Conception of Information Systems
Lecture 11: Building Web Services with JAX-RPC

24 May 2005

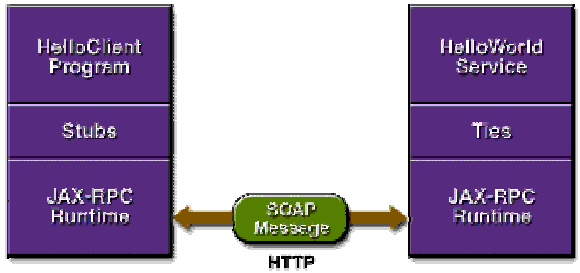
http://lsirwww.epfl.ch/courses/cis/2005ss/

Outline

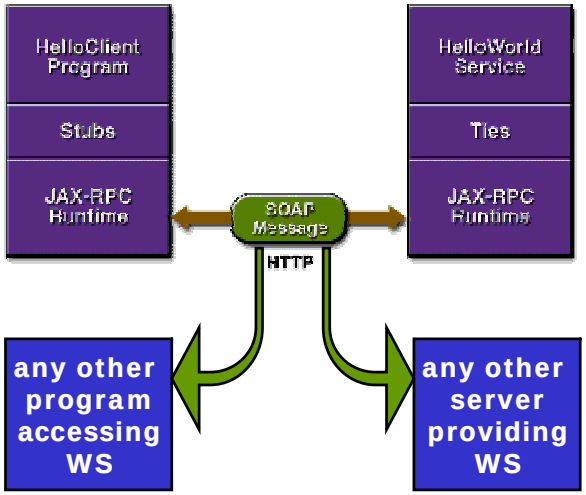
1. *The Big Picture*
2. *SOAP*
3. *WSDL*
4. *UDDI*
5. JAX-RPC: the Big Picture
6. Building and Deploying a JAX-RPC Web Service
7. Building and Running a JAX-RPC Client

5 JAX-RPC: the Big Picture

- Problems
  - implement a Web Service and deploy it over a Web Server in a way that is consistent with the Web Service description in the WSDL file
  - invoke from a Java application a Web Service with a known description
- Solution
  - JAX-RPC (Java API for XML-based RPC)
- Hello World example
  - The Web Service exposes a `sayHello(String s)` method

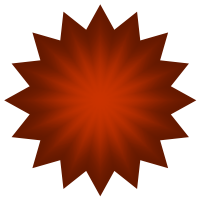


Java view

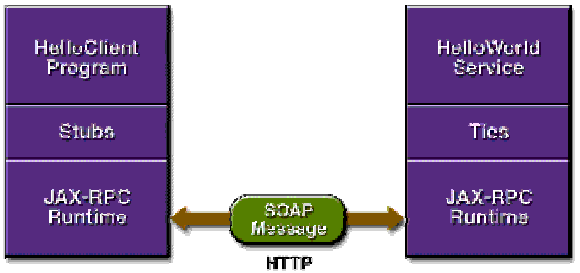


Web Services view

Runtime View of a Web Service



- The `HelloClient` program invokes a method on a *stub*
  - a local object that represents the remote service (RMI)
  - The stub invokes routines in the JAX-RPC runtime system (RS)
  - The RS converts the remote method invocation into a SOAP message
  - The RS transmits the message as an HTTP request
- The Web server receives the HTTP request
  - the RS extracts the SOAP message from the request
  - it translates it into a method call
  - The RS invokes the method on the *tie* object
  - The tie object invokes the method on the implementation of the `HelloWorld` service
- The `HelloWorld` application produces a response
  - The RS on the server converts the method's response into a SOAP message
  - it then transmits the message back to the client as an HTTP response
- The RS on the client receives an HTTP response
  - it extracts the SOAP message from the HTTP response
  - it translates it into a method response for the `HelloClient` program



Development of a Web Service with JAX-RPC

- **Application / Service developer** creates the Web Service
  - `HelloIF.java` - the service definition interface
  - `HelloImpl.java` - the service definition implementation class, which implements the `HelloIF` interface
- **Application / Client developer** creates the client program
  - `HelloClient.java` - the remote client that contacts the service and then invokes its methods
- **wscompile tool** generates WSDL and stubs (run by the developer)
  - `config.xml` - a configuration file read by the `wscompile` tool to create WSDL
  - `config-client.xml` – a configuration file read by the `wscompile` tool to create stubs
- **wsdeploy tool** makes the service available by generating ties (run by the developer)
  - parameters provided by GUI

