BSOA Orchestra

Overview & Examples

BULL SERVICE-ORIENTED ARCHITECTURE (BSOA)

REFERENCE
86 A2 53ER 01

BULL SERVICE-ORIENTED ARCHITECTURE (BSOA)

BSOA Orchestra

Overview & Examples

BSOA Orchestra v3.0

Software

November 2006

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
86 A2 53ER 01

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this book.

Intel® and Itanium® are registered trademarks of Intel Corporation.

Windows® and Microsoft® software are registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark in the United States of America and other countries licensed exclusively through the Open Group.

Linux® is a registered trademark of Linus Torvalds.

Table of Contents

List of Figures

List of Tables

Preface

The purpose of this tutorial is to guide the user through writing a first BPEL Process. It involves creating three different files: a BPEL file, a WSDL file and a client file.

This tutorial is based on the echo sample. This sample receives a Web Service call and replies with the same value.

Chapter 1. Writing a BPEL Process

1.1 BPEL Language

BPEL stands for **B**usiness **P**rocess **E**xecution **L**anguage.

The BPEL language (ws-bpel.pdf) is the way to describe Business Processes. It is an XML, schema-based standard defined by the Oasis consortium. It enables the composition of multiple synchronous and asynchronous Web Services into an end-to-end business flow.

1.2 Implementing an Orchestra BPEL Process

An Orchestra process consists of a BPEL file containing the BPEL language statements, a WSDL file describing the Web Service interface (message formats, available operations, etc) for the BPEL process, a WSDL file describing the partners in the business process, and the WSDL files describing the partner Web Services. These files are described in following sections.

These files are deployed to an Orchestra instance. Deploying a process translates the BPEL file into Java classes used by Orchestra to execute each BPEL activity and Java classes to call the external web services. These classes are located in Orchestras class path.

Note that the README file for each sample gives a command line example for deploying that sample. These samples may be followed for user processes, but they assume the process is located in the /BPEL/samples directory. This directory may be lost when reinstalling Orchestra. See the command line help for the bsoap deploy command for locating processes in a different folder.

An Orchestra BPEL process is itself an Axis Web Service and may be called in any environment that can call a Web Service. This is typically a JSP or a freestanding Java client. The samples provide a freestanding Java client to run the sample and the bsoap launch command runs the sample. Again, the README file provides a command line to run each sample. Since the process is also an Axis Web Service, the Axis wsdl2java command can be used with the WSDL file for the process to produce stubs for use by other Java clients and JSPs.

1.3 Writing a WSDL File

WSDL stands for **W**eb **S**ervices **D**escription **L**anguage.

WSDL is a document written in XML. The document describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes.

For a complete tutorial on how to write a WSDL file refer to the WSDL Tutorial (http://www.w3schools.com/wsdl/default.asp) .

For the simple "echo" sample, the message type that will be used by the Web Service must be defined. In this case, it is just a simple message with one part of type String:

```
<message name="StringMessageType">
  <part name="echoString" type="xsd:string"/>
</message>
```

The web service, the operations, and the messages that are involved need to be described. For this sample, the following must be defined:

```
<portType name="echoPT">
  <operation name="echo">
    <input message="tns:StringMessageType"/>
    <output message="tns:StringMessageType"/>
  </operation>
</portType>
```

Then, the binding must be specified. It corresponds to the message format and protocol details for a web service. In this case:

```
<binding name="EchoPTSOAPBinding" type="tns:echoPT">
  <soap:binding
  style="rpc"transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="echo">
    <input>
      <soap:body use="encoded" namespace="urn:echocomplex:bpel:bsoap"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:body use="encoded" namespace="urn:echocomplex:bpel:bsoap"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>
</binding>
```

Finally, define the location of the web service. The following is the code for this sample, if Orchestra is running on the port 9000 of localhost:

```
<service name="EchoServiceBP">
  <port name="echoPT" binding="tns:EchoPTSOAPBinding">
    <soap:address
    location="http://localhost:9000/axis/services/echoPT"/>
  </port>
</service>
```

The entire WSDL file can be viewed in the section titled WSDL File.

1.4 Writing a BPEL File

Writing a BPEL file manually is fairly difficult. A BPEL editor such as Zenflow is useful. Refer to the Zenflow Overview (Zenflow.pdf or zenflow.htm).

Hereafter are described the different parts of a simple process: "Echo".

First, define the process tag with the name of the process and its target namespace. The **name** attribute of the process tag (the first of a BPEL file) must be the same as the process name.

```
<process name="echo"
     targetNamespace="http://orchestra.objectweb.org/samples/echo"
     xmlns:tns="http://orchestra.objectweb.org/samples/echo"
     xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-
process/">
```

Next, define the partnerLinks of the process:

```
<partnerLinks>
  <partnerLink name="echo" partnerLinkType="tns:Echo"
  myRole="service"/>
</partnerLinks>
```

Then define the variables and handlers that will be used by the process:

```
<variables>
  <variable name="request" messageType="tns:StringMessageType"/>
</variables>
```

Define what the process does. The following is the code for the echo sample:

```
<sequence name="EchoSequence">
  <receive partnerLink="echo" portType="tns:echoPT"
        operation="echo" variable="request"
        createInstance="yes" name="EchoReceive"/>
  <reply partnerLink="echo" portType="tns:echoPT"
        operation="echo" variable="request"
        name="EchoReply"/>
</sequence>
```

The entire BPEL file can be viewed in the section titled BPEL File.

1.5 Writing a Client File

A BPEL process is seen as a standard web service. To write a test client, the Web Service must be called:

```
EchoServiceBP es=new EchoServiceBPLocator();
EchoPT ept=es.getechoPT();
StringHolder in = new StringHolder(args[0]);
ept.echo(in);
```

The first two lines get the Port Type Method. A String holder is then defined that is the in/out message of the Web Service. The Web Service is called with that variable.

The entire Client file can be viewed in the section titled Client File.

