BPEL orchestration demonstrator

Implementing and running the client

Tools & Prerequisite

Needed tools for developing and testing are:

- Java JDK 1.5
- Apache Ant 1.7.0
- Celtix & Axis 1.4
- Orchestra 3.0.3 and its prerequisites
- Optional : IDE Eclipse 3.2 & resin 2.1.14

If you use your own way to develop and launch a client for the booking service, these tools may be unnecessary.

In this document, we will describe each step:

- Unit testing of called WS
- Simple BPEL orchestration testing
- Building a client for the "booking" service

Attachments

Some ZIP archives should come with this tutorial:

- *"HotelBooking & RestaurantBooking" sources*
  - Online.zip: WSDL shared files
  - src.zip: both services sources
  - ClientSrc.zip: WS testing sources
- *The « BPEL & Orchestra » part*
  - bsoa.zip: Files to be deployed in Orchestra
- *The « Client » part*
  - DemoBPELOrchestraTest.zip: Eclipse Workspace of the unit-test client
  - DemoBPELOrchestraClient.zip: Eclipse Workspace of the demo client
  - resin-booking.zip: Built servlet (webapp in server-side)

Environment initialisation

You will find in annexes scripts allowing to initialize the environment to run examples of this document. According to the where your tools are located, those scripts may have to be modified.

Testing services

Unit-tests of "HotelBooking & RestaurantBooking" services

We can generate simple clients to test these WS calling thanks to Celtix:

```
mkdir hotelbookingclient
wsdl2java -p org.ow2.jones.demo.booking.service.hotelbooking -client -d
hotelbookingclient -ant
http://chlore.inrialpes.fr:8080/hotelbooking/celtix/hotelbooking

mkdir restaurantbookingclient
wsdl2java -p org.ow2.jones.demo.booking.service.restaurantbooking
-client -d restaurantbookingclient -ant
http://chlore.inrialpes.fr:8080/restaurantbooking/celtix/restaurantbooki
ng
```

Then you need to modify default implementations before to launch tests. You will find in the "ClientSrc.zip" a sample of client implementation for "HotelBooking" & "RestaurantBooking" services. To build Java sources and launch test, you just have to run the following commands:

```
cd restaurantbookingclient
ant RestaurantBooking.Client
cd ..
cd hotelbookingclient
ant HotelBooking.Client
```

Java classes are the compiled and CeltixRun will execute client code with the good environment. Results are display on the console; in our sample, you should obtain the following result:

```
Invoking bookRestaurant...
  0. Invoking with arg "NULL"
javax.xml.ws.WebServiceException: Could not set parts into wrapper
element
    [...]
    NULL
  1. Invoking with arg ""
    Result:  true
    Price:   50
    Message: Restaurant successfully booked (full price)
  2. Invoking with arg "HalfPrice"
    Result:  true
    Price:   25
    Message: Restaurant successfully booked (half price)
  3. Invoking with arg "FreeWithHotel"
    Result:  true
    Price:   0
    Message: Restaurant successfully booked (for free)
  4. Invoking with arg "BadPromoCode"
    Result:  false
    Price:   0
    Message: Error booking restaurant

Invoking bookHotel...
  0. Invoking with arg "0"
    Result:  false
    Price:   0
    Message: Error booking hotel: You have to book 1 room at least
    RefCode:
```

```
1. Invoking with arg "1"
  Result:  true
  Price:   100
  Message: 1 rooms sucessfully booked
  RefCode: HalfPrice
2. Invoking with arg "2"
  Result:  true
  Price:   200
  Message: 2 rooms sucessfully booked
  RefCode: HalfPrice
3. Invoking with arg "3"
  Result:  true
  Price:   300
  Message: 3 rooms sucessfully booked
  RefCode: FreeWithHotel
4. Invoking with arg "9"
  Result:  true
  Price:   900
  Message: 9 rooms sucessfully booked
  RefCode: FreeWithHotel
5. Invoking with arg "10"
  Result:  false
  Price:   0
  Message: Error booking hotel: Only 9 rooms available
  RefCode:
```

BPEL orchestration testing

For testing, we will use Bull Orchestra BPEL engine.

We will describe here the whole manipulation, from Orchestra installation to “Booking” BPEL process deployment and test in the “Orchestra Web Console”.