6 Building and Deploying a Web Service – Step by Step

1. Code (and compile) the service definition interface and implementation class
2. Generate the WSDL file
3. Generate the ties and deploy the service

Step 1: The Interface and Service Implementation Class

The service definition interface extends the `java.rmi.Remote` interface.

```
package cis;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface HelloIF extends Remote {
     public String sayHello(String s) throws RemoteException; }
```

The service definition implementation class implements the interface.

```
package cis;
public class HelloImpl implements HelloIF
     throws RemoteException {
     public String message = "Hello ";
     public String sayHello(String s)
     { return message + s; } }
```

- The methods can throw the `java.rmi.RemoteException` or one of its subclasses. The methods may also throw service-specific exceptions.

To compile the classes: `javac HelloIF.java HelloImpl.java`

Step 2: Generate the WSDL file

- Command: `wscompile -define -mapping build/mapping.xml -d build -nd build -classpath build/config.xml`
- Command arguments:
  - The `define` flag is used to create the WSDL file
  - The `mapping` file is used by the `wsdeploy` tool for deployment
  - The `d` and `nd` options specify where to create the WSDL and mapping files

- Example of configuration file (config.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service
      name="HelloWorldService"
      targetNamespace="http://lsirwww.epfl.ch/"
      typeNamespace="http://lsirwww.epfl.ch/"
      packageName="cis">
      <interface name="cis.HelloIF"/>
  </service>
</configuration>
```

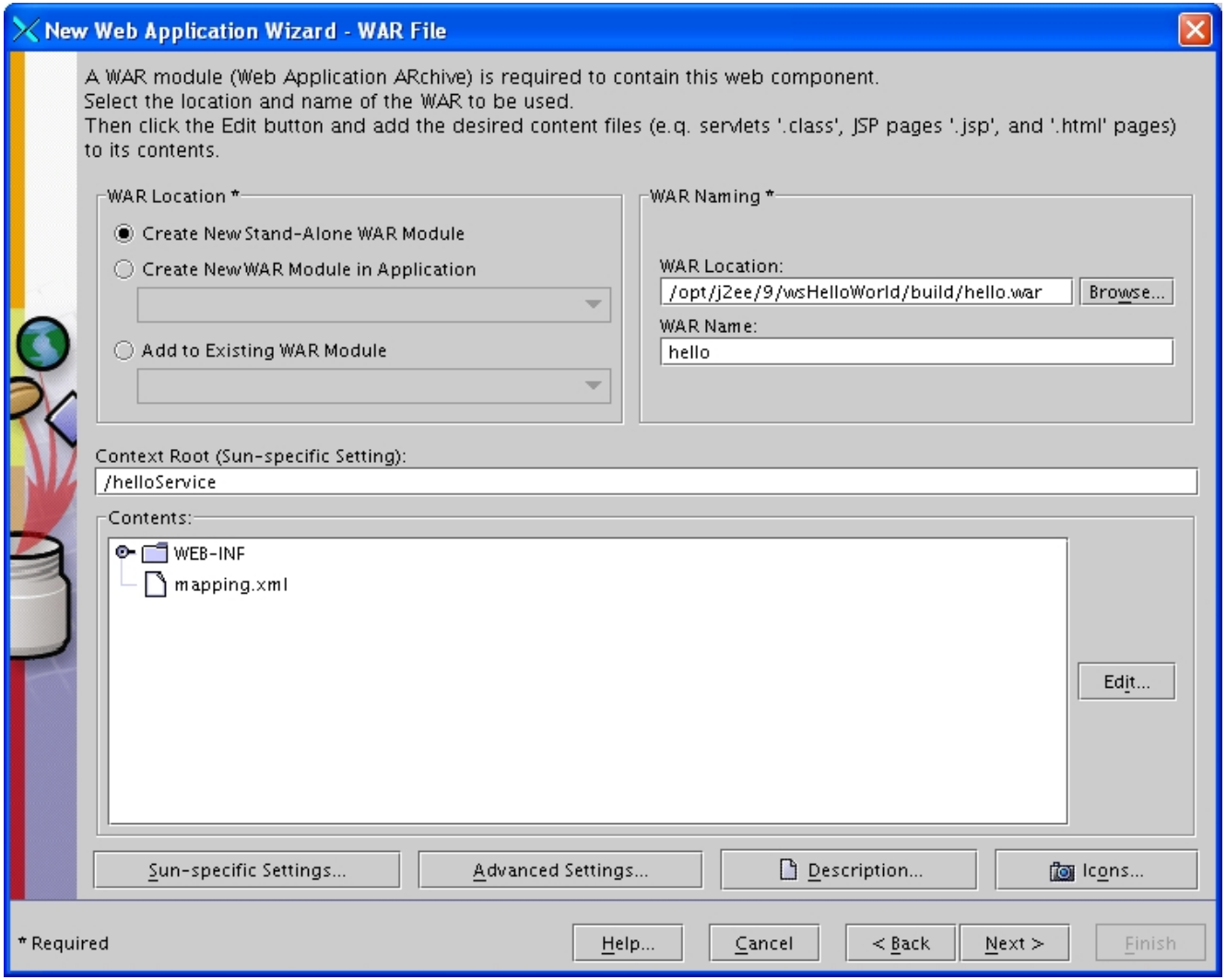
Generated WSDL File

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloWorldService" targetNamespace="http://lsirwww.epfl.ch/"
    xmlns:tns="http://lsirwww.epfl.ch/" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types/>
  <message name="HelloIF_sayHello">
    <part name="String_1" type="xsd:string"/></message>
  <message name="HelloIF_sayHelloResponse">
    <part name="result" type="xsd:string"/></message>
  <portType name="HelloIF">
    <operation name="sayHello" parameterOrder="String_1">
      <input message="tns:HelloIF_sayHello"/>
      <output message="tns:HelloIF_sayHelloResponse"/></operation></portType>
  <binding name="HelloIFBinding" type="tns:HelloIF">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="sayHello">
      <soap:operation soapAction=""/>
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="http://lsirwww.epfl.ch/"/></input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="http://lsirwww.epfl.ch/"/></output></operation></binding>
  <service name="HelloWorldService">
    <port name="HelloIFPort" binding="tns:HelloIFBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/></port></service></definitions>
```

