Web Services Development Using Apache Axis

Asst. Prof. Dr. Kanda Runapongsa
(krunapon@kku.ac.th)
Department of Computer Engineering
Khon Kaen University

Agenda

- Web Services Stack
- Installing Apache Axis
- Creating a Simple Web Service
- Calling Apache Axis Web Services
  - Developing a Dynamic SOAP Client
  - Developing a Static SOAP Client
- Monitoring SOAP Messages
- Custom Deployment
Web Services Stack

- WSFL: Service Flow
- UDDI: Service Discovery
- UDDI: Service publication
- WSDL: Services Description
- SOAP: XML-Based Message
- HTTP(S), SMTP, FTP, RMI/IIOP, MQ Series: Network

Apache Axis

( http://ws.apache.org/axis )

- A SOAP Processing Engine
- SOAP Server and Client
- Plug into servlet engine such as Tomcat
- Open source
- Java based
### Requirements

- **J2SE SDK**
  - [http://java.sun.com/j2se/](http://java.sun.com/j2se/)
- **Servlet container (Tomcat)**
- **Apache Axis**
  - version 1.3
  - [http://ws.apache.org/axis](http://ws.apache.org/axis)

### Agenda

- **Web Services Stack**
- **Installing Apache Axis**
- **Creating a Simple Web Service**
- **Calling Apache Axis Web Services**
  - Developing a Dynamic SOAP Client
  - Developing a Static SOAP Client
- **Monitoring SOAP Messages**
- **Custom Deployment**
### Apache Axis Installation

- Install J2SDK
- Install Web Server, such as Tomcat
- Unzip and deploy Apache Axis
- Install Apache Axis

### Installation

- Export system environment
  - set AXIS_HOME=c:\apache\axis
  - set AXIS_LIB=%AXIS_HOME%\lib
  - set AXISCLASSPATH=
    %AXIS_LIB%\activation.jar;%AXIS_LIB%\axis.jar;
    %AXIS_LIB%\axis-ant.jar;
    %AXIS_LIB%\commons-discovery-0.2.jar;
    %AXIS_LIB%\commons-logging-1.0.4.jar;
    %AXIS_LIB%\jaxrpc.jar;%AXIS_LIB%\saaj.jar;
    %AXIS_LIB%\log4j-1.2.8.jar;%AXIS_LIB%\mail.jar;
    %AXIS_LIB%\wsdl4j-1.5.1.jar;%AXIS_LIB%\xalan-2.4.1.jar;
    %AXIS_LIB%\xmlsec.jar
  - set CLASSPATH=.;%AXISCLASSPATH%
Deploy Apache Axis

- Copy `webapps/axis` directory to `webapps` directory of Tomcat
- Start tomcat

```
http://localhost:8080/axis
```
Make Apache Axis Happy

- Download these following files from http://gear.kku.ac.th/~krunapon/178375/tools.html
  - activation.jar
  - mail.jar
  - xalan-2.4.1.jar
  - xmlsec.jar

- Then copy these files into directory
  - %TOMCAT_HOME%/webapps/axis/WEB-INF/lib
Creating Web Services

- Simple Web Services
  - Return simple type
  - Create .jws file (Java Web Services File)

- Advanced Web Services
  - Return simple type
  - Return Complex type
  - Deployment
  - Using Axis Tools
Agenda

- Web Services Stack
- Installing Apache Axis
- Creating a Simple Web Service
- Calling Apache Axis Web Services
  - Developing a Dynamic SOAP Client
  - Developing a Static SOAP Client
- Monitoring SOAP Messages
- Custom Deployment

Creating a Simple Web Service

- Creating Math Web Service
  - Creating Math.java
  - Renaming Math.java to Math.jws
  - Copying Math.jws to the location %TOMCAT_HOME%/webapps/axis
  - Accessing the service at http://localhost:8080/axis/Math.jws
Math Web Service (1/2)

- Create Math.java with operations
  - add
  - subtract
- Rename Math.java to Math.jws

```java
public class Math {
    public double add(double a, double b) {
        return a + b;
    }
    public double subtract(double a, double b) {
        return a - b;
    }
}
```

Math Web Service (2/2)

- Copy *.jws files under directory
  `- %TOMCAT_HOME%/webapps/axis`
Creating a Simple Web Service

- Three steps for creating a simple web service
  - Create a Java file
  - Rename .java file to .jws file
  - Copy .jws file to %TOMCAT_HOME%webapps\axis
Our First SOAP-based Web Service

- Axis automatically compiles and wraps the *.jws file as a web service when requested by an appropriate client (such as a web browser or a SOAP client).
- You can modify the *.jws file, switch to your browser, hit refresh, and the file is automatically recompiled and made available as a web service.
- The *.jws file is not actually compiled until you request either the service or the WSDL file.
- The automatically generated WSDL file shows that the Math.jws has been compiled and is now available for use by clients.