Prerequisite for Orchestra install

Orchestra install procedure is available at the following URL: http://wiki.orchestra.objectweb.org/xwiki/bin/view/Main/Documentation. It needs JDK, JOnAS and Apache Ant. The JOnAS install procedure is available here: http://jonas.objectweb.org/current/doc/howto/install_j2ee.html; this one require BCEL.

Notice: sometimes I got an error message saying that « org.apache.tools.ant.launch.AntMain » class was not found. A possible by-pass is to copy “ant-launcher.jar” Ant Jar file to « ${JONAS_BASE}/lib/ext/ » Orchestra folder.

Needed files for BPEL orchestration testing

Importing a BPEL process in Orchestra requires at least 2 files: WSDL & BPEL service description. In our example, we need 4 files (available in the “bsoa.zip” archive):

- booking.bpel : BPEL process description;
- booking.wsdl : interface allowing to expose the BPEL process as a Web-Service and to call it from the Orchestra Web Console (or from another client);
- hotelbooking.wsdl & restaurantbooking.wsdl : WSDL of WS directly interacting with the BPEL process (that is to say referenced in the BPEL file as partner links).

To add the “booking” BPEL process, we can add a “booking” forlder in the “<BPEL_HOME>/samples” directory, and copy inside the 4 previously mentioned files.

Deploying the BPEL process

There are two ways to deploy the BPEL process: command line or Web Console.

Here's the command line to enter under Windows OS:

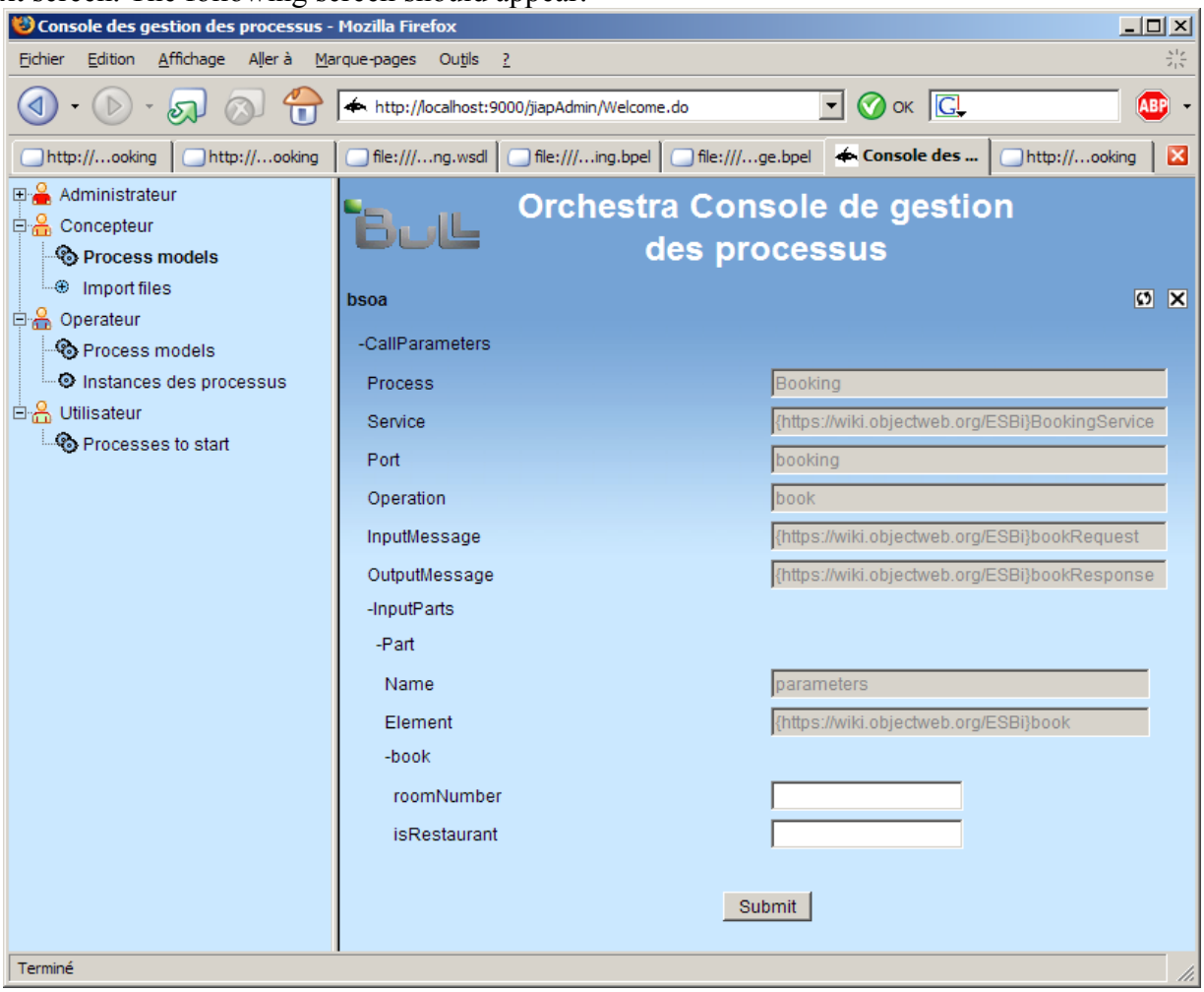
```
REM Start the BSOA server if needed
bsoap start

REM (Re)deploy the booking process
bsoap deploy -samples -p booking
%BPEL_HOME%\samples\booking\hotelbooking.wsdl
%BPEL_HOME%\samples\booking\restaurantbooking.wsdl
```

