Introducing AXIOM
Pull Based XML Object Model Optimized for SOAP Processing

Paul Fremantle
WSO2 Inc.
About Me...

○ Apache Software Foundation - Member
○ Apache Web Services - PMC Member
○ Apache Axis2, Axiom – Committer, Release Manager.
○ Apache Synapse - Committer
○ Member of W3C Working Group for WS-Addressing, WSDL 2.0
○ Working with WSO₂ on Axis2, Axiom, Synapse and Tungsten.
Agenda

- Introduction
- XML Processing Technologies
- AXIOM Features
- AXIOM Architecture
  - Deferred Building
  - Caching
  - Factories
  - Input/Output
- Using Axiom – Code examples
- AXIOM Performance
Agenda

- Introduction
- XML Processing Technologies
- AXIOM Features
- AXIOM Architecture
  - Deferred Building
  - Caching
  - Factories
  - Input/Output
- Using Axiom – Code examples
- AXIOM Performance
Introduction

- **AXIOM - AXis Object Model**
  - also referred to as OM (Object Model)

- XML info-set model
  - info-set: information included inside the XML

- Initially developed for Apache Axis2

- for convenient programmatical manipulation --> representation of XML infoset is in a language specific manner
  - model made up of objects
    - e.g.: DOM and JDOM
Introduction

- **OM**
  - external behavior - conceptually the same as DOM and JDOM
  - internals – different and unique
  - Based on StAX

- **History**
  - Was proposed as a store for the pull parser events for later processing
  - Started during first Axis2 f2f held in Colombo, Sri Lanka, in September 2004
Agenda

- Introduction
- XML Processing Technologies
- AXIOM Features
- AXIOM Architecture
  - Deferred Building
  - Caching
  - Factories
  - Input/Output
- Using Axiom – Code examples
- AXIOM Performance
Current Approaches for XML Processing

- Tree based approach (e.g.: DOM)
  - Complete XML in memory
- Event based approach (e.g.: SAX)
  - Can only go forward (no reverse gear !!)

Problems with the above approaches

- memory consumption
- performance
XML Processing Technologies

- Introduction of StAX – JSR 173
  - Push vs Pull parsing

```
10111101100
```

parser

push

pull

client
StAX (Streaming API for XML)

- Events
  - XMLStreamConstants.START_ELEMENT
  - XMLStreamConstants.END_ELEMENT
  - XMLStreamConstants.PROCESSING_INSTRUCTION
  - XMLStreamConstants.CHARACTERS
  - XMLStreamConstants.COMMENT
  - XMLStreamConstants.SPACE
  - XMLStreamConstants.START_DOCUMENT
  - XMLStreamConstants.END_DOCUMENT
  - XMLStreamConstants.ENTITY_REFERENCE
  - XMLStreamConstants.DTD
  - XMLStreamConstants.CDATA
StAX (Streaming API for XML)

- Valid methods for events
  - All states
    - `getProperty()`, `hasNext()`, `require()`, `close()`, `getNamespaceURI()`, `isStartElement()`, `isEndElement()`, `isCharacters()`, `isWhiteSpace()`, `getNamespaceContext()`, `getEventType()`, `getLocation()`, `hasText()`, `hasName()`
  - START_ELEMENT
    - `next()`, `getName()`, `getLocalName()`, `hasName()`, `getPrefix()`, `getAttributeXXX()`, `isAttributeSpecified()`, `getNamespaceXXX()`, `getElementText()`, `nextTag()`
  - CHARACTERS
    - `next()`, `getTextXXX()`, `nextTag()`
Advantages of StAX

- Client gets the control of this parsing model and parsing happens according to client requirements
- Pull parsing libraries are much smaller,
- Filtering of elements is easier as the client knows that when a particular element comes in
Agenda

- Introduction
- XML Processing Technologies
- AXIOM Features
- AXIOM Architecture
  - Deferred Building
  - Caching
  - Factories
  - Input/Output
- Using Axiom – Code examples
- AXIOM Performance
AXIOM

- AXIOM – Fully XML info-set compliant object model, based on StAX
- API is like a simplified DOM
- Deferred building support
  - “Lazy Parsing”
- In-built support to serialize/deserialize binary data
- Provides StAX events from any place of the tree
MTOM Support

- 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
MTOM - 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 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
MTOM - Example

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.createTextNode(dataHandler, true);
data.addChild(textData);

