Quick Start Axis2: From Newbie to SOAP Guru

By
Deepal Jayasinghe
WSO2 Inc. & Apache Software Foundation
About the Presenter

Technical Lead at WSO2 Inc.

- www.wso2.com
- A start-up aiming to develop and support leading edge Open Source Web service software
  - Provides support, training and consultancy for Web services

Apache Software Foundation

- Apache member
- WS-PMC member
- Contributes to a number of WS projects
- Frequent speaker at ApacheCon and SOA related conferences
- Axis2 developer since day one.
Apache WS Paradigm

Apache SOAP
- Developed by the IBM research team
- Donated by IBM shortly after IBM joined the SOAP/WS initiative
- Proof of concept

Apache Axis1
- SAX based
- Handler architecture
- Highly successful
  - Still used in many companies' products
Axis 1.x Add-ons

- **WSS4J**
  - Web service security support

- **Sandesha**
  - WS-Reliable Messaging

- **Kandula**
  - WS-Coordination, Atomic Transaction and Business Activities

- **Pubscribe**
  - WS-Notification

- **EWS**
  - Enterprise Web services /JSR 109

- **WSRF**
  - Resource Framework Implementation
If Axis1 is so wonderful ....

Why do we need Axis2?

Changes to Web service landscape
- WS-Addressing, Reliable Messaging

Performance
- Parsers, Optimizing based on use

Ease of use
- Hot Deployment and other capabilities
Axis2 Features

- High performance XML processing model
- Extensible messaging engine
- Context and description hierarchy
- Pluggable module architecture
- Easy to use deployment model
- Convenient client API with Asynchronous support
- Pluggable data binding
- Both WSDL 1.1 and 2.0 support
- REST support
- MTOM support
High Performance XML Processing Model (AXIOM)
AXIOM - **AXIS2 Object Model**

It is NOT, yet another XML object model

- API is more like a simplified DOM

**Fundamental difference?**

- Objects are created 'on demand' using a pull model
- Allows direct access to the underlying pull stream with or without building the tree
- Supports storing binary data (Inbuilt support for MTOM.)
AXIOM cont.

API also provides a StAX parser interface for any element
- Allows the event based navigation of the OM tree.
New XML Infoset Representation cont.
What has Axiom got to do with Axis2?

AXIOM is the primary means of representing/manipulating the XML message inside Axis2.

Any message coming into Axis2 represents as an AXIOM object.
Code is worth more than 100 slides 😊

Let's do some samples with AXIOM to understand its basic usages

- Serialization
- De-serialization
- Caching
- XPath navigation
How to Find AXIOM

AXIOM is a separate Apache Web service commons project
It has its own release cycle and community
You can download AXIOM from
http://ws.apache.org/commons/axiom/index.html
Get back to the topic: Axis2
We can download Axis2 from

- http://ws.apache.org/axis2/download.cgi

Latest version is 1.3 released on 8/14/2007

Release consists of four artifacts and IDE plugins

- Axis2 binary distribution
- Source distribution
- War distribution and
- Document distribution
- Eclipse and IntelliJ idea plugins
Deploying Axis2 cont.

Deploying a war distribution is just a matter of copying the .war file.

Once you deploy Axis2, go to http://localhost:8080/axis2.

Then you will see the Axis2 web administration console.

We can use that to view the runtime and perform basic management operations.

Let's navigate through Axis2 and see what it looks like.
Welcome!

Welcome to the new generation of Axis. If you can see this page you have successfully deployed the Axis2 Web Application. However, to ensure that Axis2 is properly working, we encourage you to click on the validate link.

- **Services**
  View the list of all the available services deployed in this server.

- **Validate**
  Check the system to see whether all the required libraries are in place and view the system information.

- **Administration**
  Console for administering this Axis2 installation.

---

Copyright © 1999-2006, The Apache Software Foundation
Licensed under the [Apache License, Version 2.0](http://www.apache.org/licenses/LICENSE-2.0)
Axis2 Happiness Page

Examining webapp configuration

Essential Components

  at C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\axis2\WEB-INF\lib\axis2-kernel-1.3.jar
Found Jakarta-Commons Logging (org.apache.commons.logging.Log)
  at C:\Program Files\Apache Software Foundation\Tomcat\%20\bin\commons-logging-api.jar
Found Streaming API for XML (javax.xml.stream.XMLStreamReader)
  at C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\axis2\WEB-INF\lib\stax-api-1.0.1.jar
Found Streaming API for XML implementation (org.codehaus.stax2.XMLStreamWriter2)
  at C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\axis2\WEB-INF\lib\wstx-asl-3.2.1.jar

The core axis2 libraries are present.

Note: Even if everything this page probes for is present, there is no guarantee your Axis Service will work, because there are many configuration options that we do not check for. These tests are necessary but not sufficient.

Examining Version Service

Found Axis2 default Version service and Axis2 is working properly.