The second way is to use the Orchestra Web Console available at the floowing URL: http://localhost:9000/jiapAdmin/ (most generally http://<host>:<port>/jiapAdmin/ depending of your Orchestra installation). Then a way to deploy "booking" process is:

- Menu "Conceptor" / "Import Files"
- Process bpel file : <BPEL_HOME>/samples/booking.bpel
- Process wsdl file : <BPEL_HOME>/samples/booking.wsdl
- External wsdl file : <BPEL_HOME>/samples/hotelbooking.wsdl and <BPEL_HOME>/samples/restaurantbooking.wsdl
- Select "Import"
- Menu "Operator/Process Models" : Click on the orange triangle of the booking line.

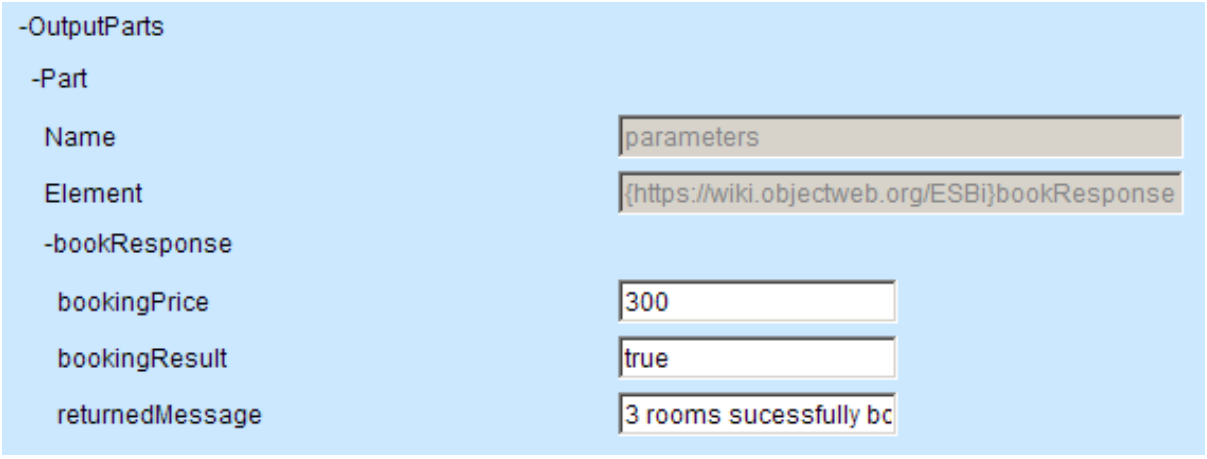
Starting/testing the BPEL process

We use the Orchestra Web Console (http://localhost:9000/jiapAdmin/). In the menu "Operator / Process Models", ilwe click on the green, triangle of the "booking" line, and do the same in the next screen. The following screen should appear:



This screen is generated from the WSDL, and so is waiting for “roomNumber” and “isRestaurant” input parameters.