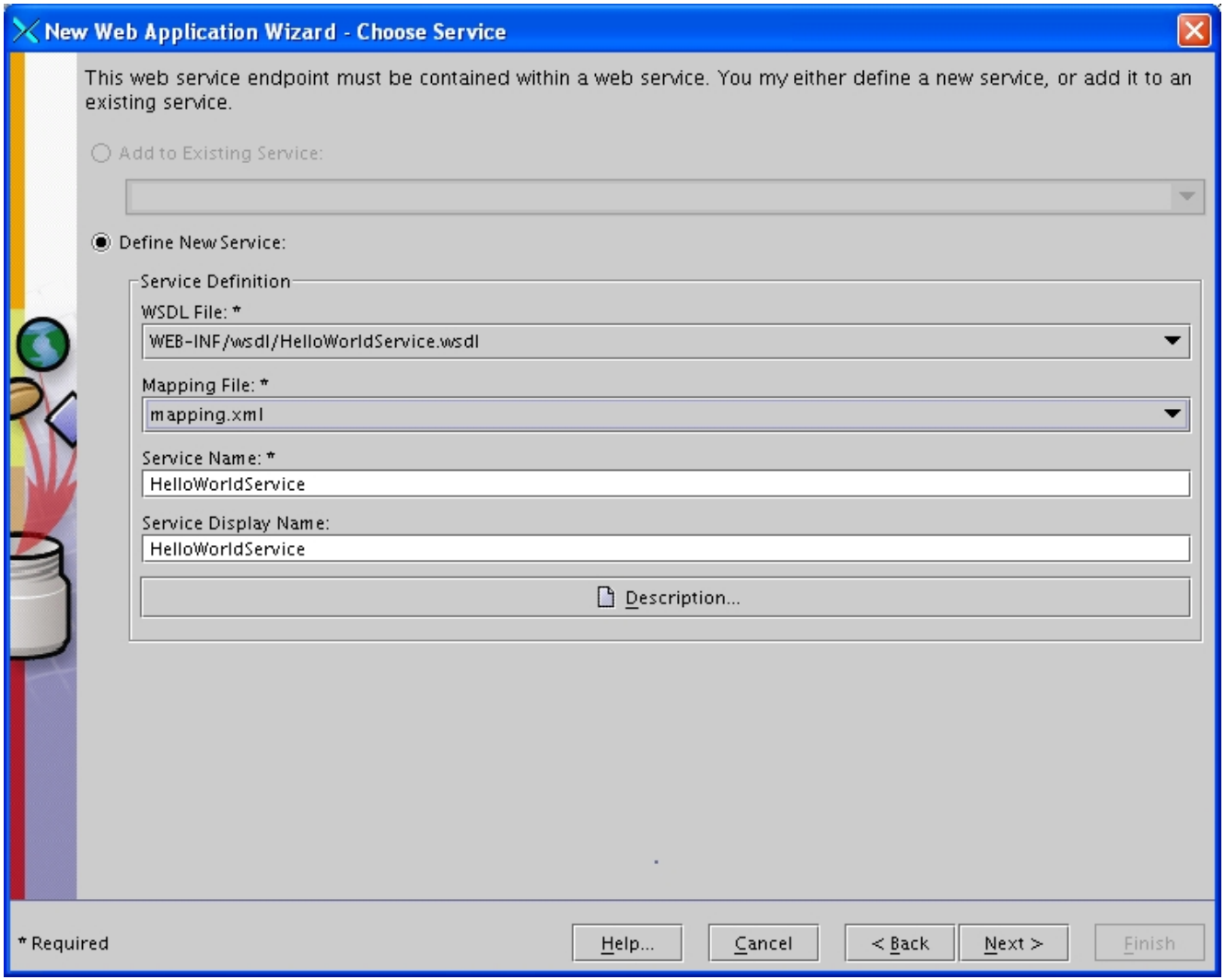
Step 3: Deploy the Service

- Behind the scene, the Web Service is deployed in a server-side container
  - a servlet container
  - an EJB container
- Let us assume that the JAX-RPC Web Service tie is implemented as a servlet
- The application is bundled in a WAR (Web Application aRchive) file
  - it needs a Context-Root
- To configure the servlet to handle the Web Service, specify
  - WSDL file
  - service endpoint implementation (HelloImpl)
  - service endpoint interface (HelloIF)
  - servlet alias (for Web access)
- The ties are generated automatically upon deployment

