

Invoking Web Services

with Axis

Web Languages Course 2009
University of Trento

Lab Objective

- *Refresh* the Axis Functionalities
- Invoke Web Services (*client-side*)



Lab Outline

- WS Sum Up: web service protocols
- Axis Functionalities
- Web Services Invocation
 - Dynamic Interface Invocation (DII)
 - Stub Generation (WSDL2Java tools)



Web Service Protocols

Brief overview

Web Services Protocols

- **REST** (Representational State Transfer): uses plain old HTTP to make method calls and you get XML back.
- **XML-RPC**: is a remote procedure call protocol which uses XML for marking up both requests and responses.
- **SOAP** (*Simple Object Access Protocol*): is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.



WS protocol stack

Discovery

UDDI

Description

WSDL

XML messaging

XML-RPC, SOAP, XML

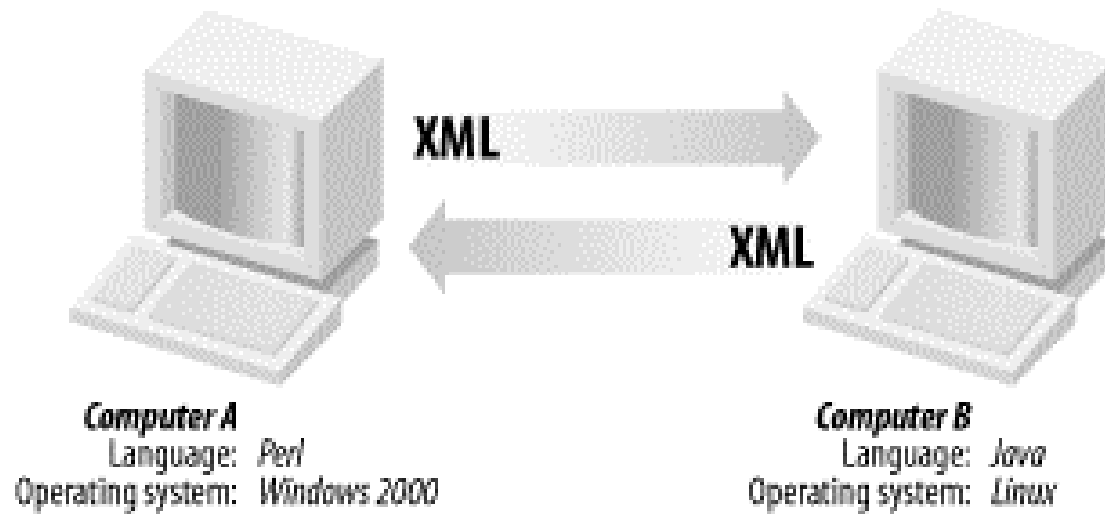
Transport

HTTP, SMTP, FTP, BEEP



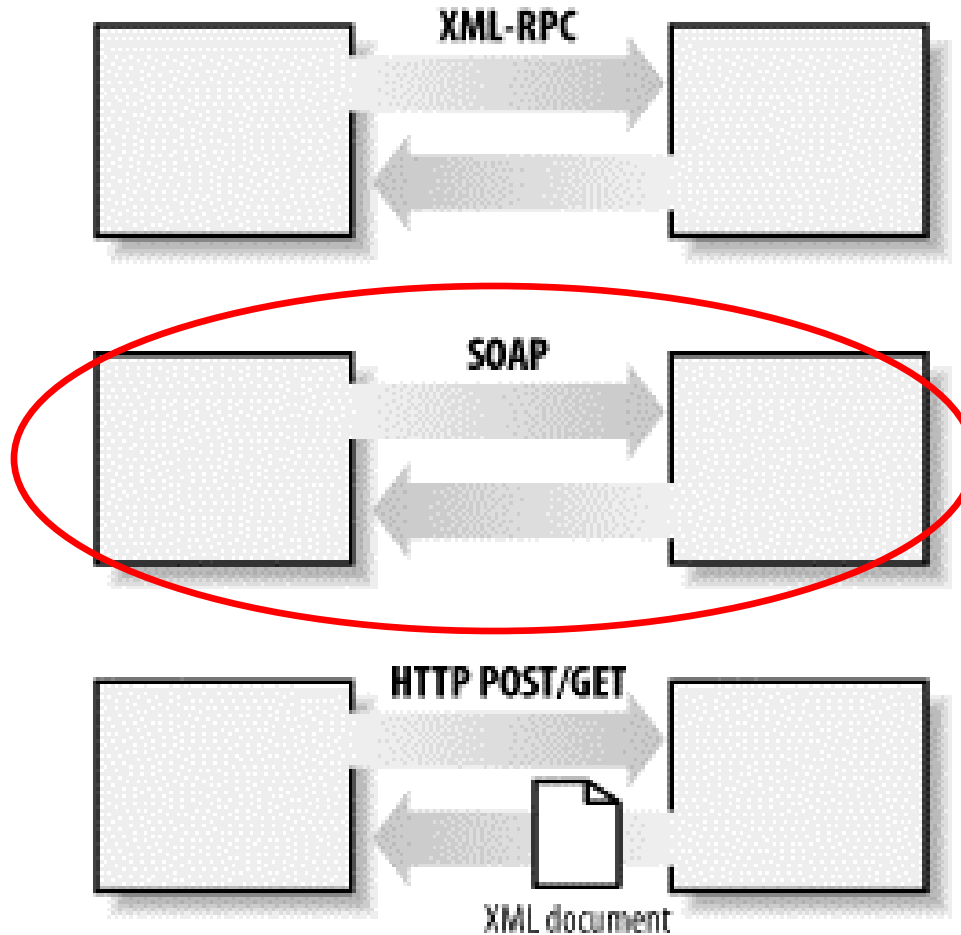
XML- messaging

1



XML- messaging

2



“Big” Web Service

- Is *available* over the Internet or private (intranet) networks
- Uses a *standardized XML messaging* system
- Is *not tied* to any one operating system or programming language
- Is *self-describing* via a common XML grammar (WSDL)
- Is *discoverable* via a simple find mechanism (UDDI)