Explicitly need to enable MTOM support in axis2.xml
Agenda

- Introduction
- XML Processing Technologies
- AXIOM Features
- AXIOM Architecture
  - Deferred Building
  - Caching
  - Factories
  - Input/Output
- Using Axiom – Code examples
- AXIOM Performance
STAXOMBuilder.getDocumentElement()

API Read (tree)

OMElement.getChildren() etc

OElement.serialize()

SAXOMBuilder.getDocumentElement()

API Write (tree)

OMFactory.createOMElement()

OMElement.getXMLStreamReader()
AXIOM Architecture

OM Architecture

Builder Interface

OM API

StAX Stream Reader

Raw XML Stream

1010001010101010

User
AXIOM Architecture

- **Builders**
  - reads from a parser, builds the OM model
  - Three in-built builders
    - StAXOMBuilder – builds an AXIOM tree
    - StAXSOAPModelBuilder – can read SOAP messages and builds SOAP object model based on AXIOM
    - MTOMStAXSOAPModelBuilder – can read MTOM messages
AXIOM Architecture

Deferred Building

- Partial building of object model in memory
- Model is built on-demand and only up to the extent information is required.
- Saves memory, without compromising performance.
- Transparent to the user
AXIOM Architecture
Deferred Building

```xml
<soap:Envelope>
  <soap:Header>
    <myNS:Security soap:mustUnderstand="true">
    </myNS:Security>
  </soap:Header>
  <soap:Body>
    <doSomethingCool>
      ... MEGABYTES OF DATA HERE ... 
    </doSomethingCool>
  </soap:Body>
</soap:Envelope>
```

Build object model to here

```java
h = envelope.getHeader(securityQName)
```
AXIOM Architecture

Caching

- Information can be accessed with or without building the object model
  - Caching

- Enables intelligent and controlled use of memory
  - example: sending messages doesn't require to build the object model
AXIOM Architecture
Concept of Factories

- AXIOM API can be implemented using different models.
  - Linked list model
  - Tree model (as in Xalan) - discontinued
  - DOM based model – this gives DOM interface support on top of AXIOM
AXIOM Architecture
Concept of Factories

OM API

OM Factory

Linked List Based Implementation

any other implementation
AXIOM Architecture

Concept of Factories

- AXIOM comes in different jars
  - axiom-api jar
  - axiom-impl jar – this has the default linked list implementation
  - axiom-dom jar – the DOM implementation of AXIOM interfaces
Agenda

- Introduction
- XML Processing Technologies
- AXIOM Features
- AXIOM Architecture
  - Deferred Building
  - Caching
  - Factories
  - Input/Output
- Using Axiom – Code examples
- AXIOM Performance
Getting Your Feet Wet With Code!!

- Download AXIOM binaries
  - http://ws.apache.org/commons/axiom/download.cgi
  - Get axiom-impl-1.0.jar and axiom-api-1.0.jar
Sample XML

```xml
<Presentation>
    <Name>AXIOM</Name>
    <Presenter xmlns:ns1="http://some-uri.org">
        <ns1:Name>Eran Chinthaka</ns1:Name>
        <ns1:Company>WSO2</ns1:Company>
    </Presenter>
</Presentation>
```
Using AXIOM

Example:

- Read an XML File in to AXIOM and print it our to System.out

```java
StAXOMBuilder builder = new StAXOMBuilder(new FileInputStream(xmlFile));
OMElement documentElement = builder.getDocumentElement();
System.out.println("xml = "+
documentElement.toStringWithConsume());
```
Sample

<Presentation>
  <Name>AXIOM</Name>
  <Presenter xmlns:ns1="http://some-uri.org">
    <ns1:Name>Eran Chinthaka</ns1:Name>
    <ns1:Company>WSO2</ns1:Company>
  </Presenter>
</Presentation>
StAXOMBuilder builder = new StAXOMBuilder(new FileInputStream(xmlFile));
OMElement documentElement = builder.getDocumentElement();
System.out.println("xml = " + documentElement.toStringWithConsume());

Create an OM Builder from the given FileInputStream
Using AXIOM (Continued)

StAXOMBuilder builder = new StAXOMBuilder(new FileInputStream(xmlFile));
OMElement documentElement = builder.getDocumentElement();
System.out.println("xml = " + documentElement.toStringWithConsume());