1.6 Entire Files for Echo Sample

This section provides complete file examples for the Echo sample as follows:

- Section 1.5.1 WSDL File
- Section 1.5.2 BPEL File
- Section 1.5.3 Client File

1.6.1 WSDL File

```
<definitions
 targetNamespace="http://orchestra.objectweb.org/samples/echo"
 xmlns:tns="http://orchestra.objectweb.org/samples/echo"
 xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

 <message name="StringMessageType">
   <part name="echoString" type="xsd:string"/>
 </message>

 <portType name="echoPT">
   <operation name="echo">
     <input message="tns:StringMessageType"/>
     <output message="tns:StringMessageType"/>
   </operation>
 </portType>

 <plnk:partnerLinkType name="Echo">
   <plnk:role name="service">
     <plnk:portType name="tns:echoPT"/>
   </plnk:role>
 </plnk:partnerLinkType>

 <binding name="EchoPTSOAPBinding" type="tns:echoPT">
   <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="echo">
     <input>
       <soap:body use="encoded"
        namespace="http://orchestra.objectweb.org/samples/echo"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </input>
     <output>
       <soap:body use="encoded"
        namespace="http://orchestra.objectweb.org/samples/echo"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </output>
   </operation>
 </binding>

 <service name="EchoServiceBP">
   <port name="echoPT" binding="tns:EchoPTSOAPBinding">
     <soap:address
      location="http://localhost:9000/axis/services/echoPT"/>
   </port>
 </service>

</definitions>
```

1.6.2 BPEL File

```
<process name="echo"
 targetNamespace="http://orchestra.objectweb.org/samples/echo"
 xmlns:tns="http://orchestra.objectweb.org/samples/echo"
 xmlns="http://schemas.xmlsoap.org/ws/2003/03/
 business-process/">

 <partnerLinks>
   <partnerLink name="echo" partnerLinkType="tns:Echo"
    myRole="service"/>
 </partnerLinks>

 <variables>
   <variable name="request" messageType="tns:StringMessageType"/>
 </variables>

 <sequence name="EchoSequence">
   <receive partnerLink="echo" portType="tns:echoPT"
    operation="echo" variable="request"
    createInstance="yes" name="EchoReceive"/>
   <reply partnerLink="echo" portType="tns:echoPT"
    operation="echo" variable="request"
    name="EchoReply"/>
 </sequence>
</process>
```

1.6.3 Client File

```
package org.objectweb.orchestra.samples.echo;
import javax.xml.rpc.holders.*;

import java.lang.*;

import java.math.*;

public class EchoClient
{
    public static void main(String [] args)
    {
        int argNb = 1;
        if (args.length != argNb) {
            System.out.println(">>>>> ERROR: Number of arguments
                                  needed : "+argNb);
            System.out.println(">>>>>      The first should be a
                                  string");
        } else {
            try {
                EchoServiceBP es=new EchoServiceBPLocator();
                EchoPT ept=es.getechoPT();
                StringHolder in = new StringHolder(args[0]);
                ept.echo(in);
                System.out.println(">>>>> ECHO response received:
                                       "+ in.value );
            } catch (Exception e) {
                e.printStackTrace(System.err);
            }
        }
    }
}
```

Chapter 2. Running the Demos

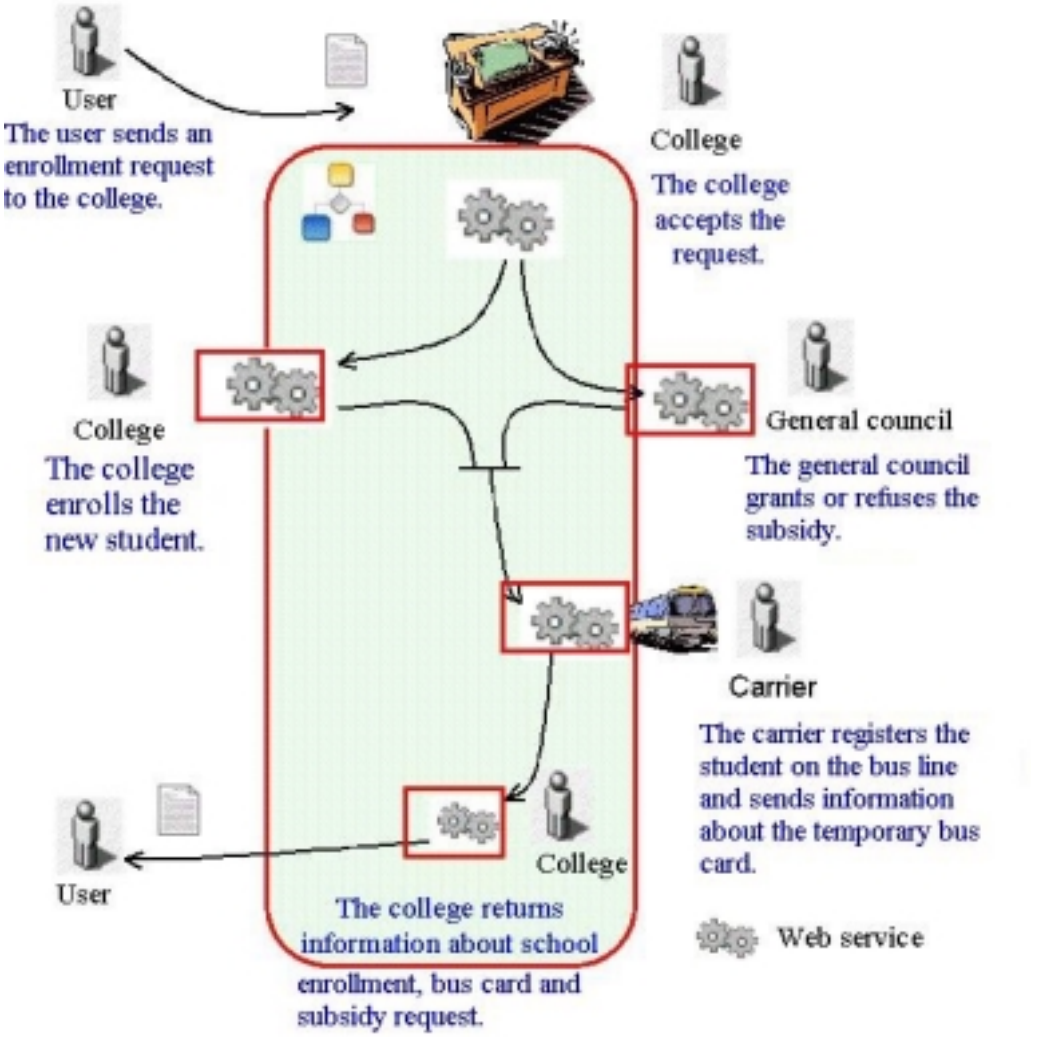
2.1 College Demo

The purpose of this tutorial is to guide the user through launching the College Demo that is provided with the package.

The College Demo is a BPEL process that models some simple steps in enrolling a student in a college. The student not only enrolls, but also requests a grant or subsidy (scholarship) and gets a transportation pass from an appropriate bus company.

The following figure graphically shows the steps.

Figure 2-1. College Demo Process Steps



The demo must be installed (Sections 2.1.1, 2.1.2, 2.1.3) before the process can be run (Section 2.1.4).

2.1.1 College Demo Setup

First, check to make sure that the BPEL JOnAS server is not running (run `bsoap stop` to stop it).

If the College Demo has previously been deployed, a necessary prerequisite is to do a Clean operation (see Cleaning the College Demo).

