Apache Axis2
The New Generation of Open Source Web Services

Paul Fremantle
VP of Technology
WSO2
paul@wso2.com
www.wso2.com

Thanks to: Ajith Ranabahu, Eran Chinthaka
And of course the Axis2 team
About the Presenter

- Ex-IBM Senior Technical Staff Member
  - Led the creation and development of
    - IBM Web Services Gateway
    - Web Services Invocation Framework
    - JWSDL / WSDL4J

- VP of Technology at WSO2
  - [www.wso2.com](http://www.wso2.com)
  - A startup aiming to develop and support leading edge Open Source Web services software
    - Provides support, training and consultancy for Apache Axis 1.x

- Co-Chair of the OASIS WS-Reliable eXchange Technical Committee

- Co-author of *Building Web Services in Java 2nd Edition*

- Blog [http://www.bloglines.com/blog/paulfremantle](http://www.bloglines.com/blog/paulfremantle)
Contents

- Introduction to Apache
- Axis history
- Overview of Axis 1.x
- Changing face of Web Services
- Axis2 objectives
- AXIOM
- Async
- Installing and using
- Conclusions
Apache

- A global open source organisation
  - Emerged out of the practice of “patching” the NCSA server
  - Now the most popular web server with ~60% share

- Numerous projects including
  - Web serving, Struts, Portal, J2EE
  - Core Java JVM, Database, Build tools, Ant, Maven
  - XML parsers, Web Services

- Meritocracy
  - Earn the right to be a committer in a project
  - Each committer gets one vote
Axis

- First there was Apache SOAP
  - written by IBM Research team
  - donated by IBM shortly after IBM joined the SOAP/WS initiative
- Axis 1.x designed as a follow-on
  - SAX based parsing
  - Handler architecture
  - Highly successful
    - Re-used in many companies’ products
Axis Add-ons

- **WSS4J**
  - Web Services Security support

- **Sandesha**
  - Web Services Reliable Messaging

- **Kandula**
  - WS – Co-ordination, Atomic Transaction and BusinessActivity

- **Pubscribe**
  - WS Notification

- **EWS**
  - Enterprise Web Services / JSR 109 support

- **WSRF**
  - Resource Framework implementation

see [http://ws.apache.org](http://ws.apache.org) for more information
If Axis Is So Wonderful…. 

- Why do we need Axis2?

- Changes to the Web services landscape
  - WS-Addressing, Reliable Messaging, Composability

- Performance
  - Parsers, Optimising based on use

- Ease of use
  - Deployment of new capabilities, service deployment
Changes to the WS Landscape

- **WS-Addressing**
  - Spec authored by IBM, Microsoft, Sun, BEA, SAP
  - August 2005 – W3C candidate recommendation

- Two main aspects
  - EndpointReferences – “pointers” to service endpoints
  - Message Addressing Properties – to/reply, etc.
EndpointReferences

- An XML encoding of an address where a service can be found

```xml
<wsa:EndpointReference>
  <wsa:Address>
    http://wso2uk.dyndns.com:8080/service/test
  </wsa:Address>
  <wsa:ReferenceParameters>
    <!-- this header added to messages to this EP -->
    <my:customerId>X98KLJJKLH87A</my:customerId>
  </wsa:ReferenceParameters>
  <wsa:Metadata
    xmlns:wsdli="http://www.w3.org/2004/08/wsdl-instance"
    wsdli:wsdlLocation="http://wso2.com/service/test.wsdl">
    <wsaw:InterfaceName>wso2:TestInterface</wsaw:InterfaceName>
  </wsa:Metadata>
</wsa:EndpointReference>
```
Message Addressing Properties

- Basic to/from/reply-to headers

```xml
  <wsa:To>http://wso2.com/test/service</wsa:To>
  <wsa:ReplyTo>
    <wsa:Address>http://fremantle.org/reply</wsa:Address>
    <wsa:ReferenceParameters>
      <custId>D64B3D4E4C7E58D53411305924340151</custId>
    </wsa:ReferenceParameters>
  </wsa:ReplyTo>
  <wsa:MessageID>8D53411305924340622</wsa:MessageID>
</s:Header>
```
WS-Addressing immediately adds asynchronous support to SOAP

Before asynchronous behaviour required:
  - Two one-way services with an implied req-resp behaviour
  - An asynchronous transport underneath – e.g. MQSeries, JMS, etc.