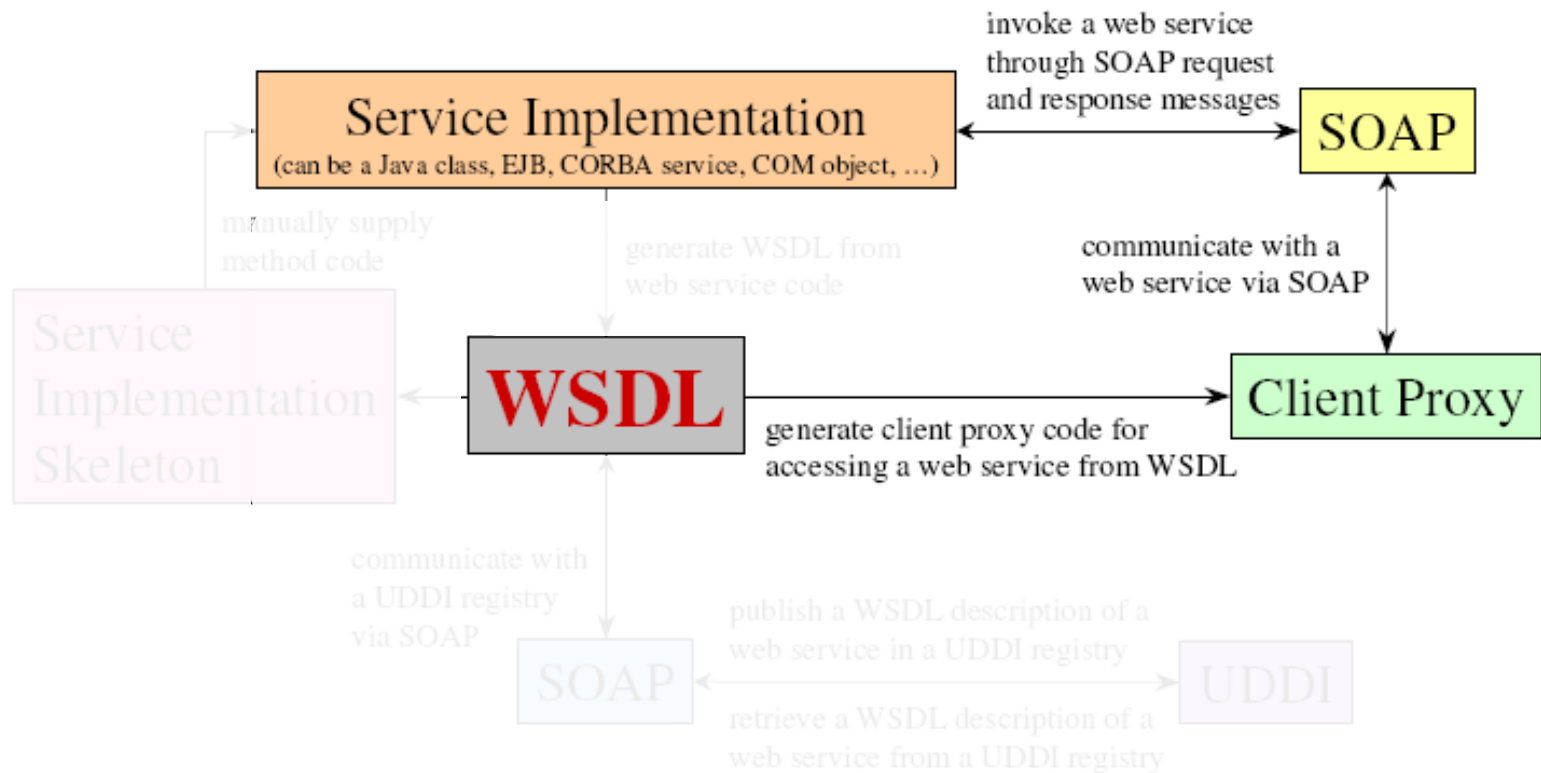
WS architecture



Axis Functionalities

“Refresh”

Toolkit Functionality for *Today Class*



“Big” Web Services Invocation

- Dynamic Invocation Interface (DII)
- Stub Generation (WSDL2Java tools)
- Exercises

Available Web Services @:

- WebServiceX
 - <http://www.websvcx.net/WCF/default.aspx>
- X-Methods
 - <http://www.xmethods.net/ve2/Directory.po>



How to invoke Web Services

- Two Options:
 - Dynamic Invocation Interface (DII)
 - Stub Generation from a Service WSDL description



Dynamic Invocation

1

- Uses Axis implementation of JAX-RPC
 - org.apache.axis.client.**Call** implements javax.xml.rpc.**Call**
 - org.apache.axis.client.**Service** implements javax.xml.rpc.**Service**
- Info Needed
 - SOAP router URL
 - also called the *target endpoint address*
 - **Service namespace**
 - required so it can be specified in Axis-generated SOAP requests
 - **Operation name**
 - Operation parameter types
 - Operation return type

get this information from a WSDL description of the service

Depending on the service, optional



Dynamic Invocation

2

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class TestTempConverter {

    public static void main(String [] args) {
        //The values of the following constants were obtained from the WSDL