If the default Orchestra installation is to be used to run the College Demo on the local machine, with or without port number changes, then no further setup is necessary. Skip to the section titled Installing the College Demo.

When using the **jms service**, change the JONAS_BASE/conf/jonas.properties:
Add the CollegeDemoQueue to the list of JMS queues:

```
jonas.service.jms.queues
BPELQueue,BPELReplyQueue,BPELFaultQueue,CollegeDemoQueue
```

Note that JMS is the default message service and these queues are added when Orchestra is installed.

When using the **JORAM rar**, change the JONAS_BASE/conf/joramAdmin.xml file to include the following lines, if not already in the file:

```
<Queue name="CollegeDemoQueue">
  <freeReader/>
  <freeWriter/>
  <jndi name="CollegeDemoQueue"/>
</Queue>
```

Go to the $BPEL_HOME directory and specify the host and port number in the build.properties. Then go to the $BPEL_HOME/Demos/College directory, specify the host and port number where the external WS is located (to deploy them on the computer hosting Orchestra, enter the same values) in the college.properties file.

2.1.2 Installing the College Demo

Start the BPEL JOnAS server:

```
bsoap start
```

Then run successively:

```
demos college config
```

and

```
demos college install
```

2.1.3 Deploying the College Demo

Before being deployed, the College Demo should have been installed (see the section titled Installing the College Demo).

Make sure that the BPEL JOnAS server is running (if not, run `bsoap start` to start it).

Then, execute:

```
demos college deploy
```

2.1.4 Using the Web Interface of the College Demo

Once the demo is running, any user can try to register his son or daughter in the College.

1. Open a web browser window.

2. Open page:
   `http://{host}:{port}/college`
   For example: `http://localhost:9000/college`

3. Then register a student by clicking on Registration, filling in the text boxes for First Name, Last Name, etc. and choosing a Grade and Bus Stop then clicking on Submit. This should display a File Information view with all of the supplied information, plus a File Number. Save the File Number for the next step.

4. Verify that the registration has been validated.
   - First click on Welcome Page in the File Information view, and then click on File Follow Up in the Home page view.
   - Enter the saved File Number and the First/Last names separated by a space into the input boxes, then click on Submit. The File Follow Up view should display the following message followed by the File Information:
   - *Your file has been handled. Your kid is registered for College. The file for the bus ticket has been approved by the State Department. The bus company has also approved the request.*

2.1.5 Cleaning the College Demo

To clean the College Demo, execute:

```
demos college clean
```

Note:

BSOAP STOP is required to undeploy the college demo using DEMOS COLLEGE CLEAN if the college demo has already been deployed (otherwise the EAR file can not be removed). An alternative is to use the JONAS Administration Console to undeploy the EAR file. For example: `http://localhost:9000/jonasAdmin/`, login, and use the menu entry Domain->Server Jonas->Deployment->Applications.

2.2 Telecom Demo

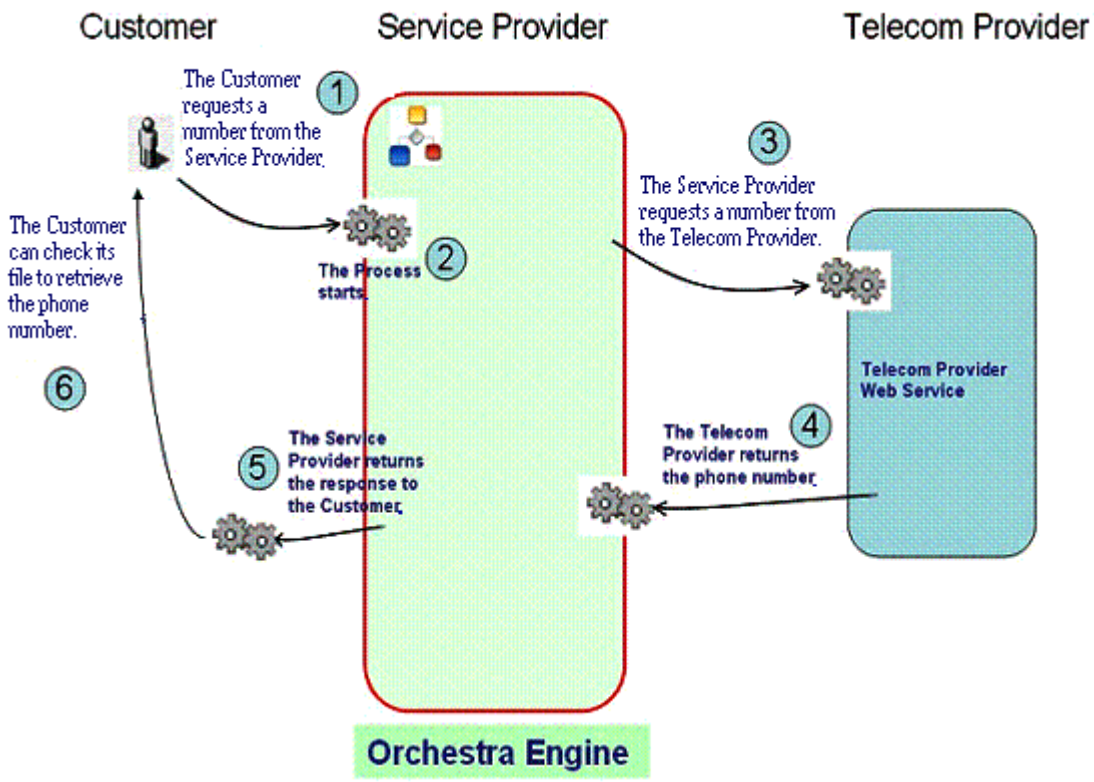
The purpose of this tutorial is to guide the user through launching the Telecom Demo provided with the package.

The following figure describes the flow of the process. The process of this demo involves a telecom phone number request. There are three partners: a customer, a service provider, and a telecom provider. The customer requests a phone number from the service provider. Then the process starts and the service provider sends the request to the telecom provider. The telecom provider responds to the service provider with a phone number. This phone number is then sent to the customer.

The process of this demo concerns a telecom phone number request. There are three partners: a customer, a service provider, and a telecom provider. The customer requests a phone number from the service provider. Then the process starts and the service provider sends the request to the telecom provider. The telecom provider responds to the service provider with a phone number. This phone number is then sent to the customer.

To show the restart mechanism, the scenario is to start the process on the customer's side by requesting a new phone number. Then the Orchestra engine of the service provider can be killed. The Orchestra engine can be restarted and the provider will furnish the phone number. This scenario demonstrates that the engine restarts properly and receives the provider's message.

Figure 2-2. Telecom Demo Process Flow



The demo must first be installed (Sections 2.2.1– 2.2.2). Then it can be executed (Section 2.2.3). Finally, this demo can show the "restart" mechanism available in the BPEL engine (Section 2.2.4).

