Consuming, Providing & Publishing WS

Ioannis G. Baltopoulos

Department of Computer Science
Imperial College London

Inverted CERN School of Computing, 2005
Geneva, Switzerland

The Software Environment

For this tutorial we are going to use the following software environment.

- **Java**
  Producers and Consumers will be based on Java version 1.4.2.
- **Eclipse**
  THE IDE for writing Java code. Version used is 3.1M4
- **Ant**
  Build tool used for automating the development process.
- **Tomcat**
  The Web Application container hosting the WS.
- **Axis**
  An open source WS implementation for Java; currently in version 1.2RC2.

Apache Tomcat (5.0.28)
Installation and Notes

Web Site

http://jakarta.apache.org/tomcat/

Step by step installation

1. Download the required file from http://jakarta.apache.org/site/binindex.cgi#tomcat
2. Extract the downloaded file in a directory of your choice.
3. Start the server from `tomcat/bin/startup`
4. Validate installation by going to http://localhost:8080/

Web Site

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

Step by step installation

1. Download the required file from http://ws.apache.org/axis/releases.html
2. Extract the downloaded file in a directory of your choice.
3. Copy the `axis/webapps` directory to `tomcat/webapps`.
4. Restart the web server.
5. Validate installation by going to http://localhost:8080/axis/happyaxis.jsp

Apache Axis
The Purpose of the Application

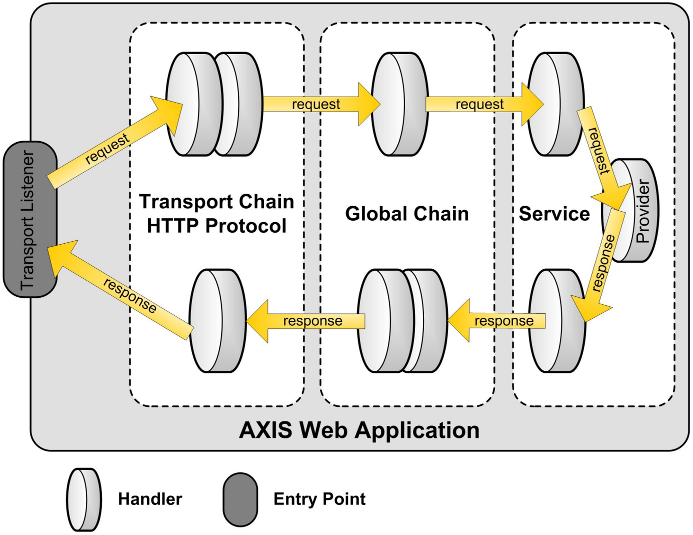
Definition

Axis is the means by which SOAP messages are taken from the transport layer and are handed to the Web Service and the means by which any response is formatted in SOAP messages and sent back to the requestor.

Apache Axis
Architectural Components

- **Axis Engine** - The main entry point into the SOAP processor
- **Handlers** - The basic building blocks inside Axis that link Axis to existing back-end systems
- **Chain** - An ordered collection of handlers
- **Transports** - Mechanisms by which SOAP messages flow in and out of Axis
- **Deployment/Configuration** - Means through which Web Services are made available through Axis
- **Serializers/Deserializers** - Code that will convert native datatypes into XML and back.

Axis Architectural Diagram



WS Consumers
The process of writing a consumer

- Locate the wsdl file for the service you're interested in.
- Use WSDL2Java to generate the stub classes.
- Writing the actual client code.

WSDL2Java
Command line and options

A tool for generating glue code in writing consumers and providers.

Command Line

```
java org.apache.axis.wsdl.WSDL2Java wsdl-file
```

Options

`-o` *directory*	Used to specify the output directory
`-p` *package*	Package specification for the output files
`-v`	Verbose output
`-t`	Generate test files
`-s`	Generate server side code

NOTE

The following files **must** be on the CLASSPATH.

axis.jar
commons-discovery.jar
commons-logging.jar
jaxrpc.jar
saaj.jar
wsdl4j.jar

Example Usage
Using a public weather web service

Capeclear offers a public weather service where given the location code of an airport ("LHR","LGW", etc) it returns a complete weather report including temperature, humidity, wind direction.

