PEtALS Use Cases User's Guide 1

This document explain how to use the usecases in the PEtALS environment.

EBM WebSourcing
Christophe HAMERLING <christophe.hamerling@ebmwebsourcing.com>

- July 2007 -

Table of Contents

List of Figures

PETALS-USECASES

This document explains how to use the usecases provided with PEtALS. It describes simple scenarii which can be used to better understand the JBI philosophy...

Chapter 1. Getting started

1.1. Prerequisiteries

To use these usecases, you must at least get the following tools :

- PEtALS : The ObjectWeb open source ESB (http://petals.objectweb.org).
- Apache ant: Used to install components, deploy service assemblies and lauch sample clients (http://ant.apache.org).
- Apache Tomcat: Used to install Web Services which will be called by PEtALS services (http://tomcat.apache.org).
- SOAPUI : A SOAP application to test Web Services (http://www.soapui.org)

If you want to build usecases from sources, you also need to use Maven (http://maven.apache.org).

The current document is Linux user oriented, but it can be easily understood by Windows users.

1.2. Generating usecases from sources

The usecases are located under the `<PETALS_SRC>/petals-demos/petals-usecases` directory, where <PETALS_SRC> is the root directory where you have checked out the sources of the project.

You can generate the usecases with maven as usual :

cd <PETALS_SRC>/petals-demos/petals-usecases

mvn clean; mvn

To build an assembly of a usecase (a packaged directory with all the required artifacts), you must go into the usecase directory and generate it :

cd petals-<usecase>

mvn clean package assembly:assembly

The usecase with all the needed components, service assemblies, clients and scripts are available under the `target` directory of the usecase.

1.3. Getting usecases

Usecases are available for download as archives from the PEtALS forge download page (http://forge.objectweb.org/project/showfiles.php?group_id=213 under the petals-usecases section).

The usecase structure is generally:

```
build.xml
taskdef.properties

client/
  petals-*-client.jar
  *.jar

deployables/
  petals-bc-*.zip
  sa-*.zip
```

1.4. Launching usecases

Each usecase comes with its own Ant script.

To install/deploy artifacts, generally you just have to call the install task (or just call Ant with no argument):

ant install

If a usecase comes with a client application, you will have to call the client task to run the program:

ant client

To clean the platform (stop and uninstall all components and service assemblies), you can call the clean task:

ant clean

Note

Special calls are detailled for each usecase.

Chapter 2. Use cases

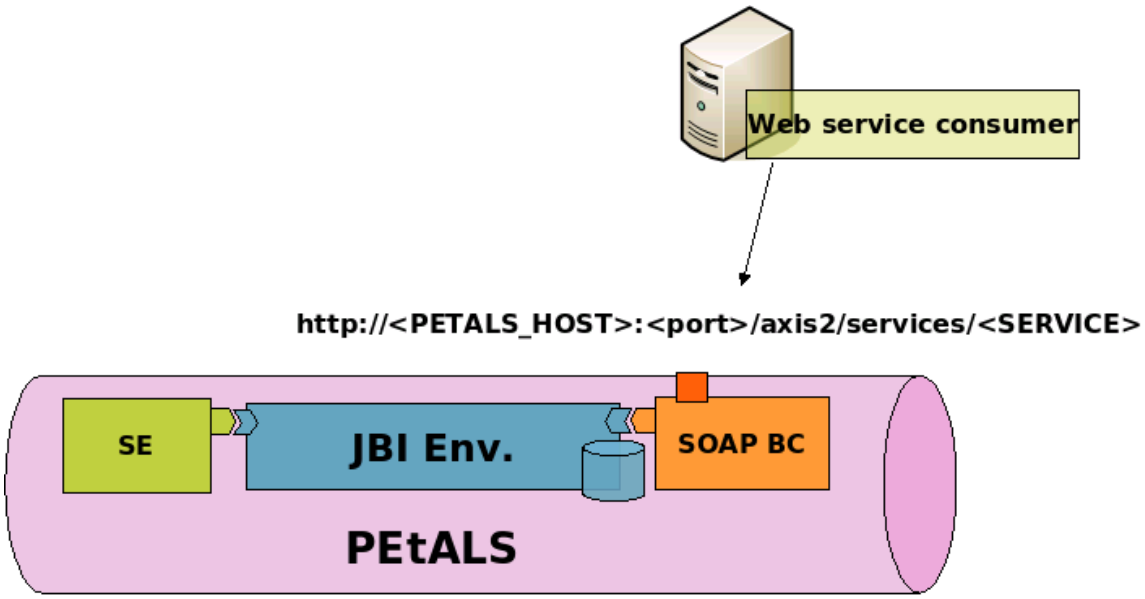
2.1. Expose a JBI service as Web Service