Get the document element from the builder
Using AXIOM (Continued)

```java
StAXOMBuilder builder = new StAXOMBuilder(new FileInputStream(xmlFile));
OMElement documentElement = builder.getDocumentElement();
System.out.println("xml = " +
        documentElement.toStringWithConsume());
```

Print the document element. `toStringWithConsume()` method will print the xml in to the out stream **without** building an OM tree
Using AXIOM (Continued)

```java
StAXOMBuilder builder = new StAXOMBuilder(new FileInputStream(xmlFile));
OMElement documentElement = builder.getDocumentElement();

System.out.println("xml = " + documentElement);
```

Print the document element. This time the XML will be printed to the output stream whilst building an OM tree.
OMFactory omFactory = OMAbstractFactory.getOMFactory();
Creating Axiom From Scratch

```xml
<Presentation>
  <Name>AXIOM</Name>
  <Presenter xmlns:ns1="http://some-uri.org">
    <ns1:Name>Eran Chinthaka</ns1:Name>
    <ns1:Company>WSO2</ns1:Company>
  </Presenter>
</Presentation>

OMFactory omFactory = OMAbstractFactory.getOMFactory();

OMElement presentation = omFactory.createOMElement("Presentation",
null);
```
Creating Axiom From Scratch

```xml
<Presentation>
  <Name>AXIOM</Name>
  <Presenter xmlns:ns1="http://some-uri.org">
    <ns1:Name>Eran Chinthaka</ns1:Name>
    <ns1:Company>WSO2</ns1:Company>
  </Presenter>
</Presentation>
```

```java
OMFactory omFactory = OMAbstractFactory.getOMFactory();

OMElement presentation = omFactory.createOMElement("Presentation", null);

omFactory.createOMElement("Name", null, presentation).setText("AXIOM");
```
<Presentation>
   <Name>AXIOM</Name>
   <Presenter xmlns:ns1="http://some-uri.org">
      <ns1:Name>Eran Chinthaka</ns1:Name>
      <ns1:Company>WSO2</ns1:Company>
   </Presenter>
</Presentation>

OMFactory omFactory = OMAbstractFactory.getOMFactory();

OMElement presentation =
   omFactory.createOMElement("Presentation", null);
omFactory.createOMElement("Name", null, presentation).setText("AXIOM");

OMElement presenter =
   omFactory.createOMElement("Presenter", null, presentation);
OMNamespace ns1 =
   presenter.declareNamespace("http://some-uri.org", "ns1");

Declaring namespaces
Creating Axiom From Scratch

```xml
<Presentation>
  <Name>AXIOM</Name>
  <Namespace xmlns:ns1="http://some-uri.org">
    <ns1:Name>Eran Chinthaka</ns1:Name>
    <ns1:Company>WSO2</ns1:Company>
  </Namespace>
</Presentation>
```

```java
OMFactory omFactory = OMAbstractFactory.getOMFactory();

OMElement presentation = omFactory.createOMElement("Presentation", null);
omFactory.createOMElement("Name", null, presentation).setText("AXIOM");

OMElement presenter = omFactory.createOMElement("Presenter", null, presentation);
OMNamespace ns1 = presenter.declareNamespace("http://some-uri.org", "ns1");

OMElement presenterName = omFactory.createOMElement("Name", ns1, presentation);
presenterName.setText("Eran Chinthaka");
```
Creating Axiom From Scratch

```xml
<Presentation>
  <Name>AXIOM</Name>
  <Presenter xmlns:ns1="http://some-uri.org">
    <ns1:Name>Eran Chinthaka</ns1:Name>
    <ns1:Company>WSO2</ns1:Company>
  </Presenter>
</Presentation>
```

Company element

Create OMText as a child node

```
OMElement presenterCompany = omFactory.createOMElement("Company", ns1, presentation);
OMText omText = omFactory.createOMText("WSO2");
presenterCompany.addChild(omText);
```
Iterator allChildren =
    presentationElement.getChildren();

while (allChildren.hasNext()) {
    OMNode omNode = (OMNode) allChildren.next();
    omNode.serialize(System.out);
    System.out.println);
}

System.out.println(""");
Navigating Object Model  (Continued)

Iterator allChildren =
    presentationElement.getChildren();

while (allChildren.hasNext()) {
    OMNode omNode = (OMNode) allChildren.next();
    omNode.serialize(System.out);
    System.out.println();
}