2.2.1 Installing the Telecom Demo

First, check that the Orchestra engine is not running (run `bsoap stop` to stop it). Then go to the $BPEL_HOME directory, and specify a host and a port number in the build.properties file.

Notes:

The default Orchestra installation process will have already set these to localhost and the specified installation port number.

Use Cleaning the Telecom Demo first, if the Telecom Demo has already been deployed.

This installation involves two steps (in the following order):

- Install the telecom provider side on a remote server.
- Install the service provider side locally.

INSTALL THE TELECOM PROVIDER SIDE

The telecom provider is a remote web service deployed on a different JOnAS server than the one on which the Orchestra engine is running.

1. Install a JOnAS server for the telecom provider on a remote computer.
   After installation, this may require setting a $JONAS_BASE value and executing "ant create_jonasbase". Then in $JONAS_BASE/conf/jonas.properties, specify either JMS as a JOnAS service or a JORAM resource adapter (for the default, see the JOnAS Installation guide).

2. On the remote computer hosting the JOnAS server, do the following based on whether using the jms service or the JORAM rar:
   - If using the jms service, change the JONAS_BASE/conf/jonas.properties:
     Add the TelecomProviderQueue to the list of JMS queues:
     ```
     jonas.service.jms.queues sampleQueue, TelecomProviderQueue
     ```
   - If using the JORAM rar, change the JONAS_BASE/conf/joramAdmin.xml file by adding the following lines, if not already in the file:
     ```
     <Queue name="TelecomProviderQueue">
       <freeReader/>
       <freeWriter/>
       <jndi name="TelecomProviderQueue"/>
     </Queue>
     ```

3. On the local computer hosting the Orchestra engine, go to the $BPEL_HOME/Demos/Telecom/TelecomProviderWebapp directory, open the build.properties file and fill in the remote computer's values for: server.name (default jonas), server.host (use its DNS name), and server.port (default 9000). Also fill in the local computer's values for: service.host (use its DNS name), and service.port (default 9000).

Warning:

Specify the complete name for server.host property. Do not use localhost and 127.0.0.1 as this file will be used by the customer.

4. On the local computer hosting the Orchestra engine, run:
   ```
   demos telecom config
   ```

5. On the remote computer, in the TelecomProviderWebapp directory, run:
   ```
   ant install
   ```

6. Then copy the $BPEL_HOME/Demos/Telecom/TelecomProviderWebapp directory from the local computer to some convenient directory on the remote computer acting as the telecom provider. A simple way to do this is to place the TelecomProviderWebapp directory contents in a JAR file, FTP it to the remote compute, and then extract the JAR file contents.
   Check to make sure that the telecom provider JOnAS server is not running.

7. Launch the remote telecom provider JOnAS server (jonas start).

8. On the remote computer, in the TelecomProviderWebapp directory, run:
   ```
   ant deploy
   ```
   Check the deployment by using a browser pointed at:
   http:// {host}:{port}/telecomProvider/

INSTALL THE SERVICE PROVIDER SIDE

The Service Provider is the BPEL process.

When using the **jms service**, change the JONAS_BASE/conf/jonas.properties.
Add the TelecomDemoQueue to the list of JMS queues:

```
jonas.service.jms.queues
BPELQueue,BPELReplyQueue,BPELFaultQueue,TelecomDemoQueue
```

When using the **JORAM rar**, change the JONAS_BASE/conf/joramAdmin.xml file to include the following lines, if not already in the file:

```
<Queue name="TelecomDemoQueue">
  <freeReader/>
  <freeWriter/>
  <jndi name="TelecomDemoQueue"/>
</Queue>
```

Note:

With the default Orchestra installation, the jonas.properties and the joramAdmin.xml files already have the correct JMS values, so no additions are necessary.

Before installing the Telecom Demo, launch Orchestra if it is not already running.
This should be done by running `bsoap start`.

Then run: `demos telecom install`

2.2.2 Deploying the Telecom Demo

Before being deployed, the Telecom Demo should have been installed (see Installing the Telecom Demo). Be sure the BPEL engine JOnAS server is running on the local machine (if not, run `bsoap start` to start it).

Then, execute: `demos telecom deploy`

2.2.3 Use the Web Interface of the Telecom Demo

Once the demo is running, try to obtain a phone number.

9. Open a web browser.

10. Open page: http://{host}:{port}/telecom
    (for example: http://localhost:9000/telecom).

11. Then, make a request to register a phone number, by filling in the form and clicking on SUBMIT

12. Remember the first name, last name, and file number that is returned for use in the next step, then click on Welcome Page.

13. Click on the File Followup button to see if the registration has been validated. Fill in the form with the returned file number and the first and last names separated by a space and click on the Submit Button, then click on the Validate button.

2.2.4 Restarting the Telecom Demo

To show the restart mechanism, kill the Orchestra engine running the Service Provider, and restart it.

Note:

For Windows, use the task manager to kill the command window where BSOAP START was done.

First, restart Orchestra: `bsoap start`.

Then launch the restart mechanism: this can be done by executing `bsoap restartEngine`.

This script restarts all instances that are not finished and are waiting for a message. Here the process is waiting for the remote Telecom Provider to validate the request. Thus, it will restart.

The request can then be validated using the following web page (see the steps under **INSTALL THE TELECOM PROVIDER SIDE** in Section 2.2.1:

```
http://{host}:{port}/telecomProvider
```

Then, use the telecom interface specified in Section 2.2.3 to see if the request is validated.

2.2.5 Cleaning the Telecom Demo

To clean the Telecom Demo, run: `demos telecom clean`

Note:

BSOAP STOP is necessary before running DEMOS TELECOM CLEAN.

2.3 Loan Approval Demo

Loan Approval Demo process:

A customer makes a credit loan request to a loan service.

- If the amount requested is above or equals 10,000, then the request is sent directly to an approver.
- If the requested amount is lower than 10 000, the request is sent to an assessor who checks the risk associated to the requester.
- If the risk is low, the loan is approved.
- If the risk is medium or high, then the request is sent to the approver.

Approver behavior:

- If the amount requested is above 100,000, the loan is refused.
- If the amount requested is above 15,000 and the risk associated with the requester is high, the loan is refused.
- If the amount requested is above 30,000 and the risk associated with the requester is medium, the loan is refused.

Else, the loan is accepted.