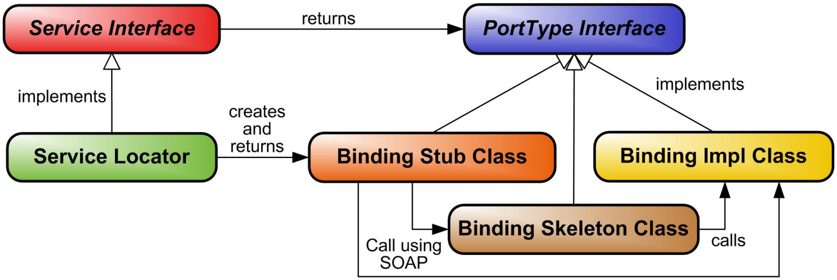
Example

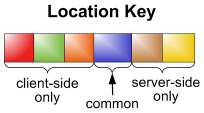
```
WSDL2Java.bat
    http://www.capeclear.com/GlobalWeather.wsdl
    -o %PROJECT_BASE%\src\java
    -p ch.cern.it.csc
    -v
```

Generated Files
What gets generated from the WSDL file

WSDL clause	Java class(es) generated
For each `<type>`	A java class. A holder if this type is used as an in-out/out parameter
For each `<portType>`	A java interface
For each `<binding>`	A stub class
For each `<service>`	A service interface. A service implementation (locator)
For each `<binding>`	A skeleton class An implementation template class
For all `<services>`	One `deploy.wsdd` file One `undeploy.wsdd` file

Generated Files
Relationship & Location of generated files





Client Code Example

Tying all the generated files together!

Example

```
import java.rmi.RemoteException;

public class Client {
    public static void main(String[] args) {
        ServiceLocator locator = new ServiceLocator();
        ServicePort service = locator.getService();
        try {
            Report report = service.getReport("Status");
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

Writing Providers
The two approaches

- Instant Deployment
  Very simple way of providing a Web Service
- Customized Deployment
  More elaborate

Instant Deployment

Step by step

1. Copy any Java source file that implements a web service into the `axis` directory
   - no special code is required
   - all public, non-static methods are exposed
   - if the class is in a package, copy it to the appropriate subdirectory
2. Change the file extension from `.java` to `.jws`
3. Place all related `.class` files under `WEB-INF/classes`
4. View the WSDL of a JWS web service using the following URL in a web browser
   `http://host:port/axis/filename.jws?wsdl`

Example
An example using Instant Deployment

A very simple banking web service. The bank allows the following four operations

- Create an Account
- Get the balance of an Account
- Withdraw a given amount from an Account
- Deposit a given amount to an Account

To implement it we will use two basic classes

- A class Account
- A BankingService class

The Account class

```
public class Account {
    private String number;
    private String owner;
    private double balance;
    public void withdraw(double amount) {
        balance -= amount;
    }
    public void deposit(double amount) {
        balance += amount;
    }
    public double getBalance() {
        return balance;
    }
}
```

The BankingService class

```java
public class BankingService {
    public void withraw(Account ac, double amount) {
        ac.withdraw(amount);
    }
    public void deposit(Account ac, double amount) {
        ac.deposit(amount);
    }
    public Account createAccount(String owner) {
        return new Account();
    }
    public double getBalance(Account ac) {
        return ac.getBalance();
    }
}
```

Limitations
The limitations of using instant deployment

The use of instant deployment is only intended for simple web services. Here are some reasons why this is so

- You cannot use packages in the pages
- As the code is compiled at run time you can not find out about errors until after deployment.
- There is limited control over the serialization/deserialization process.
- The actual source code is placed on the web server
- Sometimes the source code is not available

Using Custom Deployment
The process of creating a Web Service

Step by step

1. Write a Facade interface the subsystem you want to expose as a Web Service.
2. Create a WSDL file either manually or by using the Java2WSDL tool that comes with Axis.
3. Create Bindings using the WSDL2Java tool making sure to activate the options for emitting server side code as well as deployment descriptors.
4. Package all the files in a .jar file
5. Copy the file to the WEB-INF/lib
6. Use the AdminClient tool to deploy the Web Services to Axis.

Java2WSDL
Command line and options

A tool for generating a WSDL file from existing Java code

Command Line

```
java org.apache.axis.wsdl.Java2WSDL wsdl-file
```

Options

`-o` *`filename`*	Specifies the output filename
`-l` *`uri`*	Specifies the URI of the service
`-n` *`namespace`*	Target namespace of the wsdl
`-p` *`package namespace`*	Generate test files
`-v`	Verbose output

Generate Server Side Bindings
Using WSDL2Java

The next step in the process is generating the server side bindings and the deployment descriptors (deploy.wsdd, undeploy.wsdd).

- Run the WSDL2Java tool using the -s and -S options (see earlier slides for consumer generation).
- Discard the client specific files
- Package all the `.class` files in a `.jar` file. Use