Provides an iterator API to navigate children. This enables to differ building of object model...
Navigating Object Model (Continued)

```
Iterator allChildren = 
presentationElement.getChildren();

while (allChildren.hasNext()) {
    OMNode omNode = (OMNode) allChildren.next();
    omNode.serialize(System.out);
    System.out.println('');
}
```

All AXIOM objects (OMElement, OMTText, etc.,) extends from OMNode
Navigating Object Model (Continued)

- Searching for a specific element

```java
OMElement presenter =
    presentationElement.getFirstChildWithName(
        new QName("Presenter"));

System.out.println("presenter = " + presenter);
```

Use of QNames are always encouraged
Getting Events From Objects

- AXIOM has the ability to get StAX events from any element.
- User need not worry whether the element is fully built or not
- Can provide events with or without building the object model
- Switching back to parser happens transparent to the user
Getting Events From Objects

```xml
<soap:Envelope>
  <soap:Header>
    <myNS:Security soap:mustUnderstand="true">
    </myNS:Security>
  </soap:Header>
  <soap:Body>
    <doSomethingCool>
      ... MEGABYTES OF DATA HERE ...
    </doSomethingCool>
  </soap:Body>
</soap:Envelope>
```

Object model has already been built up-to here

```java
h = envelope.getHeader();
```

...and then you can do

```java
body = envelope.getBody();
reader = body.getXMLStreamReader();
while (reader.hasNext()) {
  ...
}
```
Getting Events From Objects

// get an XMLStreamReader from the presentation element with caching
XMLStreamReader xmlStreamReader = presentationElement.getXMLStreamReader();

// get an XMLStreamReader from the presentation element without caching
XMLStreamReader xmlStreamReaderWithoutCaching = presentationElement.getXMLStreamReaderWithoutCaching();
AXIOM with Databinding

XML, binary XML  →  SAX or StAX  →  Data binding framework (JAXB, XMLBeans, and others)
AXIOM and XPATH

- AXIOM has in-built support for Xpath
- Implemented using Jaxen

`XPathEvaluator.evaluateXpath(String xpathExpression, Object element, String nsURI)`
AXIOM and DOM

- AXIOM has an implementation of DOM api over OM.
- Just switch to DOOM (DOM over OM) factory
- Enabled WSS4J implementation on top of AXIOM as XML Canonicalization works on DOM.
AXIOM and SOAP

- Extended OM API to support SOAP handling
- Envelope, Body, Header, Fault classes for convenience
- Makes the main object model for Axis2
- Validation is in-built using a separate builder (StAXSOAPModelBuilder)
Agenda

- Introduction
- XML Processing Technologies
- AXIOM Features
- AXIOM Architecture
  - Deferred Building
  - Caching
  - Factories
  - Input/Output
- Using Axiom – Code examples
- AXIOM Performance
AXIOM Performance

- Two different XMLs
  - sample1.xml – small in size
  - sample2.xml – larger in size

- Tested with different object models
  - AXIOM
  - jDom
  - dom4j
  - Xerces2
AXIOM Performance

- Parameters
  - Build time
  - Walking time
  - Build and Walk time
  - Write time
  - Modification time

- Benchmark
  - Sosnoski XMLBench (http://www.sosnoski.com/opensrc/xmlbench/)
AXIOM Performance (Sample2.xml)
Where Do People Use AXIOM

- Apache WS Project including Apache Axis2
- Spring
- Abdera
  - high-performance implementation of the IETF Atom Syndication Format (RFC 4287) and Atom Publishing Protocol (in-progress) specifications.
Releases......

- 1.0 released on 2\textsuperscript{nd} May, 2006...
- 0.95 released on 23\textsuperscript{rd} March, 2006
- M1, M2, 0.90, 0.91, 0.92, 0.93, 0.94 and 0.95 already released....
Releases …...

- 1.0 released on 2\textsuperscript{nd} May, 2006...
- 0.95 released on 23\textsuperscript{rd} March, 2006
- M1, M2, 0.90, 0.91, 0.92, 0.93, 0.94 and 0.95 already released....
Join with us ....

- Use AXIOM and give us feedback
- Let us know your specific requirements
- Join commons-dev@ws.apache.org
  (remember to prefix subject with [AXIOM] )
Summary

- Introduction
- XML Processing Technologies
- AXIOM Features
- AXIOM Architecture
  - Deferred Building
  - Caching
  - Factories
  - Input/Output
- Using Axiom – Code examples
- AXIOM Performance
Questions?

Visit us at

http://ws.apache.org/commons/axiom/
Thank you