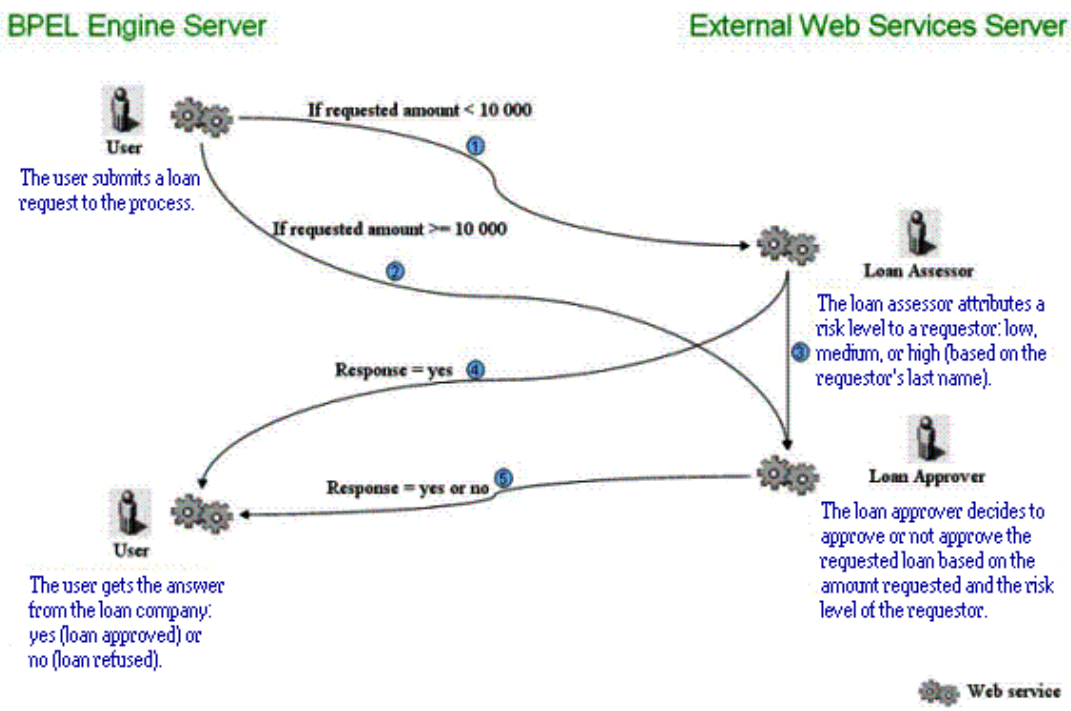
Medium risk list: LastName equals to:

1. juliani
2. zvong
3. kadash
4. pouit
5. koron

High-risk list: LastName equals to:

1. robert
2. durand
3. dupont
4. macfy
5. frund
6. liogi
7. max

Figure 2-3. Loan Approval Process for Loan Approval Demo



Loan Approval Samples:

Table 2-1. Loan Approval Demo Samples

Name	Amount	Risk Level	Tracks executed	Loan response
DiMento	5 000	low	1,4	yes
Arthui	11 000	low	2,5	yes
Koron	9 000	medium	1,3,5	yes
Robert	11 000	high	2,5	yes
Tyuin	101 000	low	2,5	no
Pouit	30 000	medium	2,5	yes
Kadash	31 000	medium	2,5	no
Frund	15 000	high	2,5	yes
Macfy	16 000	high	2,5	no

2.3.1 Loan Approval Demo Setup

If the Loan Approval Demo has never been installed before and the default Orchestra installation is being used, with or without port number changes, then skip to the section Installing the Loan Approval Demo.

If the Loan Approval Demo was previously deployed, use the cleaning procedure (see Cleaning the Loan Approval Demo) to remove it.

Go to the $BPEL_HOME directory, and specify a host and port number in the build.properties file. (The default orchestra installation will already have set the host/port values in the build.properties file.)

2.3.2 Installing the Loan Approval Demo

To install the Loan Approval Demo, perform the following steps:

1. First, check that the BPEL JOnAS server is running (if not, run `bsoap start` to start it).
2. Then, execute:
   ```
   demos loan install
   ```

2.3.3 Deploying the Loan Approval Demo

Before being deployed, the Loan Approval Demo should have been installed (see Installing Loan Approval Demo).

1. First, check that the BPEL JOnAS server is running (if not, run `bsoap start` to start it).
2. Then, execute:
   ```
   demos loan deploy
   ```

2.3.4 Deploying the Loan Approval Demo External Web Services

Deploying the Loan Approval Demo External Web Services on Remote Server Locally

To deploy the external web services on the same server on which the JOnAS BPEL engine is running, instead of doing the procedure in the previous section, just execute the following on the local machine:

```
demos loan deployExt
```

Deploying the Loan Approval Demo External Web Services on Remote Server

Before being deployed, the Loan Approval Demo should have been installed (see Loan Approval Demo Setup.

Make sure that the JOnAS server hosting the external web services is running with 'ws' jonas service activated.

Note:

The WS service is part of a default JONAS installation; therefore no changes should be necessary.

Then, copy the $BPEL_HOME/Demos/LoanApproval/wsp directory on the server hosting the JOnAS server for external web services.

Go to this directory on the external server and execute:

```
ant install-wsp
```

Setting Server Name, Host Number, and Port Number

Whether the external web service was installed locally or on a remote server, the ANT script will ask the following questions to allow setting the server name, host number, and port number.

Enter the JOnAS server name on which you want to deploy the Web Services (default is BPEL)

Enter the host on which you want to deploy the Web Services (default is localhost)

Enter the port on which you want to deploy the Web Services (default is 9000)

Respond with the settings that were used for the server hosting the external web service.

2.3.5 Using the Web Interface of the Loan Approval Demo

Once the demo is running, any user can make a request for a loan.

1. Open a web browser window.
2. Open page: http://{host}:{port}/loanApprovalDemo
   For example: http://localhost:9000/loanApprovalDemo
3. Make a loan request. The Demo can be tested by using the first two columns of table 2-1 as input for the last name and loan amount (use any first name) and then clicking on Submit.
4. See if the request was accepted. If table 2-1 was used for the input values, the expected approval results are those listed in the last column of that table. To run a test case with different input values, click on the Home button.

2.3.6 Cleaning the Loan Approval Demo

To clean the Loan Approval Demo, run: `demos loan clean`

If the external web services are on another server, go into the $BPEL_HOME/Demos/LoanApproval/wsp directory and run: `ant clean`

Note:

BSOAP STOP must be used before doing the DEMOS LOAN CLEAN operation on the local server and the external server, if used, must be stopped before doing the ANT CLEAN operation.

Chapter 3. Running a Sample

The purpose of this tutorial is to guide the user through launching the samples provided with the package. It is comprised of two steps: first, deploy a sample, and then execute the sample. For more information, see the README file for each sample located under $BPEL_HOME/samples/<sample_name>.

3.1 Deploying a Sample

Before deploying a sample, launch the JOnAS application server. This should be done by running `bsoap start`.