Agenda

- Web Services Stack
- Installing Apache Axis
- Creating a Simple Web Service
- Calling Apache Axis Web Services
  - Developing a Dynamic SOAP Client
  - Developing a Static SOAP Client
- Monitoring SOAP Messages
- Custom Deployment
Dynamic SOAP Client

- Information about the web service is constructed at runtime, as opposed to being precompiled.
- This runtime construction of a web service request allows for very rapid development when initially connecting to a web service.
- It uses Apache axis classes (org.apache.axis.client.Call and org.apache.axis.client.Service) and JAX-RPC classes (javax.xml.namespace.QName).

Dynamic Math WS Client (1/2)

```java
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;
public class MathClient {
    public static void main(String[] args) {
        try {
            String endpoint = "http://localhost:8080/axis/Math.jws";
            String operation = "add";
            Service service = new Service();
            Call call = (Call) service.createCall();
            call.setTargetEndpointAddress(new java.net.URL(endpoint));
            call.setOperationName(new QName(endpoint, operation));
            Double num1 = new Double(args[0]);
            Double num2 = new Double(args[1]);
            Double addResult = (Double) call.invoke(new Object[] {num1, num2});
            System.out.println(num1 + " + " + num2 + " = " + addResult.doubleValue());
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
```
Dynamic Math WS Client (2/2)

```java
import org.apache.axis.client.Call;

public class MathClient {
    public static void main(String[] args) {
        try {
            String endpoint = "http://localhost:8080/axis/Math.jws";
            String operation = args[0];
            Service service = new Service();
            Call call = (Call) service.createCall();
            call.setTargetEndpointAddress(new java.net.URL(endpoint));
            call.setOperationName(new QName(endpoint, operation));
            Double num1 = new Double(args[1]);
            Double num2 = new Double(args[2]);
            System.out.println(num1 + num2 + = 3.0);
        }
    }
}
```

Agenda

- Web Services Stack
- Installing Apache Axis
- Creating a Simple Web Service
- Calling Apache Axis Web Services
  - Developing a Dynamic SOAP Client
  - Developing a Static SOAP Client
- Monitoring SOAP Messages
- Custom Deployment
Static SOAP Client

- A static SOAP client uses a provided WSDL to generate a set of corresponding Java classes
- This makes it easier to build and maintain an application, allowing for stricter type information and less bookkeeping
- It also makes it easier for an IDE to assist you in your development (e.g., providing automatic code completion)

Generate Client Stub Files

- To generate client-side Java objects that can access the remote SOAP service we just created, run the AXIS WSDL2Java tool
  ```
  java org.apache.axis.wsdl.WSDL2Java
  -N"http://localhost:8080/axis/Math.jws"
  mathWS
  ```
- The command should be entered as a single line, and if successful there will be no visible output
Generate Client Stub Files

- mathWS the directory which contain Java files that are generated automatically

```java
import mathWS.*;
public class StaticMathClient {
    public static void main(String[] args) {
        MathService myService = new MathServiceLocator();
        try {
            mathWS.Math math = myService.getMath();
            System.out.print("The result of 2+3 is ");
            System.out.println(math.add(2,3));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```
Static Math WS Client (2/2)

```java
import mathWS.*;

