogus to the org.xml.sax.ContentHandler interface for SAX parsing.

Or easier you can extendDefaultSoapHandler

Class
org.jamsterdam.spax.DefaultSoapHandler

Trivial implementation of SoapHandler with callback methods that simply return. Tha easiest wat to parse SOAP messages is probably to extend this class, overriding tha various methods that you need for parsing. In this way, is analogous to org.xml.sax.DefaultHandler for SAX parsing.

Class
org.jamsterdam.spax.SoapHandlerException
extends org.xml.sax.SAXException.

This is a package-specific exception that is declared to be thrown by all methods of the SoapHandler interface. You can throw exceptions of this class when you encounter application-level errors in parsing the SOAP message. The exceptions will be thrown by the SpaxParser class, and can be caught by your application in order to return the appropriate SOAP fault messages.

org.jamsterdam.xml.Soap
Utility class providing low-level SOAP handling methods used by SPAXParser. can also be used in generating the response messages.

The SPAX interface currently comprises the following Java Methods.

```
public void startEnvelope()   throws SoapHandlerException
public void endEnvelope()   throws SoapHandlerException;

public void startHeader()   throws SoapHandlerException
public void endHeader()   throws SoapHandlerException;

public void startBody()   throws SoapHandlerException
public void endBody()   throws SoapHandlerException;

public void startHeaderElement(String uri, String local, String raw, org.xml.sax.Attributes attr)
                                    throws SoapHandlerException;


public void endHeaderElement(String uri, String local, String raw)
                                    throws SoapHandlerException;

public void startBodyElement(String uri, String local, String raw, org.xml.sax.Attributes attr)
                                    throws SoapHandlerException;

public void endBodyElement(String uri, String local, String raw)
                                    throws SoapHandlerException;

public void bodyCharacters(char ch[], int begin, int len)
                                    throws SoapHandlerException;
```

Links to more general information.

SOAP - Library for SOAP clients and servers in Perl

http://search.cpan.org/author/KBROWN/SOAP-0.28/lib/SOAP.pm#SYNOPSIS

http://perlhelp.web.cern.ch/PerlHelp/site/lib/SOAP.html

SOAP Library Profile.

http://www.pythonware.com/products/soap/profile.htm

Simple Object Access Ptotocol (SOAP) 1 .1
http://www.w3.org/TR/SOAP/

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoapspec/html/soapspecindex.asp

EasySoap
http://sourceforge.net/projects/easysoap/
http://easysoap.sourceforge.net/