Web Service Programming Using Axis

Christian Gross

O'Reilly Open Source Convention
July 7 - 11, 2003

Introduction: Christian Gross

- Trainer, Consultant, Mentor
  - Open / Shared Source / Binary technologies
  - Technologies (Web Services, Apache / XML, Java, C++, .NET)
- Training
  - http://www.devspace.com
- Author: Upcoming Book
  “Software Engineering Using Java Jakarta”
- Contact: christianhgross@yahoo.ca
Theory of SOAP

- Initially used for RPC (Remote Procedure Call)
- Has undergone quite a few revisions in a very short period of time (0.9, 1.0, 1.1, 1.2)
- Part of the XML Protocol working group
- Based on the concept of an envelope encapsulating a document
  - Like snail mail, envelope and letter within
So Far So Good?

- Web Services is only the beginning
- Some make it more complicated than it should be
  - Problem areas is infrastructure
  - DISCO, UDDI, ROPE, WSDL (some are already dead)
- Complexity depends on the context
  - Talking among friends no protocol needed
  - Talking in a court protocol is needed

What is SOAP Solving?

- The problem of Server to Server communications
- The problem of a “dumb” web where meta information is lost
- The problem of having ten zillion different data formats
SOAP Theory: Message

- Make an HTTP Request (SOAP in payload)
  - HTTP is an example of a SOAP Binding
  - Binding and Message are now two separate aspects of the SOAP specification

POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml
Content-Length: nnnn
SOAPMethodName: Some-Namespace-URI#GetLastTradePrice

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:GetLastTradePrice xmlns:m="Some-Namespace">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP:Body>
</SOAP:Envelope>

What is Axis?

- Axis is a distributed computing framework much like DCOM, IIOP, CORBA, RMI
- Axis is based on SOAP 1.1 (Simple Object Access Protocol)
  - SOAP is a XML based technology (more later)
  - Is SOAP 1.2 ready and able, but a few little details need sorting
    - No problem from architecture point of view
- Axis allows RPC and messaging calls
  - RPC (Remote Procedure Call): Conversational style communication
  - Messaging: Mail type communication with an envelope and contents
Axis Architecture

- Axis Server itself is a Servlet implementation
  - Can run in any Servlet Container (our preferred is Tomcat)
- Based on the concept of a chained message request / response
  - Very similar to Tomcat pipeline
- MessageContext is a wrapper object for the following objects…
  - Contextual information about process, request, response, etc and other grab bag of information
  - Request message
  - Response message

Axis Architecture (cont.)

- Request and Response are specific handlers that manipulate the MessageContext
- SOAP Transport is responsible for creating, sending and receiving a SOAP message on a specific transport (http, etc)
- SOAP Service is responsible for creating, sending and receiving MessageContext to a Java class
Pipeline Concepts

- Core concept of the pipeline includes three abstractions and a pivot point.
- Pipeline starts with a request and ends with a response:
  - Pivot point defines where the request turns into a response.
- The three abstractions of the Pipeline are:
  - Transport: Is it http or another protocol?
  - Global: A multi-length pipeline that allows multiple processing cycles for each SOAP request.
  - Service: Final pivot processor that processes the Web Service request.
- Client and Server architecture have identical pipeline process:
  - Pivot points are opposite.
  - Client pivot point is SOAP transport request (XML file).
  - Server pivot point is SOAP service request (Java method call).
**Axis Practical Implementation Details**

- **AxisServlet** is the core Servlet that manages everything with respect to SOAP requests
  - Capable of processing formally deployed and Auto-Deployed Web Services
- **AxisServlet** gets all of its instructions from the default Axis supplied web.xml file
  - .jws files are associated as Auto-Deploy Web Services
  - The URL `/axis/services/*` (where `*` is the name of a Web Service) is used to call formally deployed Web Services

---

**Axis Practical Implementation Details**

- The Axis group likes to use packages and hence starting programs requires specification of the packages
  - eg `org.apache.client.AdminClient` is used to manage the Axis server
- In most cases it is not necessary to modify the pipeline architecture
  - Adding of request and response handlers is accomplished within the `wsdd` Web Service configuration file

When using with Tomcat need to put axis jars into `$CATALINA_HOME/common/libs`
Web Service Auto Deployment Request

Web Service Auto Deployment Request (cont.)