WS-Addressing adds direct HTTP (or other) support into the SOAP headers
Message Exchange Patterns

- WSDL 2.0 and SOAP 1.2
  - [http://www.w3.org/TR/wsd120-extensions/](http://www.w3.org/TR/wsd120-extensions/)
    - In-Only
    - Robust In-Only
    - In-Out
    - In-Optional-Out
    - Out-Only
    - Robust Out-Only
    - Out-In
    - Out-Optional-In

- Fundamentally a message based model not an RPC model
WS-ReliableMessaging

- Builds on top of WS-Addressing to add reliable delivery
  - at least once
  - at most once
  - exactly once
  - in order
Composability

- Security + Reliability (RAMP)
- Security + Reliability + Transactions

At some point you have to:
- Sign
- Encrypt
- Add message #
- Store
- Send

How do you interleave the security plugin with the reliability plugin?
RM – one-way

- Composable with existing logic

```
createSequence
  Send #1
  Ack #1
  Send #2
  Send #3
  Ack #1, #3
  Send #2
  Ack #1
```

App

Sender

Receiver

App
Composability Requires Async

- RM – with existing blocking req-resp

Diagram:
- App
- MW
- Call
- Block
- Timeout fault
- Send
- Send
- Send
- Send
- Receive
Performance

XML performance is tricky!

Traditional parsers
- DOM – reads whole message into a tree
  - Slow and memory intensive
- SAX – event driven parser drives callbacks
  - Good, but must deal with whole message
  - Axis 1.x has to store all events to replay them for WSS support

New generation – Pull Parsers
- STAX API is the Java Community standard
  - “Give me the next event”
  - Allows partial parsing of the message
Choosing the Right Parser

The problem is optimising for different cases

- Simple, unsecure, stub based access
  - Ideally pass the stream directly to code-generated de-marshallers
  - Just like CORBA used to do!
  - Parse the complete message

- Fully secured, encrypted, signed
  - Need a DOM-like tree structure to run DOMHASH

- Intermediary
  - Parse and modify the headers and then stream the body
Axis2
Axis2 Main Features

- New high performance object model (Axiom)
- Asynchronous and message based model
  - Better support for MEPs
- Improved support for composition and extensibility
  - Modules and phases
- Enhanced deployment and isolation
- Flexible Data Binding
- MTOM support
  - Standardised approach to attachments
- REST support
The Axis2 Team (in alphabetical order)

Committer

- Eran Chinthaka
- Glen Daniels
- Jaliya Ekanayake
- Thilina Gunaratne
- Chathura Herath
- Deepal Jayasinghe
- Srinath Perera
- Jeyachandra Rao
- Ajith Ranabahu
- Venkat Reddy
- Ashutosh Shahi
- Aleksander Slominski
- Davanum Srinivas
- Dasarath Weeratunga
- Sanjiva Weerawarana

Contributors

- Shawn Dahlen
- Geoffrey Fox
- Paul Fremantle
- Tom Jordahl
- Cheng Shin Lee
- Steve Loughran
- Gulliame Sauthier
AXIOM

The Heart of Axis2
AXIOM / OM

One OM to rule them all and in the runtime bind them?

- AXIIs Object Model
  - A lightweight, low-memory footprint, high-performance object model
- Multiple ways of looking at the same data
  - Tree structure
  - STAX pull stream
  - SAX API
- Supports caching of the tree and pull stream
  - So that you can read the stream and then read the tree
  - Or read the tree and then read the stream
- Supports MTOM (see later) – built in binary representation
- Supports DataBinding frameworks
  - via STAX and SAX
  - JAXB, Apache XMLBeans
The Key to Axiom Performance

- Do only what is needed, when it’s needed

The layabout’s model of efficiency!

