Accelerating Web Services Development with Axis2

By
Deepal Jayasinghe / Ajith Ranabahu
Apache Software Foundation & WSO2
About us

- Both Apache committers
  - Axis2
  - Synapse
- Part of the Axis2 team from Day 1
- Working for WSO2
Motivation for Axis2

- History of ASF SOAP engines (paradigm)
  - Apache SOAP
  - Axis 1.x designed as a follow-on
- Why do we need a new SOAP engine?
  - Changes to the Web services landscape
    » WS-A, WS-RM
  - Performance
    » Parsers, Optimizing based on use
  - Ease of use
    » Deployment of new capabilities, service deployment
Axis2 features

- High Performance XML Processing Model
- Extensible Messaging Engine
- Improved Context Hierarchy
- Pluggable Module Architecture
- Improved Deployment Model
- New Client API
- Optional Pluggable Data Binding
- WSDL 2.0 Support
- REST Support
High Performance XML Processing Model
New XML Infoset Representation

• Known as AXIOM (AXIS Object Model)
• 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
  – Support for storing binary data
New XML Infoset Representation (Cont…)

• API also provides a StAX parser interface at any element
  – Allows the event based navigation of the OM tree.
  – <a>
  –  <b/>
  – </a>
New XML Infoset Representation (Cont…)
New XML Infoset Representation (cont..)

• In built binary storage support
  – Can store binary (unchanged)
  – Natively supports XOP/MTOM

• XOP? MTOM??
AXIOM and Axis2

- AXIOM is the primary means of representing / manipulating the XML message inside Axis2
Time to Dig Into Code.. 😊

• Code samples to explain AXIOM
  • Serialization
  • De-serialization
  • Caching
  • XPath navigation
Extensible Messaging Engine
Axis2 Terminology

• Handler
• Phase
  – Phase rules
• Flow (Execution chain)
• Extensible module
• Service
• Message Context
A Word About Phase and Phase Rules...

• Phase is 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 phase resolving happens at the deployment time and module engagement time
Phase and Phase Rules (Cont...)

• Example

```
<inflow>
    <handler
        name="SampleHandler"
        class="org.apache.axis2.SampleHandler">
        <order phase="userphase1" />
    </handler>
</inflow>
```
Extensible Messaging Engine

The Message 'Flow'

XML Message
Transport
Engine
Message Receiver
Application
The Flow of a Message

• Steps of handling a message inside Axis2
  – Transport Receiver
  – Engine
    • Dispatching
  – Message Receiver
Step1 : 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 (wait till next slide 😊)

• Invoke service phases

• Call the Message Receiver
  – Can we pause the execution ?
    • Yes , but there are things to keep in mind!
Step 2.5 - Dispatching

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

• Default dispatchers
  • AddressingBasedDispatcher
  • RequestURIBasedDispatcher
  • SOAPActionBasedDispatcher
  • SOAPMessageBodyBasedDispatcher
Step 2.5 : Dispatching (Cont..)

• Order of tasks in dispatching
  • Finding Operation context
  • Finding Service context
  • Finding 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.
• E.g.
  – In – Out
  – In Only
  – In – In – Out !
• WSDL 2.0 defines 8 standard MEPs.
• Axis2 supports all inbound MEPs
Coding time again...

• Write a Custom Message Receiver
Contexts and Descriptions
Hierarchy
Why are the Contexts and Descriptions Needed?

- Descriptions keep static information
  - Information extracted from deployment descriptors
- Contexts keep runtime information
- This Information needs to be in various scopes
- Good to keep them separate!
Life Time of Descriptions and Contexts

• Axis* life time = System life time
• *Context life time varies
• Sharing data across different level of descriptions and contexts
Parameters and Properties

- **Parameters**
  - Defining parameters
  - The “locked” attribute
  - Setting and getting
  - Parameter can be any object
  - Can get an OMElement as the parameter

- **Properties**
  - Difference between property and parameter
  - Accessing and retrieving property appropriately
Pluggable Module Architecture
What is a module?

- Modules define the way of extending Axis2
- Encapsulates a specific functionality (mostly a WS-* function)
  - e.g. Addressing module adds WS-Addressing support
- Usually consists of a set of handlers
- Modules are not hot deployable
  - Because they change the overall behavior of the system
Inside an Axis2 Module

- What does it contain?
  - Module descriptor: module.xml
    - (more in the next slide)
  - Module implementation class
  - Handlers
  - Third party libraries

- Can be deployed as an archive file 😊
  - Bundle all together and deploy

- Can be deployed as a directory as well

- Isolated – separate class loader
Module Descriptor

<module>
  <inflow>
    <handler name="AddressingInHandler"
             class="org.apache.axis2.handlers.addressing.AddressingInHandler">
      <order phase="PreDispatch"/>
    </handler>
  </inflow>

  <outflow>
    <handler name="AddressingOutHandler"
              class="org.apache.axis2.handlers.addressing.AddressingOutHandler">
      <order phase="MessageOut"/>
    </handler>
  </outflow>
</module>
Availability and Engaging of Modules

• Concept of Availability
  – Presence of the module in the system
  – Module loaded, but not active

• Concept of *Engaging*
  – Activating the module
  – Can be done
    • Per System
    • Per Service Group
    • Per Service
    • Per Operation
Back to Code ...