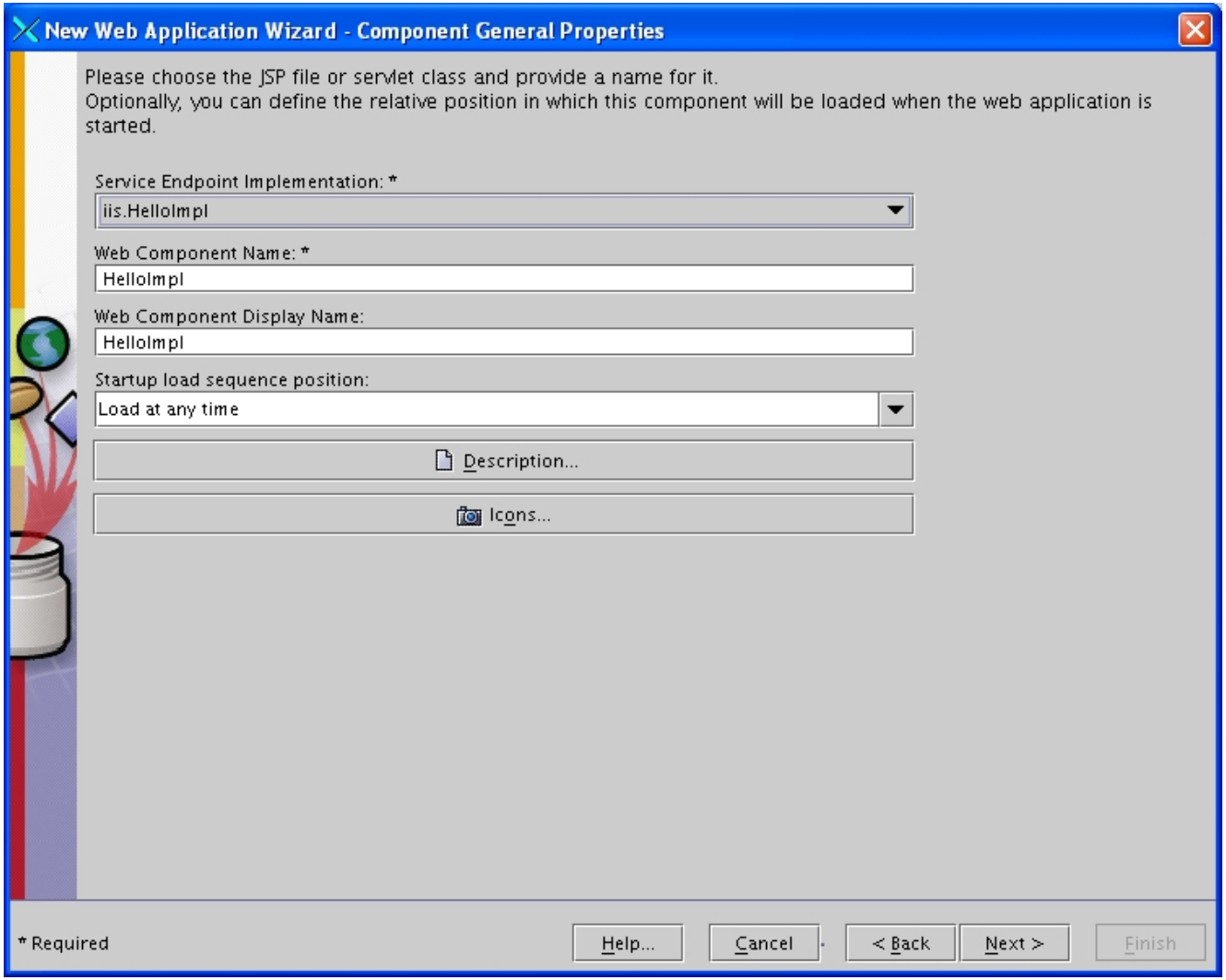
WAR file and Context-Root



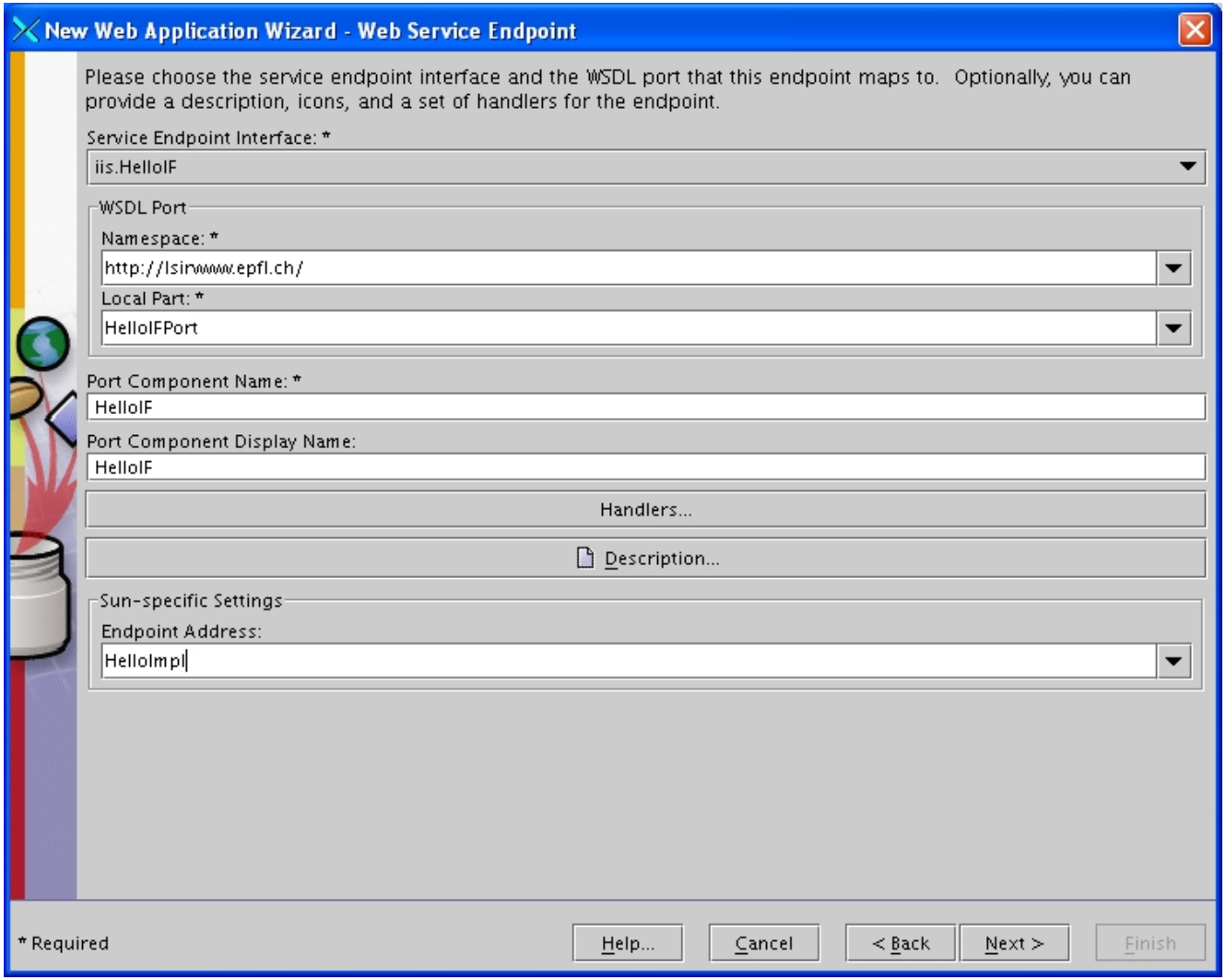
WSDL File



Service Endpoint Implementation



Service Endpoint Interface



WSDL File after Deployment

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloWorldService" targetNamespace="http://lsirwww.epfl.ch/"
    xmlns:tns="http://lsirwww.epfl.ch/" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types/>
  <message name="HelloIF_sayHello">
    <part name="String_1" type="xsd:string"/></message>
  <message name="HelloIF_sayHelloResponse">
    <part name="result" type="xsd:string"/></message>
  <portType name="HelloIF">
    <operation name="sayHello" parameterOrder="String_1">
      <input message="tns:HelloIF_sayHello"/>
      <output message="tns:HelloIF_sayHelloResponse"/></operation></portType>
  <binding name="HelloIFBinding" type="tns:HelloIF">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="sayHello">
      <soap:operation soapAction=""/>
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
    namespace="http://lsirwww.epfl.ch/"/></input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
    namespace="http://lsirwww.epfl.ch/"/></output></operation></binding>
  <service name="HelloWorldService">
    <port name="HelloIFPort" binding="tns:HelloIFBinding">
      <soap:address location="http://lsir-cis-pcx:8009/hello/helloService"/>