Only build a tree when requested
Only build as much as requested
Only parse the stream when needed
Only parse as much as is needed
AXIOM Example 1 –
Creating an XML Document

```java
OMFactory factory =
    OMAbstractFactory.getOMFactory(); // access to OM
OMNamespace freoNs =
    factory.createOMNamespace("http://fremantle.org/ns/test", "freo");
OMElem person =
    factory.createOMElem("person", freoNs);
OMElem name =
    factory.createOMElem("name", freoNs);
name.setText("Paul Fremantle");
person.addChild(name);
```
Outputting the XML

```java
XMLOutputFactory xof = XMLOutputFactory.newInstance();
    //STAX
try {
    XMLStreamWriter writer =
        xof.createXMLStreamWriter(System.out);
    person.serialize(writer);
    writer.flush();
} catch (XMLStreamException e) {
    e.printStackTrace();
}
```
The Result

<freo:person
  xmlns:freo="http://fremantle.org/ns/test">
  <freo:name>Paul Fremantle</freo:name>
</freo:person>
AXIOM Example 2

String xml =
    "<people><person gender="male">"
  "<name>Paul</name><hair>Spiky</hair><eyes>Grey</eyes>"
  "<height measure="cms">178</height></person></people>"
byte arr[] = xml.getBytes();
ByteArrayInputStream bais = new ByteArrayInputStream(arr);

XMLStreamReader reader = null;
try {
    XMLInputFactory xif = XMLInputFactory.newInstance();
    reader = xif.createXMLStreamReader(bais);
} catch (XMLStreamException e) {
    e.printStackTrace();
}
StAXOMBuilder builder = new StAXOMBuilder(reader);
OMElement people = builder.getDocumentElement();
DOM-like Access to the Tree

```java
QName ps = new QName("person");
QName nm = new QName("name");
Iterator it = people.getChildrenWithName(ps);
OMElement pers = (OMElement)it.next();
Iterator it2 = pers.getChildrenWithName(nm);
OMElement name= (OMElement)it2.next();
OMText firstName = (OMText)(name.getFirstOMChild());
String nameText = firstName.getText();
System.out.println(nameText);
// Paul
```
Lazy Evaluation

- You can see the lazy evaluation....
  - but it’s not simple
- The Java inputstream is buffered
  - on my Sun JDK 1.4.2 – 8192 bytes
- So... take a large XML > 8192
  - Read first few tags
  - Then look at the remainder of the ByteInputStream
- However, if you looked at the parse events you would see exactly where the previous parse ended.
Ease of Use

- Axis 1.x deployment requires you to:
  - Either modify the XML files
    - or
  - Call the admin client
- Add to the classpath
- Restart the server
Axis2 Hot Deployment

Axis2 has a “hot deployment” model

- You create a JAR file that contains
  - the service class file, and
  - a deployment descriptor
  - any service handlers
  - any required libraries

- If Hot Deploy is enabled
  - Drop it into the repository folder

AAR – Axis2ARchive
META-INF\service.xml
Service class files
Service handlers
Service libs
Axis2 Repository

- `<repository dir>`
  - `axis2.xml` – overall settings
  - `classes` - shared classes
    - `.class`
  - `lib` - shared libraries
    - `.jar`
  - `modules` – next slide 😊
    - `.mar`
  - `services`
    - `.aar`
Axis2 Modules

- Modules are how Axis2 is extended:
  - Supports composability
  - Can add support for new WS-standards simply and cleanly
    - e.g. addressing.mar supports WS-Addressing
    - Module Archive contains
      - module.xml, classes, libs, handlers
    - Modules are not hot deployable
      - Because they change the overall behaviour of the system
Phases

- In order to support the addition of new functionality, there are a set of phases into which handlers are deployed.
- The phases are configured by the axis2.xml file, with predefined phases:

```xml
<phaseOrder type="inflow">
    <phase name="TransportIn"/>
    <phase name="PreDispatch"/>
    <phase name="Dispatch"/>
    <phase name="PostDispatch"/>
    <!-- After Postdispatch phase module author or service author can add any phase he wants -->
    <phase name="userphase1"/>
</phaseOrder>
```
How Addressing Fits In

```xml
<module name="addressing">
  <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>
```
Phases, Modules, Handlers