Now you can drop a service archive in axis2/WEB-INF/services. Following output was produced while invoking Axis2 version service

Hello I am Axis2 version service, My version is 1.3
Available Services

Version

Service EPR: http://localhost:8080/axis2/services/Version

Service Description: Version

Service Status: Active
Engaged modules for the service

- addressing: Disengage

Available operations

- getVersion

Engaged Modules for the Operation

- addressing: Disengage
Deploying and Running the Binary Distribution

Deploying the binary distribution is just a matter of copying the file to the file system.

Running the binary distribution
- On Windows - axis2server.bat
- On Linux – axis2server.sh

Start Axis2 in port 8080
Go to http://localhost:8080

You will see the available services listed there
- Doesn't look as nice as a web administration application
Axis2 Service
Axis2 Service and POJO

POJO – Plain Old Java Object
Let's write a very simple Java class and make it a service
- The Java class has a method called “sayHello”

Compile the Java class and drop that into
- CATALINA_HOME/webapps/axis2 WEB-INF/pojo

Go back to
- http://localhost:8080/axis2/services

Now let's try to invoke the service in a REST manner
- See the output you got
Axis2 Service cont.

How about deploying a service and running it using only one line?

- Yes it is possible!
- new AxisServer.deployService(className)

Now browse
- http://localhost:6060/axis2
  - You will see the service

Now invoke the service
  - We will see the same output as above
Problem with POJOs

- We do not have control over the generated WSDL
- We do not have control over the service level configuration
  - (Using the JSR 181 annotation we can achieve some of them, but it is limited.)

More issues when we have dependencies in the Java class
- To some other Java beans or
- To a third party library

To solve that issue, Axis2 has a service configuration file called services.xml
services.xml

What is there:
- Service name
- Service class
- Parameters
- Operations
- Message receivers
- And more
Axis2 Service cont.

Concept of the service archive file
- Structure of a service archive file

Create a service and deploy that in Tomcat above service
Axis2 Deployment
What's the Fuss with Deployment?

Axis 1.x deployment requires you to:
- Either modify the XML files
  - or
- Call the admin client
- Add to the classpath
- Restart the server

For a beginner, it's a bit of a headache :( 
**Axis2 Deployment Model**

- **Archive based deployment**
  - Bundle all together and drop in

- **Directory based deployment** (similar structure as archive)

- **Hot Deployment 😊😊**

- **Archive file can contain:**
  - Class files
  - Third party libraries
  - Any other resources required by the service
Idea of a Repository

- A location in a file system to deploy our resources
  - Service
  - Modules (will be discussed soon)
- Repository may be located in the same machine or a remote machine
  - File system based configurator
  - URL based configurator
- Service and module isolation
  - Each service and module gets its own classloader
Axis2 Global Configuration File

Known as axis2.xml
Has all the configuration data
- Parameters
- Transport Senders/Receivers
- Message Builders/Formatters
- Flows and Phases, etc
Client API
Axis2 Client API

Two types of client

- ServiceClient
  • For average users

- OperationClient
  • For advanced users

Support for WSDL 2.0 MEPs

Support for both synchronous and asynchronous Web service invocation

- Application level async
- Transport level async
WS-Addressing and Asyn

WS-Addressing provides a transport independent mechanism and asynchronous web service invocation

- `<wsa:to>`
- `<wsa:replyTo>`

Let's use the ServiceClient to invoke the service in a fully asynchronous manner.
Axis2 Module
What is a module?

- Modules define the way of extending Axis2
- Encapsulates a specific functionality (mostly a WS-* function)
  - For example, the Addressing module adds WS-Addressing support
- Usually consists of a set of handlers
  - Modules are not hot deployable
    - Because they change the overall behaviour of the system
Configuring a Module

An idea about module.xml
- Handlers
- Phase rules
- Parameters

Module class

What is a Phase and Phase rule?
TransportIn → Pre-dispatch → Dispatch

Module

WS-A handler
A Word About Phase and Phase Rules...

Phase is a logical collection of handlers

Why do we need phase rules?
- dynamic handler chains

Writing phase rules
- Phase Name
- PhaseFirst
- PhaseLast
- Before
- After
- Before and After

How does phase resolving happen at the time of deployment and module engagement?
Concept of Availability
- Presence of the module in the system

Concept of **Engaging**
- Activating the module
- Can be done
  - Per System
  - Per service group
  - Per Service
  - Per Operation
Description and Context

Axis2 keeps logic and data separately
- It's good to keep them separately
There are two types of object hierarchies
- Deployment time data or static data
- Runtime data
Static data may have one or more runtime data associated with it
Life Time of Descriptions and Contexts

Axis* life time = System life time

*Context life time varies

Sharing data across different levels of descriptions and contexts
Parameters and Properties

Parameters
- Defining a parameter
- The 'locked' attribute
- Setting and getting
- Parameter can be any object
- Getting the original OMElement from the parameter

Properties
- Difference between property and parameter
- Accessing and retrieving property appropriately
Axis2 Runtime
Message Flow

XML Message

Transport

Engine

Message Receiver

Application
The Flow of a Message

Steps to handle a message inside Axis2

- Transport Receiver
- Engine
  - Dispatching
- Message Receiver
Step 1: Transport Receiver

Create a Message Context (MC)
Add transport information into MC
Create an AxisEngine
  • Remember, AxisEngine is stateless
Call engine.receive()
Step 2: Engine

Invoke the global phases
- Why do we need global phases?