    </port></service></definitions>
```

7 Building and Running a Client – Step by Step

1. Generate the stubs
2. Code and compile the client
3. Package and execute the client

Step 1: Generate the Stubs

- Run the `wscompile` command:

```
wscompile -gen:client -d build -classpath build config-client.xml
```

- This command generates stub files based on the information it reads in the WSDL and `config-client.xml` files
  - The location of the WSDL file is specified by the `<wsdl>` element of the `config-client.xml` file
  - The `-d` option can be used to specify a directory where to place generated output files

- Example of `config-client.xml` file:

```
<configuration
   xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
   <wsdl location="build/BankService.wsdl" packageName="cis"/>
</configuration>
```

- Beware: the `wscompile` command does NOT compile the client code (see next slide)

Step 2: Code and Compile the Client

Client application

```
package cis;
import javax.xml.rpc.Stub;
public class HelloClient {
  private String endpointAddress;
  public static void main(String[] args) {
    try {
      Stub stub = createProxy();
      stub._setProperty (javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY, args[0]);
      HelloIF hello = (HelloIF)stub;
      System.out.println(hello.sayHello(args[1]));
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  private static Stub createProxy() {
    // Note: the _Impl suffix is implementation-specific standard naming convention
    return (Stub) (new HelloWorldService_Impl().getHelloIFPort());
  }
}
```

Web Service address

Compile the client application

```
javac –classpath system_jars:dir_with_server_class_files:dir_with_stub_class_files
  HelloClient.java
```

Step 3: Package and Execute the Client

- Package the client classes into a JAR file
- Command to pack the client
  `jar cvf hello-client.jar` *`all_client_class_files:all_server_class_files`*
- Command to run the client
  `java –classpath hello-client.jar:jwsdp-jars hello.HelloClient`
- Note: in the exercise, all commands are automated by using the `asant` scripting utility provided by Sun J2EE

References

- JAX-RPC
  - http://java.sun.com/xml/jaxrpc/
  - http://java.sun.com/webservices/docs/1.3/tutorial/doc/IntroWS7.html
- Standard documents
  - http://www.w3.org/2002/ws/
  - http://www.w3.org/TR/2002/CR-soap12-part0-20021219/ (SOAP primer)
  - http://www.w3.org/TR/SOAP/
  - http://www.w3.org/TR/wsdl
  - http://www.uddi.org
- Articles
  - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/wsdlexplained.asp?frame=true (Introduction to WSDL)
  - http://www2002.org/CDROM/alternate/395/ (Comparison to CORBA)