- TransportIn
- Pre-dispatch
- Dispatch
- Post-Dispatch

WS-A handler

Module

Axis2
Client API

OMElement payload = // some payload
Call call = new Call();
call.setTo(targetEPR);

call.setTransportInfo(Exists.TRANSPORT_HTTP,
               Constants.TRANSPORT_HTTP, false);

    //Blocking invocation
    OMEElement result =
        call.invokeBlocking("echo", payload);
Asynchronous Calling

Callback callback = new Callback() {
    public void onComplete(AsyncResult result) {
        try {
            StringWriter writer = new StringWriter();
            result.getResponseEnvelope().serializeWithCache(XMLOutputFactory.newInstance()
                .createXMLStreamWriter(writer));
            writer.flush();
            System.out.println(writer.toString());
        } catch (XMLStreamException e) {
            e.printStackTrace();
        }
    }
};
Asynchronous Calling, Part 2

//Non-Blocking Invocation

```java
call.invokeNonBlocking("echo",
                     payload, callback);
```

//Wait till the callback receives the response.
```java
while (!callback.isComplete()) {
    Thread.sleep(1000);
}
```
Callback API

package org.apache.axis2.clientapi;
public abstract class Callback {

    // filled in by programmer
    public abstract void onComplete(AsyncResult result);

    // filled in by programmer
    public abstract void reportError(Exception e);

    // called by programmer
    public boolean isComplete();
    public void setComplete(boolean complete);
}
Getting Started with Axis

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

Three options

- Download the latest source with subversion
- Download the stable source tree and build
- Download the stable binary driver
Download the Latest Source

- Subversion is a CVS alternative
- Based on WEBDAV / DeltaV
- Lots of cool features that are irrelevant to this presentation 😊
- [http://subversion.tigris.org/](http://subversion.tigris.org/)

Building Axis2

Maven

- Install Maven
  - [http://maven.apache.org](http://maven.apache.org)
  - Maven is a very cool build tool that expands on Ant
    - On Windows, I like to change
    - MAVEN_HOME_LOCAL=c:\maven
      (the default is some path that includes ‘_’s and breaks java classpath)
  - cd \axis2
  - maven
Axis2 builds a nice WAR file that can be deployed in Tomcat:

- Or download from the distribution site
- Just drop it into tomcat\webapps directory
- http://localhost:8080/axis2
Axis Happiness

Axis 2

Axis2 Happiness Page

Examining webapp configuration

Needed Components

Found Apache-Axis (org.apache.axis2.transport.http.AxisServlet) at C:\tomcat\webapps\axis2WEB-INF\lib\axis2-0.91.jar
Found Jakarta-Commons Logging (org.apache.commons.logging.Log) at C:\tomcat\bin\commons-logging-api.jar
Found Log4j (org.apache.log4j.LogLayout) at C:\tomcat\webapps\axis2WEB-INF\lib\log4j-1.2.8.jar
SimpleHTTPServer

> java
  org.apache.axis2.transport.http.SimpleHTTPServer
repos 3052

starting SimpleHTTPServer in port 3052 using the repository C:\axdb\target\lib\repos

[Axis2] Using the Repository C:\axdb\target\lib\repos
[Axis2] Starting the SimpleHTTPServer on port 3052
[Axis2] SimpleHTTPServer started
Data Binding and User APIs

- Axis2 has been designed to be much more “loosely coupled” to the data binding method

Why?

- Many of the problems users have with Axis1.x/JAX-RPC are issues with data binding
- XML data binding is better established
  - Don’t re-invent it
- Makes the right split between application code and middleware
  - the data binding is part of the application code not the middleware!
- It’s analogous to Object-Relational binding
  - Spring / Hibernate, etc. all point towards separating binding from connection
Coding to Real Services

- Two options
  - AXIOM + Call API
    - full control of the XML body
    - Can use your own data binding or just write AXIOM code
  - Code generation
    - WSDL2Java / WSDL2Code
    - uses XMLBeans data binding
    - Recently added a simpler “Axis Data Binding” – ADB
Writing a Simple Client – AXIOM