Then, execute: `bsoap deploy -p {samplename} –samples`

For example:

```
bsoap deploy -p echo -samples
```

Notes:

- When deploying a sample that uses externally referenced web services, those web services should have the path of their WSDL files listed at the end of the deploy command line. See the Invoke sample README file for an example.
- When deploying a process that is not in the sample folder, the following command syntax is used.

  deploy -p process -bpel process.bpel -wsdl process.wsdl -srcDir srcDir {service.wsdl} {service.wsdl}
  process is the name of the process
  srcDir is the directory containing the process
  {service.wsdl} are external services invoked by the process.

3.2 Executing a Sample

A sample must be deployed (see Deploying a Sample) before being executed.

Once it has been deployed, execute:

```
bsoap launch -p {samplename} -cc {clientname} {argument1}
{argument2} ...
```

For example:

```
bsoap launch -p echo -cc
org.objectweb.orchestra.samples.echo.EchoClient
Dupont
```

or:

```
bsoap launch -p marketplace -cc
org.objectweb.orchestra.samples.marketplace.MarketPlaceClient
buyer auto
12000
```

Notes:

- The 'invoke' and 'while' samples are dependent on the 'incrementService' sample. The 'incrementService' sample must be compiled (see Deploying a Sample) (also see the Invoke README file) before these samples can be successfully executed.

- The 'marketplace' sample requires two separate invocations of the client, one for the 'buyer' and one for the 'seller', to complete successfully:

```
bsoap launch -p marketplace -cc
org.objectweb.orchestra.samples.marketplace.MarketPlaceClient
seller auto
12000
```

```
bsoap launch -p marketplace -cc
org.objectweb.orchestra.samples.marketplace.MarketPlaceClient
buyer auto
12000
```

Chapter 4. Unit Testing

The purpose of this tutorial is to guide the user through executing unit tests. Guidance is also provided for reading JUnit reports and writing a unit test.

4.1 Executing One Unit Test as a Client Execution

Before executing a unit test, launch the JOnAS application server. This should be done by running `bsoap start`.

Then, execute:

```
tests runOne -p {testName} [-tn {testname}]
```

For example:

```
tests runOne -p unitCopy -tn testCopy1
```

If -tn arg is not specified, all suites of this test will be launched.

Note that there is a `-launchonly` option that enables re-launching a test that has already been deployed. The test will then only be executed, not redeployed.

For example:

```
tests runOne -p unitCopy -tn testCopy1 –launchonly
```

4.2 Executing All Unit Tests With JUnit

Before executing a unit test, launch the JOnAS application server. This should be done by running `bsoap start`.

Then, execute:

```
tests runAll
```

The `-launchonly` option is also available for the runAll command. It will launch the tests without re-deploying them. Therefore, they must be deployed beforehand.

```
tests runAll -launchonly
```

4.3 Reading the JUnit Report

If all the unit tests are launched, view the report of their execution in a browser window.

Open the file:

$BPEL_HOME/tests/org/objectweb/orchestra/tests/unit/reports/index.html

Chapter 5. Advanced Configuration

5.1 Orchestra Engine Configuration

The configuration of the Orchestra Engine can be changed. This can be done before launching Orchestra by changing the file $BPEL_HOME/conf/BPELConfig.xml.

It is also possible to change the configuration at run time (not recommended) using the administration console.

5.1.1 Engine Mode

The Orchestra Engine typically runs using Entity Beans to store all data it is using during the run of processes instances. However, it is possible to switch to a mode in which nothing will be stored. To change this mode, change the config file.

Switch from:

```
<EngineMode mode="DB"/>
to:
<EngineMode mode="Memory"/>
```

5.1.2 Monitoring Mode

It is also possible to change the monitoring mode. To do this, change the following line in the config file:

```
<Monitor mode="RunningOnly"/>
```

The five possible modes and their meaning are as follows:

- **Nothing**: Monitors nothing. This means that it is not possible to see the state of either running instances or the finished ones.
- **RunningOnly**: Monitors only Running Instances. Once the instance is finished, the data is lost.
- **MessageOnly**: The Messages exchanged by the Engine will be stored in files at the end of instances.
- **RunningAndMessage**: Monitors running instances. All exchanged messages are stored in files at the end of the instances; other data is lost.
- **All**: This option will monitor running AND finished instances.

5.1.3 JBI Mode

```
<Jbi mode="off"/>
```

Orchestra will soon be integrated as a JBI Service Engine in ServiceMix. There is an option in the configuration file BPELConfig.xml to turn this function on or off. By default, the value of this paramater is "off".

The capability for the Orchestra Service Engine to be able to plug Orchestra into ServiceMix and other ESBs, such as Petals, will soon be available. Documentation will be provided to explain the configuration steps. At that time it will be possible to change this parameter to "on".

5.2 Orchestra Tuning

The following parameters can be modified to tune the JOnAS server:

- Transaction timeout (jonas.service.jtm.timeout) in $JONAS_BASE/conf/jonas.properties
- Threads number (maxThreads, minSpareThreads, maxSpareThreads)allowed in tomcat container (servlet): $JONAS_BASE/conf/server.xml
- JDBC connection pool configuration: $JONAS_BASE/conf/PostgresSQL1.properties (jdbc.minconpool, jdbc.maxconpool, jdbc.connmaxage, jdbc.maxwaiters ...)
- jonas.service.ejb.maxworkthreads in $JONAS_BASE/conf/jonas.properties should be at least greater than the sum of all mdb max cache size

The following parameters can be modified to tune the database:

Max. connection number allowed in database: /etc/postgresql/postgresql.conf (max_connections, for postgres database).

The following parameters can be modified to tune the BPEL engine:

- Servlet session timeout in tomcat container: web.xml (<session-timeout>30</session-timeout>).
- Number of axis servlet loaded on startup: web.xml.
- WS call timeout: web service implementation (setTimeOut(0)).
- Database sharing in JOnAS (default is not to share the database): deployment descriptor for each entity bean.

5.3 Binding Framework Configuration

The Binding Framework allows for the re-mapping of URLs and the addition of SOAP aware binding components (Sender) that act as web services for systems that do not support web services. This configuration is done by adding entries to the BFConfig.xml file that is located in the $JONAS_BASE/conf directory.

5.3.1 URL Mapping

- Mapping a URL endpoint to a configured "Sender":

```
    .
    <Senders>
        <Sender id='parts'
class='com.supplier.partSOAPSender'/>
    </Senders>
    <Endpoints>
      <Endpoint id='Supplier' href='http://supplier'
altref='bsoa://parts/partinfo'/>
    </Endpoints>
    .
```

This will cause the 'com.test.testSOAPSender' class to be instantiated and the “invoke” method will be passed in the SOAP envelope string input as described in the WSDL. It must return a SOAP envelope string as described in the WSDL for this operation. (This will be discussed in more detail below.)