- In the simplest case a web service is exposed using a "bean" file named *.jws
- Axis will recognize this as an auto-compile, auto-deploy file
- From previous slide steps are as follows:
  - .jws file is loaded and compiled to a .class file
  - .class file is loaded and executed by RPCProvider
  - Output from .class file is piped back as a response

Complex functionality is more complicated to maintain in this scenario
- Formal deployment is better
Web Service Formal Deployment

- Formal deployment means "installing" a Web Service and optionally defining a request chain or response chain.
- Web Services are defined using an XML instruction file.
  - Web Service is installed using a "deployment file" (deploy.wsdd).
  - Web Service is uninstalled using "undeployment file" (undeploy.wsdd).
- *.wsdd files are sent to the Axis framework and processed as an instruction set of Web Services actions.
  - Execute using command:
    `java org.apache.axis.client.AdminClient <filename>`
  - To view installed Web Services use command:
    `java org.apache.axis.client.AdminClient list`

Example Deploy.wsdd

```xml
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="MathService" provider="java:RPC">
    <parameter name="className" value="ApacheJavaCourse.MathServiceSoapBindingImpl"/>
    <operation name="doSubtract">
      <parameter name="inp1" type="tns:int"
        xmlns:tns="http://www.w3.org/2001/XMLSchema"
        mode="IN"/>
      <parameter name="inp2" type="tns:int"
        xmlns:tns="http://www.w3.org/2001/XMLSchema"
        mode="IN"/>
    </operation>
    <parameter name="allowedMethods" value="doSubtract"/>
  </service>
</deployment>
```
Example Deploy.wsdd Explained

- **deployment**: Root XML element that defines that the web service is to be deployed or installed
- **service**: Defines the Web Service that is to be installed. Attribute `name` is the identifier of the Web Service and `value` is the type of Web Service
  - `java:RPC`: Web Service is of type RPC
  - `java:MSG`: Web Service is of type message based
- **parameter**: Descriptor used to describe and define parent XML node, which could be `service` or `operation`.
- **operation**: Defines the operation that the Web Service will support. This tag is optional, but is useful for early binding of the SOAP request.
  - Early binding is faster, but requires definition
  - Late binding is dynamic, but slower

```xml
<deployment xmlns="http://xml.apache.org/axis/wsdd/">
  <service name="MathService"/>
</deployment>
```

Example Undeploy.wsdd

- **undeployment**: Root XML element that defines that the web service is to be undeployed or uninstalled
- **service**: Name of Web Service that is to be removed

```xml
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">
  <service name="MathService"/>
</undeployment>
```
WSDL Theory: Introduction

- A way of describing your web service
  - May or MAY NOT be SOAP based
  - Includes the notion of binding to SOAP or HTTP
- Note that WSDL binding to HTTP is not the HTTP SOAP binding

WSDL XML File

Main sections of a WSDL File
- Types
- Message
- Service references
- Port
- Binding, which references Port
- etc

WSDL Theory: Types

- Defines the data types for the message being sent
- Defines the namespaces used within the XML document
- Is represented as a valid XML Schema document within the WSDL file
WSDL Theory: Messages

- Defines the messages that are sent between the client and sender
- Is based on logical client and logical sender
- Logical is an abstract concept that has no relation to anything physical
- A message can have multiple parts to indicate multiple documents within the SOAP message (SOAP blocks)

WSDL Theory: Port Types

- Nothing to do with an operating system socket port
- A way of describing a sender and receiver
- Defines whether there is a request and response
- Four Operations
  - One Way: sender to receiver no response expect by client
  - Request Response: sender to receiver to sender
  - Solicit Response: receiver to sender no response expect by receiver
  - Notification Operation: receiver to sender to receiver
WSDL Theory: Binding
- Defines on how the Web Service is bound
- Up to this point SOAP has been Web Services, but Web Services need not be SOAP
- Defines physical bindings such as SOAP or HTTP
- Can be simple or complex
  - Is dependent on the binding implementation