If everything is OK, when click on the “Submit” button, the form is updated (“OutputParts”) and the response is displayed (“bresponse” variable, see code source BPEL):



Building a client for the “Booking” service

Once the BPEL booking process successfully deployed in Orchestra (or another BPEL engine), we can test it outside of the Orchestra Web Console in many different ways: web pages and javascript, Java stand-alone program, Servlet, and more generally with anything able to call a WS.

Building a simple test client

We can simply generate a Java client with Celtix and “Booking” WSDL service (available in the “bsoa.zip” archive). From a folder containing the booking.wsdl file, just enter the following command line:

```
wsdl2java -client -p org.ow2.jones.demo.booking.client -ant booking.wsdl
```

that will generate in the current folder Java sources of a client. Then you have just to modify the Java class « org.ow2.jones.demo.booking.client.BookingClient ». To start the test, type:

```
ant Booking.Client
```

The “DemoBPELOrchestraTest.zip” archive contains an implementation example of “BookingClient.java”. A possible result (depends of the content of “booking.bpel” deployed in Orchestra) may be the following:

```
>ant Booking.Client
Buildfile: build.xml

compile:
    [mkdir] Created dir: E:\INRIA-OW\Projets\tmpworkspace\DemoBPELOrchestraClient\src\build\classes
    [javac] Compiling 7 source files to E:\INRIA-
OW\Projets\tmpworkspace\DemoBPELOrchestraClient\src\build\classes

Booking.Client:
     [java] Invoking book...
     [java] 0;true : 50;false;Error booking hotel: You have to book 1 room at least - Restaurant
successfully booked (full price)
     [java] 1;true : 150;true;1 rooms sucessfully booked - Restaurant successfully booked (full price)
     [java] 2;true : 250;true;2 rooms sucessfully booked - Restaurant successfully booked (full price)
     [java] 3;true : 300;true;3 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 4;true : 400;true;4 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 5;true : 500;true;5 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 6;true : 600;true;6 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 7;true : 700;true;7 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 8;true : 800;true;8 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 9;true : 900;true;9 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 10;true : 50;false;Error booking hotel: Only 9 rooms available - Restaurant successfully
booked (full price)
     [java] 0;false: 0;false;Error booking hotel: You have to book 1 room at least - No restaurant
booked
     [java] 1;false: 100;true;1 rooms sucessfully booked - No restaurant booked
     [java] 2;false: 200;true;2 rooms sucessfully booked - No restaurant booked
     [java] 3;false: 300;true;3 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 4;false: 400;true;4 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 5;false: 500;true;5 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 6;false: 600;true;6 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 7;false: 700;true;7 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 8;false: 800;true;8 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 9;false: 900;true;9 rooms sucessfully booked - Restaurant successfully booked (for free)
     [java] 10;false: 0;false;Error booking hotel: Only 9 rooms available - No restaurant booked

BUILD SUCCESSFUL
Total time: 18 seconds
```

Implementing a client for the demonstration

We choose to implement the client shown in the demonstration through a Java Servlet deployed in the “resin” Servlet engine. We will use a mix of web technologies such as XHTML, XForms, CSS, XSLT...

Here's the step by step description of the implementation process.

1. **Generating Java code calling the “booking” service**

We use here AXIS to generate the client (for unknown reason and a lack of debugging time, we did not use Celtix, because CELTIX_HOME environment variable was not correctly initialized when running in a Resin Servlet context). The command line is the following one:

```
cd <client_folder>
java org.apache.axis.wsdl.WSDL2Java -p org.ow2.jones.demo.booking.client
http://s4allsdk.objectweb.org/demo/booking/booking.wsdl
```

2. « resin » install

We used an open source servlet engine called resin (version 2.1.14) available here: http://www.caucho.com/download/index.xtp. To install it, unzipping the downloaded archive is enough. Then servlets have to be deployed in the "<RESIN_HOME>/webapps" folder.

The "resin-booking.zip" archive contains a "server-side" image of the "booking" service client, and which has to be uncompressed in the "<RESIN_HOME>/webapps" folder.