```
jar cvf filename.jar file(s)
```

- Copy the generated file into the `WEB-INF/lib` directory.

Service Deployment

Using the AdminClient tool and the .wsdd files

Deployment Descriptor Files

- End with .wsdd (usually named deploy.wsdd and undeploy.wsdd)
- Specifies Axis components to be deployed or undeployed
- Specifies special type mappings between XML and Java

Command Line

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

Options

`-h` *host*	Specifies the host
`-p` *port*	Specifies the port
`-s` *servletPath*	Sets the path to the Axis Servlet

UDDI Overview

Universal Description, Discovery and Integration (UDDI)

Definition

UDDI is a specification for creating distributed Web-based registries of Web services. It defines

- A UDDI **registry** which stores information on businesses, the services offered by these businesses, and technical information about these services.
- The **data model** and programming API that provides a way to publish and locate all kinds of services.

Specifically, UDDI is said to support three kinds of registry data

- **White Pages** (organizing businesses by name)
- **Yellow Pages** (organizing businesses by category)
- **Green Pages** (organizing businesses by service)

The Colored Papers
White, yellow and green pages

White Pages

They contain information on a business itself, including

- A name,
- Contact details
- Location of the business
- Unique identifiers

Yellow Pages

Yellow pages contain categorized information about the services provided by a business.

- Categorization is done by assigning one or more taxonomies to the business.

Green Pages

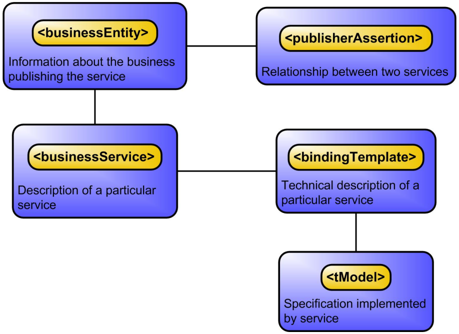
Green pages contain technical information about a service which a business offers. You can find information like

- Service location
- the category to which this service belongs

UDDI Data structures
Specifying entries in the Registry

UDDI defines five data type structures to specify an entry in the registry. Each of these data structures is represented by an XML document, containing both technical and descriptive information. These are:

- `<businessEntity>`
- `<businessService>`
- `<bindingTemplate>`
- `<tModel>`
- `<publisherAssertion>`


<businessEntity>
Information about the business publishing the service
<publisherAssertion>
Relationship between two services
<businessService>
Description of a particular service
<bindingTemplate>
Technical description of a particular service
<tModel>
Specification implemented by service

Data Structure Details I

`<businessEntity>`

The businessEntity structure contains all descriptive information about the business and the services it offers. Information includes name and description of the business as well as contact information, categorization, and relationships to other businesses. This structure can be seen as the top-level structure of the service in the registry.

`<businessService>`

Each businessEntity structure contains one or more businessService structures. A businessService structure describes a categorized set of services a business offers. A businessService element is not owned by one businessEntity element, but can be shared among multiple businesses.

Data Structure Details II

<bindingTemplate>

The bindingTemplate structure contains a technical description of a service. Each bindingTemplate belongs to a single businessService element.

<tModel>

One of the key elements of UDDI is the tModel. A tModel describes the specification, the behavior, the concept, or even the shared design to which a service complies. It provides specific information about how to interact with this service. The content of a tModel structure consists of a key, a name, an optional description, and a URL element. The URL, in most cases, points to a location where you can find more information about this particular tModel. Two conventions have been applied for using tModels.

Data Structure Details III

<publisherAssertion>

The publisherAssertion structure contains information about a relationship between two parties asserted by one or both. Many businesses, such as large corporations or marketplaces, are not effectively represented by a single businessEntity. A publisherAssertion can be used to denote the relationship between the businesses. The content of a publisherAssertion structure consists of a key (fromKey) for the first business, a key (toKey) of the second business, and a reference (keyedReference) that designates the asserted relationship in terms of a keyName, keyValue pair within a tModel.

Publishing Services on UDDI

The manual way if doing things

Step by step installation

1. Logon to `http://www.uddi.org/`
2. Select a registry from IBM, Microsoft, SAP or NTT
3. Obtain login and password
4. Follow the step by step instructions on the website

Concluding Remarks

In this lecture we saw

- the software environment for developing and deploying Web Services in Java
- how to write Web Service consumers
- how to write Web Service providers using instant and custom deployment deployment.
- what UDDI is and how to manually publish Web Services to the Registry.