gpublic class StaticMathClient
{
    public static void main(String[] args)
    {
        MathService myService = new MathServiceLocator()
        try
        {
            mathWS.Math math = myService.getMath();
            System.out.println("The result of 1+2 is ");
            System.out.println(math.add(1,2));
        }
    }
}
```

The result of 1+2 is 3.0

Agenda

- Web Services Stack
- Installing Apache Axis
- Creating a Simple Web Service
- Calling Apache Axis Web Services
  - Developing a Dynamic SOAP Client
  - Developing a Static SOAP Client
- Monitoring SOAP Messages
- Custom Deployment
Using the Axis TCP Monitor (tcpmon)

- TCP monitor allows us to see messages sent between the requester and the provider.
- The included “tcpmon” utility can be found in the org.apache.axis.utils package.
- To run it from the command line:
  ```
  %java org.apache.axis.utils.tcpmon [listenPort targetHost targetPort]
  ```
- In our case, listenPort is 8090, targetHost is localhost, and targetPort is 8080.
  ```
  Modify MathClient.java
  String endpoint = "http://localhost:8090/axis/Math.jws";
  ```
  ```
  Type command
  %java org.apache.axis.utils.tcpmon 8090 localhost 8080
  ```

SOAP Monitoring

- Using Apache TCPMonitor
  ```
  org.apache.axis.utils.tcpmon [listenport] [targethost] [targetport]
  ```
- In MathClient.java, change URL endpoint port as listenport
  ```
  http://localhost:8090/axis/Math.jws
  ```
- Type command
  ```
  start java org.apache.axis.utils.tcpmon 8090 localhost 8080
  ```
Web Services Development Using Apache Axis

Drawbacks of JWS Web Services

- JWS web services are intended for simple web services
- You need the source codes - there might be times when you want to expose a pre-existing class without source
- You cannot find out about errors until after deployment
- You cannot completely control how the service gets accessed
- You cannot specify custom type mappings
- Production quality web services should use Java classes with custom deployment
Agenda

- Web Services Stack
- Installing Apache Axis
- Creating a Simple Web Service
- Calling Apache Axis Web Services
  - Developing a Dynamic SOAP Client
  - Developing a Static SOAP Client
- Monitoring SOAP Messages
- Custom Deployment

Custom Deployment - Introducing WSDD

- To really use the flexibility available to you in Axis, you should get familiar with the Axis Web Service Deployment Descriptor (WSDD) format
- A deployment descriptor contains a bunch of things you want to “deploy” into Axis - make available to the Axis engine
- The most common thing to deploy is a Web Service
Custom Deploy Web Services

- WSDD File
- Axis Engine
- Web Services
- webapps/axis/WEB-INF/classes
- webapps/axis/WEB-INF/lib

Java Class

BahtDollarExchange Web Service

step 1: create java class
- Return simple type
- Operation
  - Dollar2Baht
  - Baht2Dollar

```java
public class BahtDollarExchange {
    public double Dollar2Baht(double dollar) {
        return dollar/40;
    }

    public double Baht2Dollar(double baht) {
        return baht*40;
    }
}
```
Step 2: Create a WSDD file

```xml
<deployment xmlns="http://xml.apache.org/axis/wsdd/
xmlns:java="http://xml.apache.org/axis/wsdd/providers/jav a">
  <service name="BahtDollarExchange" provider="java:RPC">
    <parameter name="allowedMethods" value="*" />
    <parameter name="className" value="BahtDollarExchange" />
  </service>
</deployment>
```
**WSDD File Explanation**

- `<deployment>` tells the engine that this is a WSDD deployment and `<service>` actually defines the service for us.
- Our provider is “java:RPC”, which is built into Axis, and indicates a Java RPC service.
- We tell the RPCProvider that it should instantiate and call the correct class by including `<parameter>` that configures the class name.
- Another `<parameter>` is to tell the engine that any public method (“*”) on the class may be called via SOAP.
  - (Using a space or comma separated list of available method names)

**Using the AdminClient**

- Once we have file *wsdd* file, we need to send it to an Axis server in order to actually deploy the described service.
- We do this with the AdminClient or the “org.apache.axis.client.AdminClient” class.
- A typical invocation of the AdminClient looks like this:
  ```
  %java org.apache.axis.client.AdminClient deploy.wsdd
  ```
  (Assume that *.jar of Axis are already included in your classpath and deploy.wsdd in the current working directory).
BahtDollarExchange Web Service

Step 3: Deploy Web Services

```
java
org.apache.axis.client.AdminClient
deploy.wsdd
```

Deployed Services

And now... Some Services

- AdminService (wsdl)
- Version (wsdl)
- BahtDollarExchange (wsdl)
  - Dollar2Baht
  - Baht2Dollar
Undeploy Web Service

- Write file “undeploy.wsdd”

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

- Run command to undeploy

```bash
%java org.apache.axis.client.AdminClient
  undeploy.wsdd
```

(Assume that *.jar of Axis are already included in your classpath and undeploy.wsdd in the current working directory)

References

- “Real World Web Services” written by Will Iversion, O’Reilly