        //contained in the source code (file under Lab6/wsdl)

        final String NAMESPACE = "urn:TempConverterIntf-ITempConverter";
        final String OPERATION = "CtoF";
        final String endpoint = "http://developerdays.com/cgi-
bin/tempconverter.exe/soap/ITempConverter";
```



Dynamic Invocation

3

```
try {
    Service service = new Service();
    Call call = (Call) service.createCall();

    call.setTargetEndpointAddress( new java.net.URL(endpoint) );
    call.setOperationName( new QName( NAMESPACE, OPERATION) );

    call.addParameter( "celsius",
                      org.apache.axis.Constants.XSD_INT,
                      javax.xml.rpc.ParameterMode.IN );

    call.setReturnType( org.apache.axis.Constants.XSD_INT );
    int celsius = 0;

    Object ret = call.invoke( new Object[] { celsius } );
    System.out.println( "Sent '" + celsius + "' Celsius, got '" + ret + "'
    Fahrenheit!");
} catch (Exception e) { System.err.println( e.toString() ); }
```

JAX-RPC objects

Parameters are named `arg0`, `arg1`, ... by default. This allows the client to give them names and data types the service expects.

optional

Some services don't automatically encode the return type. This service does. If not, this is how the client can set it.



Generated HTTP SOAP Request