Dispatch (explained on the next slide 😊)

Invoke service phases

Call the Message Receiver
- Can we pause the execution?
- Yes, but there are things to keep in mind!
Step 2.25 - Dispatching

Two types of dispatching
- Finding the corresponding descriptions
- Finding the corresponding contexts

Default dispatchers
- AddressingBasedDispatcher
- RequestURIBasedDispatcher
- SOAPActionBasedDispatcher
- SOAPMessageBodyBasedDispatcher
Step 2.25: Dispatching cont.

Order of tasks in dispatching
- Finding the Operation context
- Finding the Service context
- Finding the Service group context
Step 3: Message Receiver

- The last handler of the execution chain
- MEP dependent (MEP??)
- Does the actual business logic invocation
- Ability to write custom message receivers
- Supports dependency injection !!

Some default Message Receivers

- RawXMLINOnlyMessageReceiver
- RawXMLINOutMessageReceiver
- RPCMessageReceiver
Message Exchange Patterns - MEP

- Describes the exchange pattern of SOAP messages per given operation.
- For example,
  - In – Out
  - In Only
  - In – In – Out
- WSDL 2.0 defines 8 standard MEPs.
- Axis2 supports all inbound MEPs
Axis2 Session Management
What is session management?

Web services are said to be stateless. However, it is very difficult to implement enterprise level applications without having session management support.

There are four types of sessions:
- Request scope
- Transport scope
- Application scope
- SOAP session scope
Tools
The New Toolkit

Plugin for Eclipse and IntelliJ IDEA
Java2WSDL, Java2WSDL2 and WSDL2Code tool
Tools to generate service and module archives
Generating WSDL from a Java Class

Write a Java class and generate a WSDL

Two options
- doclit bare
- doclit wrapped

Two types
- WSDL 1.1
- WSDL 2.0
Tools: Code Generation

java org.apache.axis2.wsdl.WSDL2Code

Usage WSDL2Code -uri <Location of WSDL> :WSDL file location
-o <output Location> : output file location
-a : Generate async style code only. Default is off
-s : Generate sync style code only. Default is off. takes precedence over -a
-p <package name> : set custom package name
-l <language> : valid languages are java and csharp. Default is java
-t : Generate TestCase to test the generated code
-ss : Generate server side code (i.e. skeletons).Default is off
-sd : Generate service descriptor (i.e. axis2.xml).Default is off.Valid with –ss
-d: choose databinding model – adb, jaxb, xmlbeans, none
Generated Code: Client

Structure

- Stub and Interface
- Empty Callback Handler
- Databinding supporter classes – Depends on the selected databinding framework
- Databinding classes - Depends on the selected databinding framework
- Ant build file
Generated Code: Service

Structure

• Skeleton
• In-Out MEP based Message Receiver
Code again...

- Codegen demonstration with the command line tool
MTOM

There are two ways to transfer binary data:

- Inline in the XML
  - base64 – 4/3x original size
  - hex – 2x original size
- Reference
  - pointer to outside the XML

MTOM allows the best of both worlds

- Appears as if it is in-line even when it’s configured
- Same programming model
- Standardized attachments
MTOM/XOP Example

```xml
<soap:Envelope
   xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
   xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
  <soap:Body>
    <m:data xmlns:m='http://example.org/stuff'>
      <m:photo xmlmime:contentType='image/png'>
        <xop:Include
           xmlns:xop='http://www.w3.org/2004/08/xop/include'
           href='cid:http://example.org/me.png'/>
      </m:photo>
    </m:data>
  </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/png
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/me.png>

// binary octets for png
Axis2 MTOM Support

```java
OMElement data = factory.createOMElement("binaryData", xNs);

// Creating the Data Handler
FileDataSource dataSource = new FileDataSource("c:\test.data");
DataHandler dataHandler = new DataHandler(dataSource);

//create an OMText node
//optimised = true means by reference
//use optimised for large data, inline for small
OMText textData = factory.createText(dataHandler, true);
data.addChild(textData);

Explicitly required to enable MTOM support in axis2.xml
```
Axis2 natively supports REST/pure XML/HTTP

Uses Content-type/soap-Action headers to differentiate SOAP vs REST
Available Transports

Other transports

- SMTP/POP based transport
- TCP based transport
- JMS based transport (yet to come)

Can easily switch between different transports

JMX Management Console (Google SoC project)
Web Services with SMTP

Deploys a service in an SMTP enabled environment
Invokes a service in an SMTP manner
Sends using HTTP and Receives using SMTP
Resources and Contribution

Code sample
- http://people.apache.org/~deepal/acus07/

More information can be found at
http://wso2.org/library/2060
http://today.java.net/pub/a/today/2006/12/13/invoking-web-services-using-apache-axis2.html
- http://ws.apache.org/axis2
- wso2.org
Questions ?
Thank You !!