This use case demonstrates how to expose a JBI service as Web Service which can be called from external web service consumers. The current usecase expose an helloworld service provided by the petals-sample-helloworld component. This usecase can be extended to other JBI services.

To be able to consume to handle incoming Web Services calls, we need to use the SOAP binding component (petals-bc-soap) and deploy service assembly (SA) to configure it.

The JBI oriented representation of this use case is :

Figure 2.1. Exposing a JBI service as Web Service



2.1.1. Setting up the platform

1. Start PEtALS (if not already started)
2. Install components (petals-bc-soap, petals-sample-helloworld), deploy service assembly and start these artifacts. This can be done with the Ant script:

   ant

The deployed Service Assembly contains :

1. A service unit which will be deployed on the petals-sample-helloworld to provide fixed endpoint/service/interface values.
2. A service unit which will be deployed on the petals-bc-soap to consume the provided helloworld service. When consuming a JBI service with the SOAP binding component, a Web Service is created and is exposed to the outside (please refer to the petals-bc-soap documentation for more details).

A sample client is available to send a SOAP message to the exposed Web Service, it calls the sayHello operation on the exposed Web Service. The SOAP binding component will transform and forward the incoming SOAP message to the JBI service (helloworld).

2.1.2. Usage

Launch the provided client from the Ant script:

ant client

This client uses the Axis2 client libraries to invoke the Web Service provided by the SOAP binding component.

As result, the client should print a message like this :

```
<sayHelloResponse xmlns="http://org.objectweb.petals/">
  <sayHelloReturn>you told me : This is the input data</sayHelloReturn>
</sayHelloResponse>
```