```
POST /cgi-bin/tempconverter.exe/wsdl/ITempConverter HTTP/1.0
Content-Length: 518
Host: developerdays.com
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<ns1:CtoF xmlns:ns1=" urn:TempConverterIntf-ITempConverter">
<celsius xsi:type="xsd:int">0</celsius>
</ns1:CtoF>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



Generated HTTP SOAP Response

HTTP/1.0 200 OK

Date: Sat, 23 Mar 2002 13:25:59 GMT

Content-Length: 465

Content-Type: text/xml; charset=utf-8

Status: 200

Servlet-Engine: Lutris Enhydra Application Server/3.5.2
(JSP 1.1; Servlet 2.2; Java 1.3.0; Linux 2.4.7-10smp x86;
java.vendor=IBM Corporation)

Set-Cookie: JSESSIONID=cdDkeXGxW5F4cbgFYEjYHnOl; Path=/soap

Server: Enhydra-MultiServer/3.5.2

Via: 1.0 C6100-2 (NetCache NetApp/5.1R2D20)

HTTP headers



Generated HTTP SOAP Response

HTTP content

```
<?xml version='1.0' encoding='UTF-8' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:CtoFResponse
xmlns:ns1=" urn:TempConverterIntf-ITempConverter "
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:int">32</return>
</ns1:CtoFResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



Client Stubs

1

- Represent the service on the client-side
 - a.k.a. *proxies*
- Generated from WSDL using WSDL2Java
 - WSDL can be accessed locally or from a remote URL
 - if Java service implementation already exists, WSDL can be generated using Java2WSDL



Client Stubs

2

- Provide type-safe invocation of web services
 - Rather than passing parameters as an array of Objects, as in DII, calls are made using an interface that specifies parameter types
 - Rather than receiving an Object result, the interface specifies the actual result type
 - Types are checked at compile-time

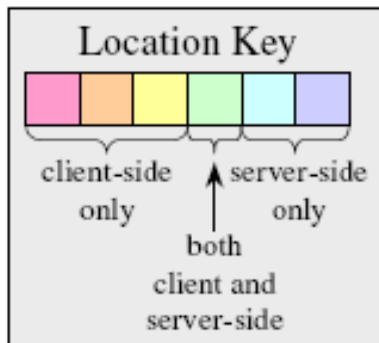
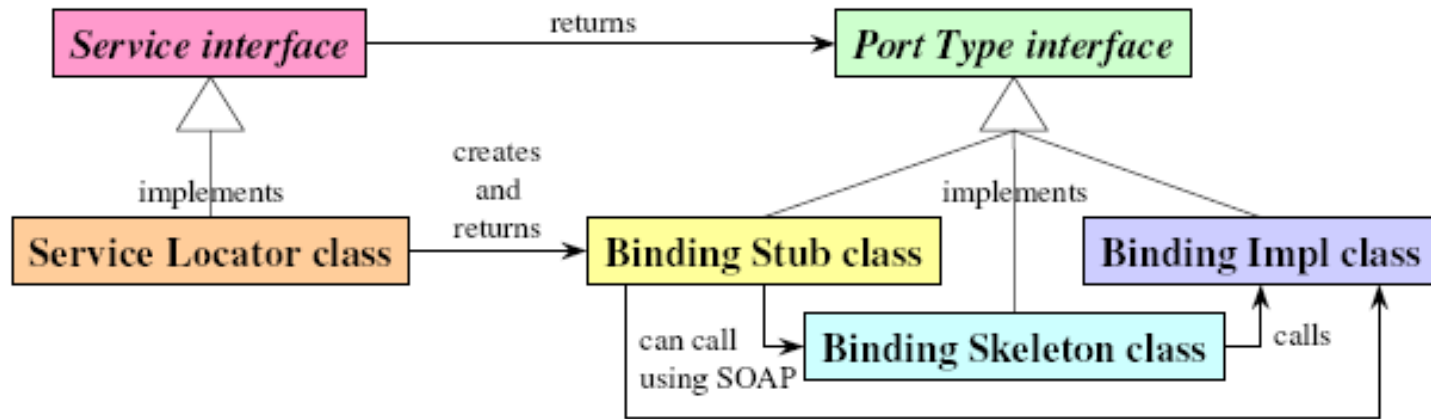


Generate Stubs: WSDL2Java

- Stub classes from a Java interface/class *:
 1. *.java: New interface file with appropriate remote usages.
 2. *Service.java: client side service interface.
 3. *ServiceLocator.java: client side service implementation factory.
 4. *SoapBindingStub.java: Client side stub.
 5. (data types): Java files produced for non-standard types.
 6. *SoapBindingImpl.java: default server implementation WS, modify manually to actual implementation.