WSDL Theory: Service
- WSDL parser entry point and pulls everything together
- Can figure out what is being defined and how to call the Web Service using which protocol and method
Axis Web Service Development

- Production environment is not based on manual coding
  - WSDL file plays central role, could be provided by another site or is the starting point of the development cycle
- In Axis typical development cycle is as follows:
  - Designing the interfaces using java interfaces and java beans
  - Compiling the java interfaces and java beans
  - Use Java2WSDL tool to generate WSDL file
  - Use WSDL2Java tool to generate server stubs and client stubs
  - Implement server stubs
  - Implementation of client application using client stubs

Java2WSDL

- Is part of Axis Framework as a command line tool
  java org.apache.axis.wsdl.Java2WSDL
- Generation of WSDL file is based on a single interface that references various Java Beans
  - Can be a Java Class, but WSDL2Java will generate an interface
- All Java programatic techniques are supported
  - But remember Java has limited abilities when compared to XML
    - Method parameters are input only
    - Nested classes are ignored
  - When defining Web Services using Java you need to assume that you are developing RPC type Web Services
  - Methods from interface are exported
  - Public Bean properties or public getters and setters are exported
**WSDL2Java**

- Is part of Axis Framework as a command line tool
  - `java org.apache.axis.wsdl.WSDL2Java`
- Generation of client and server stubs based on a WSDL file
- NOT ALL WSDL files can be converted into valid Java Class
- Best analogy is that if you can convert Java to WSDL then you can convert WSDL to Java
  - Anything extra most likely will not work
  - If it does work then the mapping is not that logical
  - Need to use Messaging or custom serialization to process WSDL files that cannot be converted
- If the WSDL file is RPC based then it is simpler to transpose into Java
  
  Experimentation is the best recipe for finding a solution
  - To get experience start simple and then go more complex

**Serialization and Data Conversion**

- When starting with a Java interface conversion to WSDL and back to a Java is simple and straightforward
- Complexity occurs when the WSDL file may be generated on another platform using another toolkit
  - Mapping from WSDL to Java may not be straightforward and simple
  - May need to map to legacy java beans and therefore WSDL file will not work
- When complexity occurs custom serialization and deserialization becomes necessary
- Custom serialization and deserialization involves the custom reading and writing of java data to XML
  - Is defined by a specific user implementation
Serialization Defined in Deploy.wsdd

- Type mapping is provided within the deployment file, which associates an XML element with a specific serialization and deserialization class
  
  - Special case of using *Bean* tag, but ignored for this course

```xml
<service name="BeanService" provider="java:RPC">
  <parameter name="className" value="ApacheJavaCourse.BeanServiceSoapBindingSkeleton"/>
  <parameter name="allowedMethods" value="SimpleObjectCall"/>
  <typeMapping xmlns:ns="http://ApacheJavaCourse" qname="ns:BeanObject"
    type="java:ApacheJavaCourse.BeanObject"
    serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
    deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
</service>
```
Serialization Specifics

- Serialization and De-serialization is not done from "scratch"
  - Basic infrastructure exists to serialize and deserialize simple data types
- Special serialization and de-serialization "extends" the basic process and converts a complex process into a simple process
  - The big idea is not to re-invent

Custom Handlers

- By default in Axis everything is a handler that is part of the calling chain
- When a service is called it is called by a class that derives from BasicHandler
  - To add a request or response handler before or after the pivot another BasicHandler derived class is added to the calling chain
- Can add a handler to the global calling chain or to the service specific calling chain
  - It is just another handler in the chain
- Handlers can track requests or responses, but should never replace the pivot handler unless necessary
  - Pivot point is the Web Service request handler
- Custom handlers typically perform logging or security tasks
Server Configuration with Handlers

- For global custom handlers need to add capability to the file `server-config.wsdd`
- For server specific handlers need to add capability to `deploy.wsdd` file

```xml
<deployment xmlns="http://xml.apache.org/axis/wsdd" >
  <globalConfiguration>
    <requestFlow>
      <handler type="java:org.apache.axis.handlers.JWSHandler"/>
    </requestFlow>
  </globalConfiguration>
  <globalConfiguration>
    <handler name="logger" type="java:ApacheJavaCourse.BeanServiceLogHandler">
      <parameter name="filename" value="MyService.log"/>
    </handler>
    <service name="BeanService" provider="java:RPC">
      <requestFlow>
        <handler type="logger"/>
      </requestFlow>
    </service>
  </globalConfiguration>
</deployment>
```