- Mapping a URL endpoint to an alternate URL:

```
    .

    <Endpoints>

      <Endpoint id='Test' href='http://www.test.com'
altref='http://www.test1.com'/>

    </Endpoints>

    .
```

This will substitute the URL http://www.test1.com for http://www.test.com. It is useful for temporarily sending requests to another URL or, as described below, when used along with the EndpointList statement.

Mapping an endpoint to a list of endpoints:

```
    .

    <Endpoints>

      <Endpoint id='Supplier1' href='http://www.supplier1.com'
altref='http://www.supplier1.com/part1'/>

      <Endpoint id='Supplier' href='http://supplier'
epref='Suppliers'/>

      <EndpointList id='Suppliers' default='SuppliersTest2'>

        <Endpoint id='SuppliersTest1' epref='Supplier1'/>

        <Endpoint id='SuppliersTest2'
altref='http://www.supplier2.com'/>

      </EndpointList>

    </Endpoints>

    .
```

This will substitute the URL http://www.supplier2.com for http://supplier. It is used when there are multiple suppliers. Supplier1 is currently to be used, but this could change; by modifying the configuration file, all the requests will go to a new URL.

5.3.2 Binding Components

A Binding component is a class that either implements com.bull.bsoap.bpel.bindingframework.BFBindingComponent or includes a method with the signature of:

```
/**
 * Invoke the configured component
 * @param sEnv String wsdl defined Request SOAPEnvelope
 * @param params String params from endpoint specified
 * @return String wsdl defined Response record
 * @throws Exception
 */
public String invoke(String sEnv, String params) throws
Exception;
```

The SOAP envelope, corresponding to the WSDL definition, that is sent by AXIS to the configured web service will be passed to the component, along with the parameter that was configured at the end of the URL:

```
bsoa://senderName/param
```

The senderName matches the id as defined in the Senders section of the BFConfig.xml file, and param is the remainder of the URL that will be passed into the invoke method.

CONFIGURATION

Currently, this must be done before executing 'bsoap start'. This restriction will be lifted when a complete configuration tool is available for the Binding Framework.

- Define a Sender in BFConfig.xml.

```
<Senders>

    <Sender id='parts' class='com.supplier.partSOAPSender'/>

</Senders>
```

- Define an Endpoint mapping to the Sender, bsoa must be the protocol used.

```
<Endpoints>

  <Endpoint id='Supplier' href='http://supplier'
   altref='bsoa://parts'/>

</Endpoints>
```

- Build the component and install into the $BPEL_HOME/components directory. This can either be a jar file or the exploded package structure.
- Deploy any supporting jars into the $BPEL_HOME/support directory that are needed by this component.

5.3.3 Deploying a Web Service on .NET

This document aims to explain how to create a web service on .NET. The Demos/College/asyncSeasonTicket web service will be used. The intent is to deploy asyncSeasonTicket on .NET. It will be called from a Java EE server for example.

PROCESS

1. Install Visual Studio 2003 and its prerequisites.
2. Update the PATH environment variable to add the bin directory of the .NET SDK. (often: C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin).
3. Create a source directory for the seasonTicket WS (it will be called sourceST).
4. Copy asyncSeasonTicket.wsdl (see RELATED DOCUMENT #1) in this directory.

Warnings:

This file must be in one part (no import), move the content of the asyncSeasonTicketAbstract.wsdl into asyncSeasonTicket.wsdl.

Consider changing the service URL (put http://localhost/SeasonTicket/Service1.asmx**)**

5. Copy asyncSchoolEnrolment.wsdl in sourceST directory (see RELATED DOCUMENT #2 for the final file).

Warnings:

Remove all import statements.

Remove all types, messages, port types not related to asyncSeasonTicketCallbackPT.

Consider changing the service URL (often http://localhost:9000/axis/services/asyncseasonticketcallbackpt) in the address location of asyncseasonticketcallbackpt port.

6. Open a shell window and go in sourceST.
7. Execute the following command: “wsdl /server asyncSeasonTicket.wsdl”. A file named “AsyncSeasonTicketServiceBP.cs” will be generated.
8. Open Visual Studio.

9. Select “File” -> “New” -> “Project”, select “Project Visual C#” in the type and “ASP.NET Web Service” in the template. In the “Location” text box, write “http://localhost/SeasonTicket”. Click on OK.

10. Double Click on ’Form1’ to edit the C# code.

11. Go to the “View” menu and choose “Solution Explorer”. It appears on the right.

12. In the Solution Explorer, expand the applicaton tree.

13. Select “File” -> “Add Existing Item”. Browse and choose sourceSt\AsyncSeasonTicketServiceBP.cs.

14. In the solution explorer, right-click on “references” and select “Add a Web Reference”. In the URL field, enter the URL to sourceST\asyncSchoolEnrolment.wsdl (e.g: file:\\D:\SeasonTicketSources\asyncSchoolEnrolment.wsdl). Click on “go”. Available operations must appear. On the right bottom of the pop up window, in the field “Web reference name:”, enter “enrollment”. Then click on “AddReference”.

15. Modify Service1.asmx.cs file in Visual Studio.
    - On the top, add “using SeasonTicket.enrolment;”
    - Just before the class declaration (public class Service1), add: “[WebService(Namespace=”http://bsoap.bull.com/bpel/demo/asyncSeasonTicket")].
    - Modify the class declaration by changing “System.Web.Services.WebService” to “AsyncSeasonTicketServiceBP”.
    - Add the implementation of the “initiate” operation (see RELATED DOCUMENT #3).

16. Remember to configure proxies of both the BPEL server and the .NET server.

17. Deploy the web service: go to the “Debugr” menu and select “Start”.

NOTES

1. Changing the URL of the callback
   - In the solution explorer, open webReferences -> enrollement -> asyncSchoolEnrolment.wsdl and enter the new URL.
   - In the solution explorer, open webReferences -> enrollement -> reference.map -> reference.cs and enter the new URL.
   - Deploy the web service.
2. Tracing the .NET web service execution

To see if the web service is called, add traces in the "initiate" method (see related RELATED DOCUMENT #4).