 7. *SoapBindingSkeleton.java: Server side skeleton.
- Deploy.wsdd / undeploy.wsdd: (Un-)deployment descriptors



Relationships Between Generated Files



- Steps to use in client code**
- 1) create instance of Service Locator (hold in variable of Service interface type)
 - 2) use it to obtain a Binding Stub (hold in variable of Port Type interface type)
 - 3) invoke web service methods on it



WSDL2Java Tool

WSDL clause	Java class(es) generated
For each entry in the type section	A java class
	A holder if this type is used as an in/out parameter
For each portType	A port-type interface
For each binding	A stub class
For each service	A service interface
	A service implementation (the locator)



WSDL2Java Generates

1

- For each **<type>**
 - Class representation called *type-name.java*
 - Holder class called *type-nameHolder.java*, if it is used as an inout or out parameter
- For each **<porttype>**
 - Interface called *port-name.java*
 - describes porttype operations
 - called the Service Definition Interface (SDI)

WARNING: Watch out for WSDL2Java
overwriting existing source files with
type-name.java and port-name.java files
if you generated the WSDL using Java2WSDL!



WSDL2Java Generates

2

- For each **<binding>**
 - Stub class called ***port-nameBindingStub.java***
 - implements *port-name.java*
 - used by clients to invoke web service methods
 - uses JAX-RPC Service and Call interfaces
 - Implementation class shell called ***port-nameSoapBindingImpl.java***
 - implements *port-name.java*
 - web service method implementations to be completed by developer
 - won't be overwritten if it already exists

1) Only has "SOAP" in the name when the transport is SOAP
2) No needed when developing the client



WSDL2Java Generates

3

- don't need this if a class implementing the operations already exists
- Optionally, a skeleton class called ***port-nameSoapBindingSkeleton.java***
 - implements *port-name.java*
 - server-side counterpart to *port-nameServiceBindingStub*
 - forwards all calls to service methods to implementation methods
 - benefit is questionable; calls can go directly to your implementation instead

1) Only has "SOAP" in the name when the transport is SOAP