Creating the Body

<ns1:getQuote xmlns:ns1="urn:xmethods-delayed-quotes">
  <symbol>IBM</symbol>
</ns1:getQuote>

OMFactory factory = OMAbstractFactory.getOMFactory(); // access to OM
OMNamespace xNs = factory.createOMNamespace(
    "urn:xmethods-delayed-quotes", "x");
OMElement getQuote = factory.createOMElement("getQuote", xNs);
OMElement symbol = factory.createOMElement("symbol", xNs);
getQuote.addChild(symbol);
symbol.setText("IBM");
Calling the Service

Call call;
call = new Call();

EndpointReference targetEPR = new EndpointReference("http://64.124.140.30:9090/soap");
call.setTo(targetEPR);

call.setTransportInfo(Constants.TRANSPORT_HTTP,
                      Constants.TRANSPORT_HTTP, false);

// Blocking invocation
OMElement result = call.invokeBlocking("getQuote", getQuote);
Reading the Result

QName gQR =
    new QName("urn:xmethods-delayed-quotes",
        "getQuoteResponse");
QName Result = new QName("Result");
OMElement qResp = (OMElement)
    result.getChildrenWithName(gQR).next();
OMText res = (OMText)
    qResp.getChildrenWithName(Result).next();

System.out.println(res.getText());
Tools

- Plugin for Eclipse
- Improved WSDL2Code
- Axis Admin Web Application
- Tools to generate service and module archives
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
Code Generation

- The DataBinding support creates a Support class that takes XML Beans objects and converts to AXIOM

```java
public class StockQuotePortTypegetQuoteDatabindingSupporter {
    public static OMElem toOM(XmlObject param);
    public static XmlObject fromOM(OMElement param);
}
```
Creating a Service

- Simply write a class like:

```java
public class MyService {
    public OMElement myOperation(OMElement body) {
        // examine body
        // do work
        // create response OMElement
        return response;
    }
}
```
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
  - Standardised attachments
MTOM / XOP Example

<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 xmlns:xmlmime='http://www.w3.org/2004/11/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 need to enable MTOM support in axis2.xml
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
Axis2 and WS-Security

- At the recent Microsoft PDC
  - MS demonstrated interop between Indigo and Axis2:
    - SecureMTOM
    - WS-Addressing, MTOM, WSSec 1.0 –
      - encryption
      - digital signatures

Apache Synapse

- An incubator project in Apache to create a Web Services centric mediation / bus framework:
  - Built on Axis2 and AXIOM
  - Using WS-* standards and model as the semantics
  - Open source

- See
Summary

- Last year I presented 10 “rules” on how to approach SOA
- Axis2 directly addresses:
  - 1. The right contracts – getting the right divides between application logic and middleware
    - Better phase+handler model, streaming support, databinding architecture
  - 4. Split logic for Headers and Body to handlers and app code
    - Improved handlers and phases
  - 5. If you are doing XML, do XML
    - Axiom
  - 6. Be ready for instances and WS-Addressing
    - WS-Addressing support
  - 7. Intermediate
    - Synapse, improved intermediary design
  - 8. Be prepared for async + reliability
  - 9. Services are not objects – message based interaction model
Further Information

- **Axis2 website:**
  - [http://ws.apache.org/axis2](http://ws.apache.org/axis2)

- **Mailing list**
  - [http://marc.theaimsgroup.com/?l=axis-dev&r=1&w=2](http://marc.theaimsgroup.com/?l=axis-dev&r=1&w=2)
  - [http://marc.theaimsgroup.com/?l=axis-user&r=1&w=2](http://marc.theaimsgroup.com/?l=axis-user&r=1&w=2)

- **Some excellent Axis2 articles:**
  - [http://www.jaxmag.com/itr/online_artikel/psecom,id,747,nodeid,147.html](http://www.jaxmag.com/itr/online_artikel/psecom,id,747,nodeid,147.html)
  - [http://jaxmag.com/itr/online_artikel/psecom,id,757,nodeid,147.html](http://jaxmag.com/itr/online_artikel/psecom,id,757,nodeid,147.html)