Custom Handler Implementation Details

- A custom handler is executed when the method invoke is called
- Custom handler will manipulate the MessageContext object
  - MessageContext object references the requestMessage, responseMessage, targetService, etc

```java
public class BeanServiceLogHandler extends BasicHandler {
  public void invoke(MessageContext msgContext) throws AxisFault {
    try {
      // ... Actions
    } catch (Exception e) {
      throw AxisFault.makeFault(e);
    }
  }
}
```
Send SOAP Messages

- SOAP messages are document based exchange of messages
- SOAP messages are based on the sending of free flowing text
- MSG: Like sending the contents of a book
  - Structured but not predictable
  - You know a message contains letters, paragraphs, etc, but not idea the order and when
- RPC: Sending of an invoice
  - Structured and predictable
  - You know the RPC has a specific set of parameters and data structures

SOAP Messaging

- When a document is sent via the message style different handlers are invoked

RPC: org.apache.axis.providers.java.RPCProvider
MSG: org.apache.axis.providers.java.MSGProvider

- Handler is defined within the deployment file

```xml
<deployment name="Message">
  <service name="MessageService" provider="java:MSG">
    <parameter name="className" value="ApacheJavaCourse.MessageService"/>
    <parameter name="allowedMethods" value="echoElements"/>
  </service>
</deployment>
```
SOAP Messages Differences

• In Axis when different handlers are used to process the request the deployment attributes differ in functionality
• Reference previous slide:
  – Attribute allowedMethods: does not reference which method is allowed, but which method is called when a message arrives
    • If not defined then the SOAP Body child element identifier is used, but this is hack and just a guess
• SOAP Messages can arrive in multiple child blocks to represent multiple documents

SOAP Attachments

• Appends data as another stream
• Not all data can be represented as XML
  – Some data is binary or non-XML based textual information
• Makes use of MIME encoding and XML HREF techniques
• Can be problematic with other implementations since it is not yet a widely used technique
• Is implemented as a regular function call with special type mapping in deployment file
  – Bean serialization technique, except serialization of attachments

```xml
<typeMapping
deserializer="org.apache.axis.encoding.ser.JAFDataHandlerDeserializerFactory"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
qname="tns2::DataHandler"
serializer="org.apache.axis.encoding.ser.JAFDataHandlerSerializerFactory"
type="java:javax.activation.DataHandler"
xmns:tns2="ClubMedAttachmentsService"/>
```
Example SOAP Attachments

POST /axis/services/ClubMedAttachmentsService HTTP/1.0
Content-Type: multipart/related; type="text/xml"; start="
<137791465.1020475278455.AXIS@hercules>";
boundary="----=_Part_0_1124227.1020475278465"
Authorization: Basic c29tZW9uZTpzb21ld2hlcmU=

-------=_Part_0_1124227.1020475278465
Content-Type: text/xml; charset=UTF-8
Content-ID: <137791465.1020475278455.AXIS@hercules>
<SOAP-ENV:Envelope>...
  <source href="cid:1115842168.1020475278334.AXIS@hercules"/>
...</SOAP-ENV:Envelope>

-------=_Part_0_1124227.1020475278465
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <1115842168.1020475278334.AXIS@hercules>
...
-------=_Part_0_1124227.1020475278465--

SOAP Attachments Implementation

- Attachment is a data type that is an object handler based on a namespace only known to the Axis environment
  - Makes cross platform more complicated
- Axis cross references the namespace in the WSDL to the qname type mapping in the deployment file
  - Type mapping cross references the serialization

```xml
<wsdl:definitions xmlns:tns1="ClubMedAttachmentsService">
  <types>
    <schema targetNamespace="ClubMedAttachmentsService">
      <element name="DataHandler" nillable="true" type="tns1:DataHandler"/>
    </schema>
  </types>
</wsdl:definitions>
```
Thanks
Questions?