2) No needed when developing the client



WSDL2Java Generates

4

- For each `<service>`
 - interface called *port-nameService.java*
 - describes methods for obtaining porttype interface implementations
 - one get method for each port defined in the service
 - Locator class called *port-nameServiceLocator.java*
 - implements *port-nameService.java* (above)
 - locates an implementation of the service using the URI specified in the WSDL or a specified URI
 - creates an instance of the binding stub class and returns it as the porttype interface type



WSDL2Java Generates

5

- Optionally, a JUnit test class called *port-nameServiceTestCase.java*
 - for testing methods in the porttype interface
 - test methods must be completed manually



WSDL2Java Generates

6

- For each **WSDL file**
 - Deployment descriptor called **deploy.wsdd**
 - used by AdminClient to deploy the service
 - implementation class referred to will either be the skeleton class (if skeleton generated) or the impl. class (if no skeleton generated)
 - can change to refer to your own class
 - Undeployment descriptor called **undeploy.wsdd**
 - used by AdminClient to undeploy the service



Using WSDL2Java

- **Command**

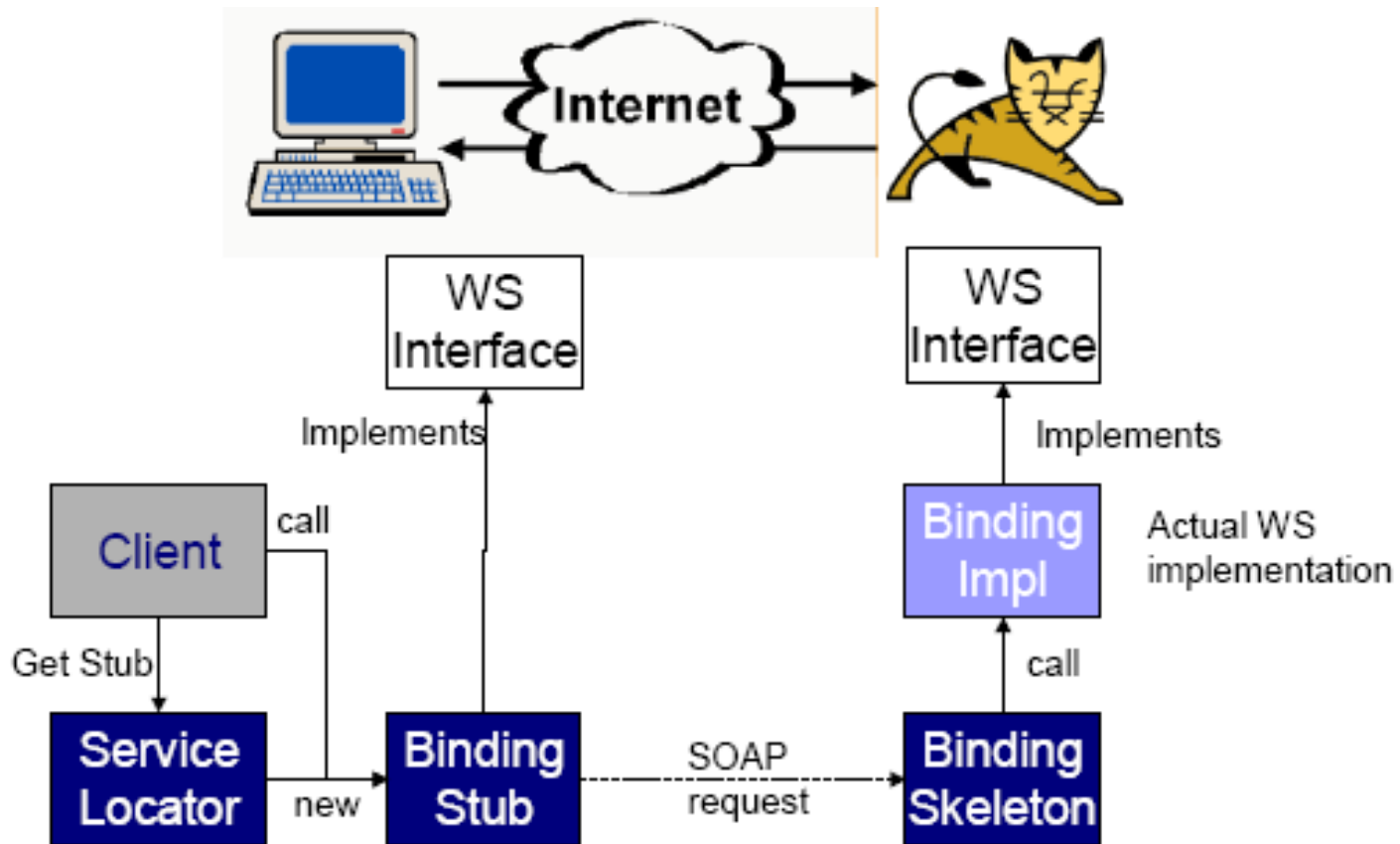
- `java org.apache.axis.wsdl.WSDL2Java wSDL-uri`

- **Option highlights**

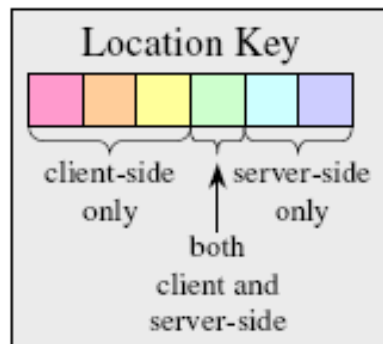
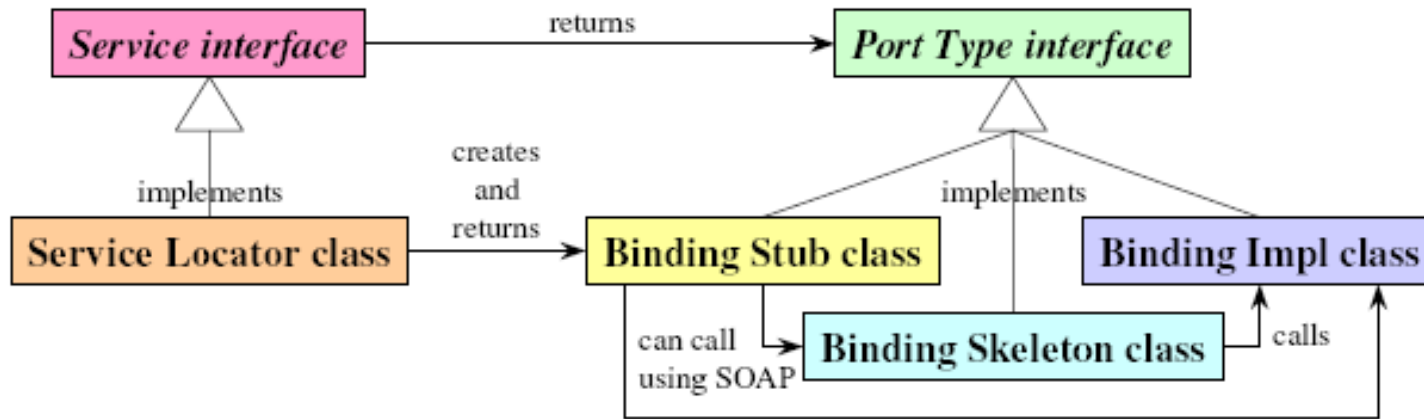
- `-Nnamespace=package` : *maps a **namespace** to a Java package*
- `-o dir` : *specifies **output** directory where generated files should be written*
- `-p pkg` : *overrides default **package** name for generated classes*
- `-s` : *generates **service-side** interfaces and classes required by a service implementation, including a skeleton class*
- `-t` : *generates **JUnit test** case for the web service*
- `-v` : *prints a message about each file that is generated*



Relationships Between Generated Files



Relationships Between Generated Files



Steps to use in client code

- 1) create instance of Service Locator (hold in variable of Service interface type)
- 2) use it to obtain a Binding Stub (hold in variable of Port Type interface type)
- 3) invoke web service methods on it

This is done in `TestTempConverter`



Client Stub Invocation