- Sample module with two handlers
- Sample module with a module implementation class
Improved Deployment Model
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, 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
Concept of Repository

• A directory containing all the resources needed for Axis2 to run

• Structure
  – axis2.xml – overall settings
  – classes\ - shared classes
    • .class
  – lib\ - shared libraries
    • *.jar
  – modules\ -
    • foo.mar
  – services\ 
    • bar.aar
Axis2 Services
What is a service?

- Can be deployed as an archive (.aar) file or as a directory with all necessary resources
- Isolated – separate Class loader
Service Descriptor

• Service configurations are given by the services.xml
  - No need to have a WSDL around to be a valid service !!!

• Contains
  - ServiceClass parameter
  - Operation
    - wsa:mapping
    - MessageReceiver
    - Messages
  - Modules to be engaged
  - Module configurations (module parameters)
Service vs. Service Group

- Deploying multiple services together
- Share data across services in a group
- Maintain sessions across a service group using contexts

- Example: use a Service Group
  - Log-in
  - Run process
  - Log out
Back to Code..

- Samples (without data binding)
  - OM-in OM-out
  - Use of RPC Message Receiver
New Client API
Client API

• Supports both blocking and non-blocking invocations models
  – Concept of callbacks for the client for non-blocking case
• Can handle both transport dependent and transport independent asynchrony.
• Based on the “MEP Client”
  – Default support to In-Only and In-Out MEPs
  – Can be extended to support custom MEPs
Important Classes in Client API

- InOutMEPClient
- InOnlyMEPClient
- Call
- MessageSender
- RPCCall
- RESTCall
There's Nothing Like Code to Explain it!

- Simple Client written from scratch
Tools
The New *Toolkit*

- Plugin for Eclipse and IntelliJ IDEA
- Improved WSDL2Code tool
- Axis Administration Web Application
- Tools to generate service and module archives
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 if off
-s : Generate sync style code only. Default if 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, xmlbeans, none
Generated Code : Client

• Structure
  • Stub & Interface
  • Empty Callback Handler
  • Databinding supporter classes – Depends on the selected databinding framework
  • Databinding classes - Depends on the selected databinding framework
  • Ant build file
• The DataBinding support creates a Support class that takes XML Beans objects and converts to AXIOM

public class StockQuotePortTypegetQuoteDatabindingSupporter {
    public static OMEElement toOM(XmlObject param);
    public static XmlObject fromOM(OMElement param);
}
Generated Code : Service

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

- Codegen demonstration with the command line tool
Advanced Topics
MTOM
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 best of both worlds
  – Appears as if it is inline even when it’s pointed to
  – 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 xmlnsmime: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
OMElement data = factory.createOMElement("binaryData", xNs);

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

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

Explicitly need to enable MTOM support in axis2.xml
Advanced Topics
REST
REST

• Axis2 natively supports REST / pure XML/HTTP

• Switch on in axis2.xml
  – <parameter name="enableREST" locked="xsd:false">true</parameter>

• Off by default in current build

• Uses Content-type / SOAP-Action headers to differentiate SOAP vs REST
Advanced Topics
Class Loading Hierarchy
New Class Loading Hierarchy

```
Axis2 repo
   /\     /
  Lib  Modules  Services
     /\      /\      /\    
    Lib  Module1  Service1 Lib
           /\    /\    /\     
          Lib Lib Lib
```
Other Improvements
Configuring Axis2 (axis2.xml)

- Important parameters
  - Deployment features
  - Enabling REST
  - Enabling MTOM
  - Serialization location (the location that the engine.serialize)
- Specifying Transports (both Senders and Receivers)
- Engaging module
- Adding Listeners
- User name /password
- Module Configuration
- Phases orders (with flow)
- Axis Storage
WSDL Object Model

- WSDL 2.0 compatible object model (WOM)
  - Feeders available to pump WSDL 1.1 into WOM.
  - Complete support for WSDL 1.1, full WSDL 2.0 support coming soon.
Other Improvements (Cont ..)

• 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)
Other Improvements (Cont ..)

• WSS4J support
  – Successfully completed interoperability with Indigo using Axis2 with WS-Security, MTOM and WS-Addressing
Other Improvements (Cont...)

• Support for other languages
  – Ability to write implementations in various JVM languages
    • Groovy
  – Client stub generation for languages other than Java !!!
    • IKVM based C# code !
Resources

• Code samples
• More information can be found at
  – http://ws.apache.org/axis2
Next Release ......

• 1.0 – almost ready to be released
• M1, M2, 0.90, 0.91 and 0.92 already released....
• 0.93 was released on 2\textsuperscript{nd} December
Questions ?
Thank You