RELATED DOCUMENT #1: asyncSeasonTicket.wsdl

```
<?xml version="1.0"?>
<definitions
targetNamespace="http://bsoap.bull.com/bpel/demo/asyncSeasonTicket"
      xmlns:tns="http://bsoap.bull.com/bpel/demo/asyncSeasonTicket"
      xmlns:college="http://bsoap.bull.com/bpel/demo/college"
      xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-
      process/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns="http://schemas.xmlsoap.org/wsdl/" >

 <types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://bsoap.bull.com/bpel/demo/asyncSeasonTicket">
   <complexType name="asyncSeasonTicketRequest">
    <sequence>
     <element name="SchoolBusDeparture" type="xsd:string" />
     <element name="SchoolBusArrival" type="xsd:string" />
     <element name="SubsidyAgreement" type="xsd:string" />
    </sequence>
   </complexType>
  </schema>

  <schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://bsoap.bull.com/bpel/demo/college">
   <complexType name="recordInfoType">
    <sequence>
     <element minOccurs="0" maxOccurs="1" name="SchoolRecordNb"
     type="string" />
     <element minOccurs="0" maxOccurs="1" name="Name" type="string" />
     <element minOccurs="0" maxOccurs="1" name="FirstName"
     type="string" />
     <element minOccurs="0" maxOccurs="1" name="Address" type="string" />
    </sequence>
   </complexType>
  </schema>
 </types>

 <message name="asyncSeasonTicketRequestMessage">
  <part name="recordInfo" type="college:recordInfoType"/>
  <part name="seasonTicketRequest" type="tns:asyncSeasonTicketRequest"/>
 </message>
```

```
 <portType name="asyncSeasonTicketPT">
  <operation name="initiate">
   <input message="tns:asyncSeasonTicketRequestMessage"/>
  </operation>
 </portType>


 <binding name="AsyncSeasonTicketPTSOAPBinding"
 type="tns:asyncSeasonTicketPT">
  <soap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="initiate" >
   <soap:operation
   soapAction="http://bsoap.bull.com/bpel/demo/asyncSeasonTicket"/>
   <input>
    <soap:body use="encoded" namespace=""
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </input>
  </operation>
 </binding>


 <service name="AsyncSeasonTicketServiceBP">
  <port name="asyncSeasonTicketPT"
  binding="tns:AsyncSeasonTicketPTSOAPBinding">
   <soap:address
   location="http://localhost/SeasonTicket2/Service1.asmx"/>
  </port>
 </service>

</definitions>
```

RELATED DOCUMENT #2: asyncSchoolEnrolment.wsdl

```
<?xml version="1.0"?>
<definitions
targetNamespace="http://bsoap.bull.com/bpel/demo/asyncSchoolEnrolment"
      xmlns:tns="http://bsoap.bull.com/bpel/demo/asyncSchoolEnrolment"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns="http://schemas.xmlsoap.org/wsdl/" >

 <types>
  <schema xmlns=http://www.w3.org/2001/XMLSchema
  targetNamespace="http://bsoap.bull.com/bpel/demo/asyncSchoolEnrolment">
   <complexType name="asyncSeasonTicketResponse">
    <sequence>
     <element name="SchoolRecordNb" type="xsd:string"/>
     <element name="SchoolBusCompanyResponse" type="xsd:string"/>
     <element name="BusCompanyRecordNb" type="xsd:string"/>
    </sequence>
   </complexType>
  </schema>
 </types>

 <message name="asyncSeasonTicketResponseMessage">
  <part name="seasonTicketResponse"
  type="tns:asyncSeasonTicketResponse"/>
 </message>
```

```
 <portType name="asyncSeasonTicketCallbackPT">
  <operation name="onResult">
   <input message="tns:asyncSeasonTicketResponseMessage"/>
  </operation>
 </portType>

 <binding name="AsyncSeasonTicketCallbackPTSOAPBinding"
 type="tns:asyncSeasonTicketCallbackPT">
  <soap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="onResult" >
   <soap:operation
   soapAction="http://bsoap.bull.com/bpel/demo/asyncSchoolEnrolment"/>
   <input>
    <soap:body use="encoded" namespace=""
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </input>
  </operation>
 </binding>

 <service name="AsyncSeasonTicketServiceCallbackBP">
  <port name="asyncSeasonTicketCallbackPT"
  binding="tns:AsyncSeasonTicketCallbackPTSOAPBinding">
   <soap:address location=
   "http://localhost:9000/axis/services/asyncSeasonTicketCallbackPT"/>
  </port>
 </service>

</definitions>
```

RELATED DOCUMENT #3: "initiate" operation

```
[WebMethod(MessageName="asyncSeasonTicketRequestMessage")]
public override void initiate(recordInfoType recordInfo,
asyncSeasonTicketRequest seasonTicketRequest)
{
  string busStop = seasonTicketRequest.SchoolBusDeparture;

  asyncSeasonTicketResponse str = new asyncSeasonTicketResponse() ;
  str.SchoolRecordNb = recordInfo.SchoolRecordNb ;
  str.BusCompanyRecordNb = "" ;
  str.SchoolBusCompanyResponse = "No BusStop with this Name : " + busStop ;

  bool found = false;
  found = found || busStop.ToLower().Equals("grenoble");
  found = found || busStop.ToLower().Equals("eybens");
  found = found || busStop.ToLower().Equals("fontaine");
  found = found || busStop.ToLower().Equals("bresson");
  found = found || busStop.ToLower().Equals("echirolles");
  found = found || busStop.ToLower().Equals("jarrie");

  if (found)
  {
    str.SchoolBusCompanyResponse = "OK";
    String rep = "BC" ;
    rep = recordInfo.SchoolRecordNb + rep ;
    str.BusCompanyRecordNb = rep ;
  }
  AsyncSeasonTicketServiceCallbackBP as1=new
  AsyncSeasonTicketServiceCallbackBP();
  as1.onResult(str);

}
```

RELATED DOCUMENT #4: tracing .NET web service execution

Add in Service1.asmx.cs file the method:

```
public void write(string st)
{
  string filename = "d:\\tmp.txt";
  StreamWriter sw;
  if (File.Exists(filename))
  {
    sw = new StreamWriter(filename);
  }
  else
  {
    sw = File.CreateText(filename);
  }
  sw.WriteLine(st+" : "+System.DateTime.Now.ToString());
  sw.Close();
}
```

Replace “d:\\tmp.txt”, with your own path.

Add “using System.IO;” onto the top of the class.

Call this method in the initiate method (e.g: write “before invoke AsyncSeaseanTicketCallBack”). This method only keeps the last trace, not all traces.

Technical publication remarks form

Title:	BSOA Orchestra

Reference No.:	86 A2 53ER 01

Date:	November 2006

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME: _____ Date: _____

COMPANY: _____

ADDRESS: _____

_____

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation Dept.
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone: +33 (0) 2 41 73 72 66
FAX: +33 (0) 2 41 73 70 66
E-Mail: srv.Dupilcopy@bull.net

CEDOC Reference #	Designation	Qty
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
[_ _] : The latest revision will be provided if no revision number is given.		

NAME: ______________________________ Date: ____________

COMPANY: ______________________________

ADDRESS: ______________________________

____

PHONE: ____________________ FAX: ____________________

E-MAIL: ______________________________

For Bull Subsidiaries:
Identification: ______________________________

For Bull Affiliated Customers:
Customer Code: ______________________________

For Bull Internal Customers:
Budgetary Section: ______________________________

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE No.
86 A2 53ER 01