```
package ws.client.stub;
import NET.webserviceX.www.*;
public class TestTempConverter {
public static void main(String [] args)
throws java.rmi.RemoteException, javax.xml.rpc.ServiceException {

    ConvertTemperature service = new ConvertTemperatureLocator();

    ConvertTemperatureSoap stub = service.getConvertTemperatureSoap();
    int celsius = 30;
    int fahrenheit = stub.convertTemp(30, TemperatureUnit.degreeCelsius,
    TemperatureUnit.degreeFahrenheit);
    System.out.println("Sent '" + celsius + "' Celsius, got '" + far +
    "' Fahrenheit!");
}
}
```

package of generated files

Notice that parameters are not passed to the service in an array of Objects and the proper return type is returned instead of an Object.



Env Variable Settings - AXIS

- **AXIS_HOME**

- %CATALINA_HOME%\webapps\axis

Tomcat home dir

- **AXIS_LIB**

- %AXIS_HOME%\WEB-INF\lib

Or jaxp-1.3.jar

- **AXIS_CP**

- .;%AXIS_LIB%\xercesImpl.jar;%AXIS_LIB%\activation.jar;
%AXIS_LIB%\wsdl4j-1.5.1.jar;%AXIS_LIB%\log4j-1.2.8.jar;
%AXIS_LIB%\commons-codec-1.3.jar;%AXIS_LIB%\axis.jar;
%AXIS_LIB%\commons-discovery-0.2.jar; %AXIS_LIB%\mail.jar;
%AXIS_LIB%\commons-logging-1.0.4.jar; %AXIS_LIB%\saaj.jar;
%AXIS_LIB%\jaxrpc.jar;%AXIS_LIB%\xml-apis.jar;
%AXIS_LIB%\servlet-api.jar; %AXIS_LIB%\axis-ant.jar



Exercises

Generating WS proxies and
Developing client applications

Exercise 1

- Run the source code
 - Create a new project using the source code in Lab6
 - Modify file `stub.properties` according to your settings
 - Go to:
 - <http://www.webservices.net/WCF/ServiceDetails.aspx?SID=61>
 - Save the relative wsdl file in the directory `wsdl`
 - Run the ant task “all” of the file `build.xml` in order to create the stubs and include the `stub.jar` in your project
 - Run the file “TestConvertTemperature” in package `ws.client.stub`



Build.xml: *axis-wsd2java*