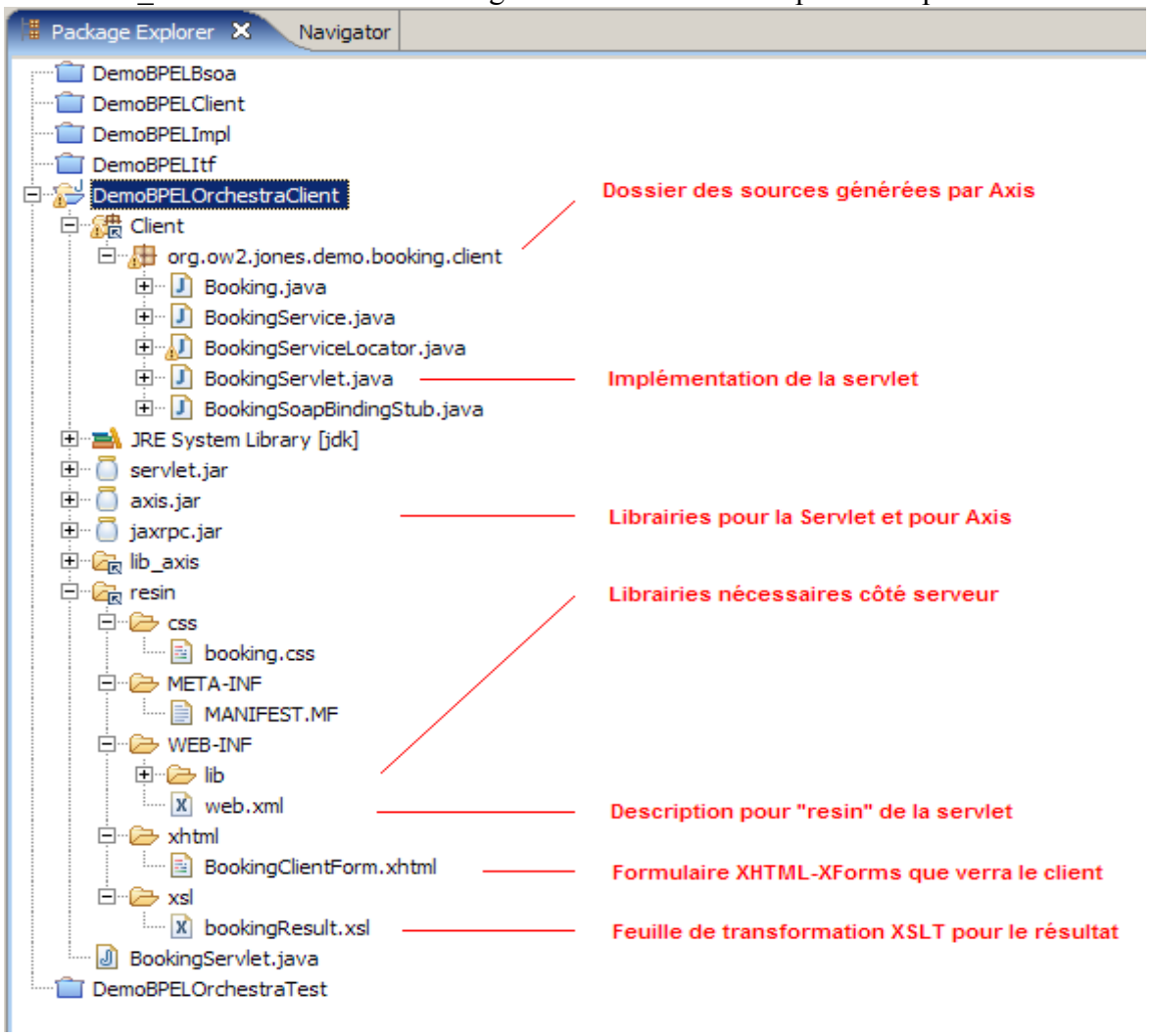
3. Implementing the client with Eclipse

This step is not compulsory because the "resin-booking.zip" contains a build server-side image of the "booking" client.