```
<!--  
    =====  
    WSDL2JAVA SECTION  
    =====  
-->  
<target name="wsdl2java" depends="init">  
  <mkdir dir="${src.stub}" />  
  <axis-wsd2java output="${src.stub}" testcase="false" verbose="true"  
    url="${local.wsdl}">  
    <!--  
      <mapping  
        namespace= "http://www.webservices.net"  
        package="webservices.lenght" />  
    -->  
  </axis-wsd2java>  
</target>
```

targetNamespace
In WSDL file



Exercise 2

- Following the schema of exercise1, test the services at:
 - <http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=71> (Length/Distance Unit)
 - <http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=47> (Translation Engine)

Create your own package (use the “mapping” element in build.xml)



Exercise 3

- AddFunction code
 - Generate stubs classes for the AddFunction code
 - Run a client using stub generation



Exercise 4:

The USA Weather Forecast

- Access another Web Service in **Xmethods.net**
 - Choose the Web service at:
<http://www.xmethods.net/ve2/ViewListing.po?key=uid:DC12A48B-1A20-36B1-4AB5-9D7EEF18193E>
 - Analyse WSDL
 - Save the WSDL file in your wsdl directory
 - Generate the stubs
 - Build a client which print info about weather forecast



Exercise 4:

Expected Output

```
PLACE NAME: NEW YORK
LAT: 40.74838
LONG: 73.996704
STATE CODE: NY
DAY 0: Sunday, March 16, 2008
MAX TEMP: 7
MIN TEMP: 3
...
DAY 6: Saturday, March 22, 2008
MAX TEMP: 9
MIN TEMP: 2
```



References

- Axis User Guide
 - Provided in the course website
- RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision:
 - <http://www2008.org/papers/pdf/p805-pautassoA.pdf>
- W3C’s Web Services Activity Home Page:
<http://www.w3.org/2002/ws/>



Troubleshooting



Ant Problems

1

- Eclipse seems to recognize an Ant file (i.e., build.xml) but cannot actually run its targets...
- It may be a problem of Eclipse version, ant version, how Eclipse was installed.
- Actions
 - If you have Ubuntu, avoid the Eclipse packages provided by Ubuntu. Install Eclipse by downloading it from the official website
 - Run the ant file from the command line
 - Tell eclipse to use an external ant installation and see what happens (last option...so far)



Ant Problems

2

- It doesn't calculate the correct path according to the given settings
 - Example: you fixed "." as a *basedir* property in your build.xml file and some files (i.e, the wsdl file), indicated together with a relative path (i.e, wsdl) in a *.properties* file (i.e, stub.properties), are not accessible.
- It may be a problem of Ant version (?)
- Action
 - put the absolute path for the file in question (i.e., wsdl file) → **file:///C://blabla//file.wsdl**



WSDL Problems

- Make sure you save the WSDL file in the correct way
 - DON'T COPY AND PAST THE CONTENT OF THE WEB PAGE
 - **SAVE** THE FILE AS AN XML FILE AND CHANGE THE EXTENSION TO WSDL



Axis Problems

- Make sure the 14 axis libraries (*.jar) are placed under **%TOMCAT_HOME%/webapps/axis/WEB-INF/lib**
- List of the libraries:
 - activation.jar
 - axis-ant.jar
 - axis.jar
 - commons-discovery-0.2.jar
 - commons-logging-1.0.4.jar
 - jaxp-1.3.jar
 - jaxrpc.jar
 - log4j-1.2.8.jar
 - mail.jar
 - saaj.jar
 - servlet-api.jar
 - wSDL4j-1.5.1.jar
 - xml-apis.jar, tool.jar



WSDL2Java Problems

- The following command doesn't work

```
java -cp %AXISCP% org.apache.axis.wsdl.WSDL2Java  
-o src -s [-S true] filename.wsdl
```

- Try to type the command WITHOUT the `-o` option first
- Retype the command WITH the `-o` option
- Note: you don't have to create the output directory (i.e., `src`) in advance (`wsdl2java` will do it for you)