If you want to modify sources using Eclipse, use the workspace included in the "DemoBPELOrchestraClient.zip" archive, but modify the ".project" file before opening the workspace into IDE:

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
  <name>DemoBPELOrchestraClient</name>
  <comment></comment>
  <projects>
  </projects>
  <buildSpec>
    <buildCommand>
      <name>org.eclipse.jdt.core.javabuilder</name>
      <arguments>
      </arguments>
    </buildCommand>
  </buildSpec>
  <natures>
    <nature>org.eclipse.jdt.core.javanature</nature>
  </natures>
  <linkedResources>
    <link>
      <name>resin</name>
      <type>2</type>
      <location><RESIN_HOME>/webapps/booking</location>
    </link>
    <link>
      <name>lib_axis</name>
      <type>2</type>
      <location><AXIS_HOME>/lib</location>
    </link>
    <link>
      <name>Client</name>
      <type>2</type>
      <location><SRC_HOME></location>
    </link>
  </linkedResources>
</projectDescription>
```

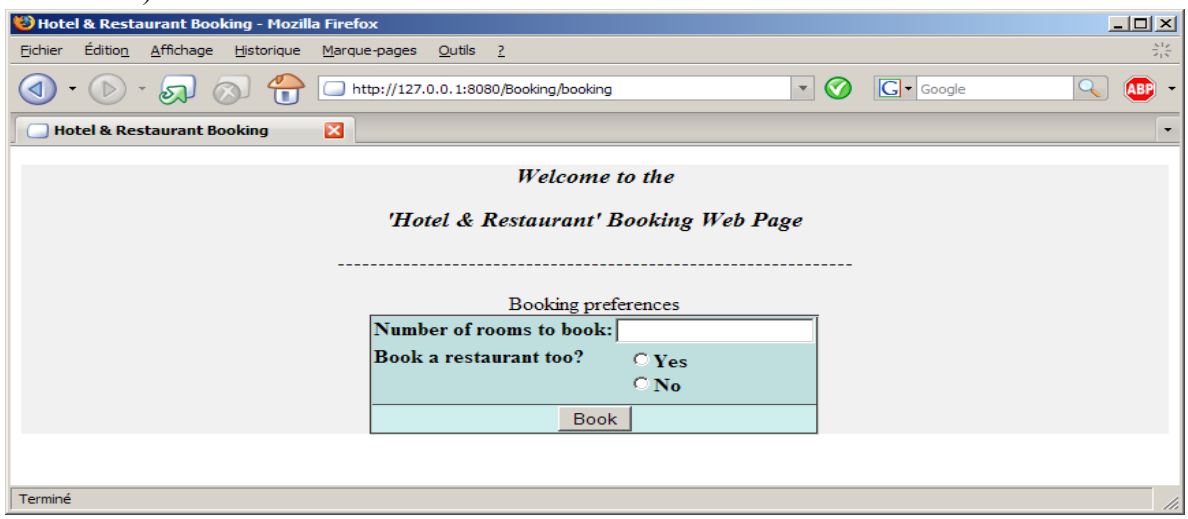
<RESIN _HOME> means the folder where “resin” is installed in, <AXIS_HOME> the Axis one, and <SRC_HOME> the one containing Java sources. The Eclipse workspace should looks like



the following annotated screen-shot. Note that using the “auto-build” option in Eclipse, any saved modification in Java sources are automatically deployed in “Resin”.

4. Acceding the client

If “Resin” is installed locally and started (see “<RESIN_HOME>\bin\httpd”), you can access the servlet at the following URL: http://127.0.0.1:8080/Booking/booking (according to the “web.xml” file).



Before testing, check that “HotelBooking, RestaurantBooking and Booking” are correctly deyed. If the form does not appear correctly, it may be because your web browser does not support XHTML/XForms (for Firefox, use the plug-in « Xforms for Mozilla » available here: https://addons.mozilla.org/firefox/824/).

Running the demonstration

Notice that the “HotelBooking” Web-Service returns a “refCode” output parameter that is not used in the BPEL orchestration demonstration.

Scenario

The theme is a room/restaurant booking service. This service is implemented thanks to a BPEL process itself calling and orchestrating two external WS: HotelBooking & RestaurantBooking.

The idea is to modify the business logic, and to make this change very easy thanks to agility given by BPEL orchestration. So we now decide to implement the following rule:

“From now, if a client books successfully 3 rooms at least, a restaurant is automatically booked too for free”.

Setting the environment

Before starting the demo, you have just to check that “HotelBooking & RestaurantBooking” services are running, that Orchestra is started and the “booking” BPEL process deployed inside, and finally “Resin” servlet engine. Finally, check that the form is correctly displayed at the servlet URL.

The demonstration step by step

For more details, please refer the web page explaining the demonstration: http://chlore.inrialpes.fr:8080/bpeldemo. It is also available off-line in “OnlineDemo.zip” archive.

Annexes

Initializing Celtix/Altix environment

For Windows OS

```
@echo off

REM Depends on the environment
REM -------------------------
@set APP_HOME=E:\INRIA-OW\Projets\JOnES\Demo-07.02\WS
@set JAVA_HOME=C:\Sun\SDK\jdk
@set ANT_HOME=C:\apache-ant-1.7.0
@set CELTIX_HOME=%APP_HOME%\..\tools\celtix
@set AXIS_HOME=%APP_HOME%\..\tools\axis-1_4

REM For everyone
REM ------------
@set AXIS_LIB=%AXIS_HOME%\lib
@set AXISCLASSPATH=%AXIS_LIB%\axis.jar;%AXIS_LIB%\commons-discovery-
0.2.jar;%AXIS_LIB%\jaxrpc.jar;%AXIS_LIB%\saaj.jar;%AXIS_LIB%\wsdl4j-
1.5.1.jar;%AXIS_LIB%\xmlapis.jar;%AXIS_LIB%\xercesImpl.jar
@set PATH=%PATH%;%JAVA_HOME%\bin;%ANT_HOME%\bin;%CELTIX_HOME%\bin
@set
CLASSPATH=.;%CELTIX_HOME%\lib\celtix.jar;%AXISCLASSPATH%;%CLASSPATH%
```

Updating WSDL for Orchestra

You can notice that WSDL in “online.zip” archive (generated with altix/celtix from “HotelBooking” and “RestaurantBooking” source code) differs a bit from those contained in “bsoa.zip” (deployed in Orchestra). Indeed, with original generated WSDL, Orchestra will lead at runtime to errors such as:

```
java.lang.ClassNotFoundException: org.objectweb.wiki.ESBi.BookType
```

To avoid this, we have to modify the “wsdl:types” part of WSDL file in order to give a name to complex type used in IO parameters. For example, for hotelbooking.wsdl:

```
<schema elementFormDefault="qualified"
        targetNamespace="https://wiki.objectweb.org/ESBi">
 <element name="bookHotel">
  <complexType>
   <sequence>
    <element name="roomNumber" type="xsd:int"/>
   </sequence>
  </complexType>
 </element>
 <element name="bookHotelResponse">
  <complexType>
   <sequence>
    <element name="bookHotelReturn" type="impl:HotelResponse"/>
   </sequence>
  </complexType>
 </element>
 <complexType name="HotelResponse">
  <sequence>
   <element name="bookingPrice" type="xsd:int"/>
   <element name="bookingResult" type="xsd:boolean"/>
   <element name="refCode" nillable="true" type="xsd:string"/>
   <element name="returnedMessage" nillable="true" type="xsd:string"/>
  </sequence>
 </complexType>
```

```
</schema>
```

we will modify it to:

```
<schema elementFormDefault="qualified"
        targetNamespace="https://wiki.objectweb.org/ESBi">
 <complexType name="bookHotelType">
  <sequence>
   <element name="roomNumber" type="xsd:int"/>
  </sequence>
 </complexType>
 <complexType name="bookHotelReturnType">
  <sequence>
   <element name="bookingPrice" type="xsd:int"/>
   <element name="bookingResult" type="xsd:boolean"/>
   <element name="returnedMessage" nillable="true" type="xsd:string"/>
   <element name="refCode" nillable="true" type="xsd:string"/>
  </sequence>
 </complexType>
 <complexType name="bookHotelResponseType">
  <sequence>
   <element name="bookHotelReturn" type="impl:bookHotelReturnType"/>
  </sequence>
 </complexType>
 <element name="bookHotel" type="impl:bookHotelType"/>
 <element name="bookHotelResponse" type="impl:bookHotelResponseType"/>
</schema>
```