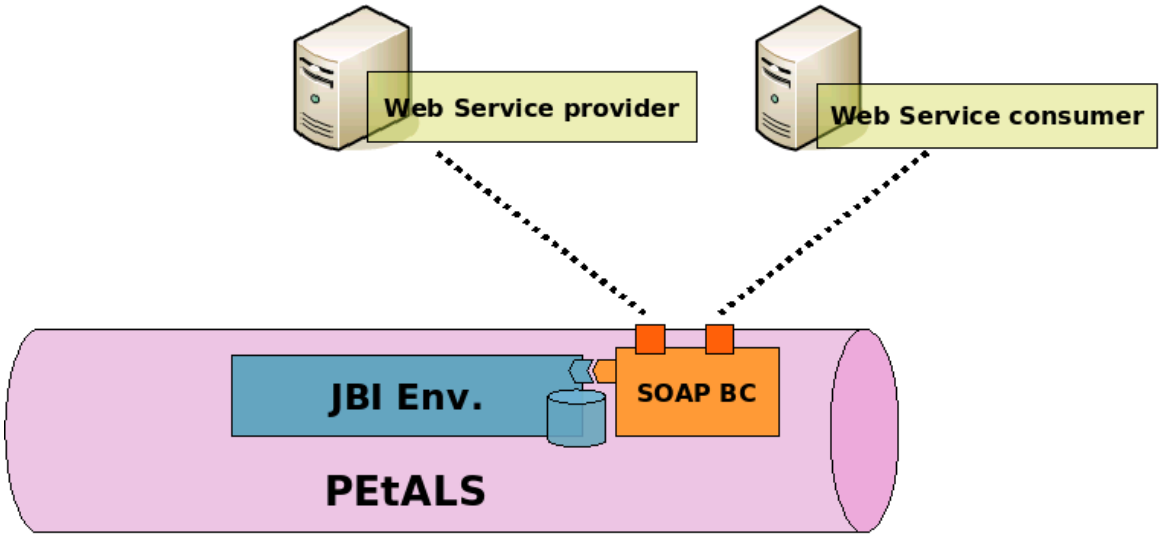
2.2. Use PEtALS as Web Service Proxy

This use case demonstrate how you can PEtALS to proxying Web Services.

The current usecase consumes an external Web Service and expose it as a JBI service in the bus. This provided service is consumed by an JBI service and exposed to external Web Services clients.

The JBI oriented representation of this usecase is :

Figure 2.2. Proxying Web Service with PEtALS



2.2.1. Setting up the platform

1. Install the Web Archive providing the echo Web Service (web-service-sample.war) into Tomcat.

   Depending on your Tomcat container, the Web Service should be available on `http://<HOST>:<PORT>/web-service-sample/services/SimpleServiceImpl`

   In the current usecase, we suppose that Tomcat is launched on the 8080 port of the localhost. Change these values if needed.

2. Start PEtALS (if not already started...)

3. Install components (petals-bc-soap), deploy service assembly and start these artifacts. This can be done with the Ant script:

ant

The deployed service assembly contains:

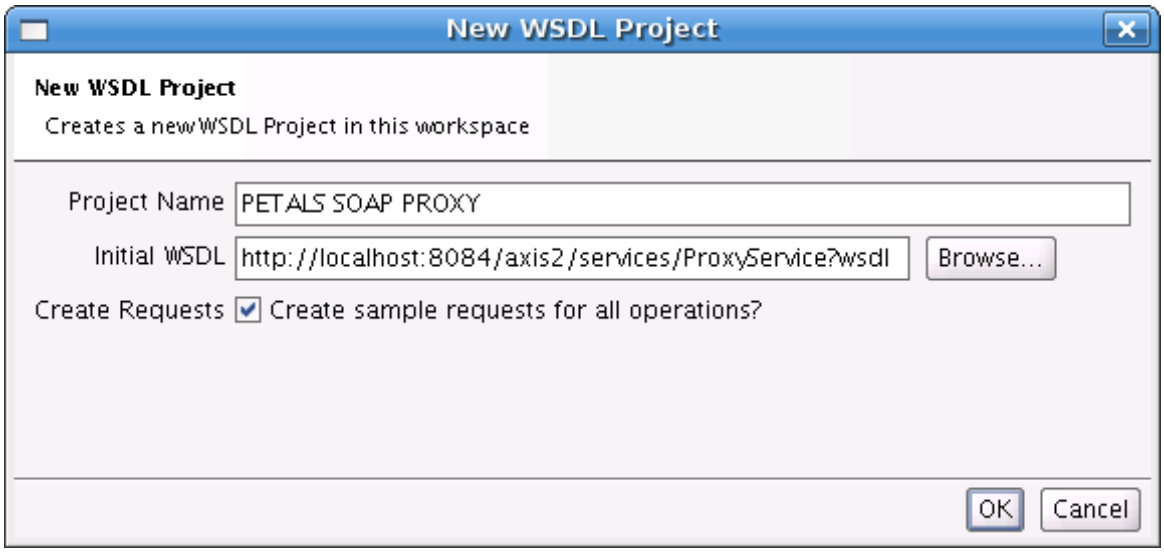
1. A service unit which will be deployed on the petals-bc-soap to consume an external Web Service and then provide it as a JBI service.

2. A service unit which will be deployed on the petals-bc-soap to consume the JBI service provided by the previous deployed service unit.

2.2.2. Usage

As a client, we use soapUI. To be able to invoke the Web Service, you must create a new WSDL project with the PEtALS based Web Service address. Modify the host and the port is required.

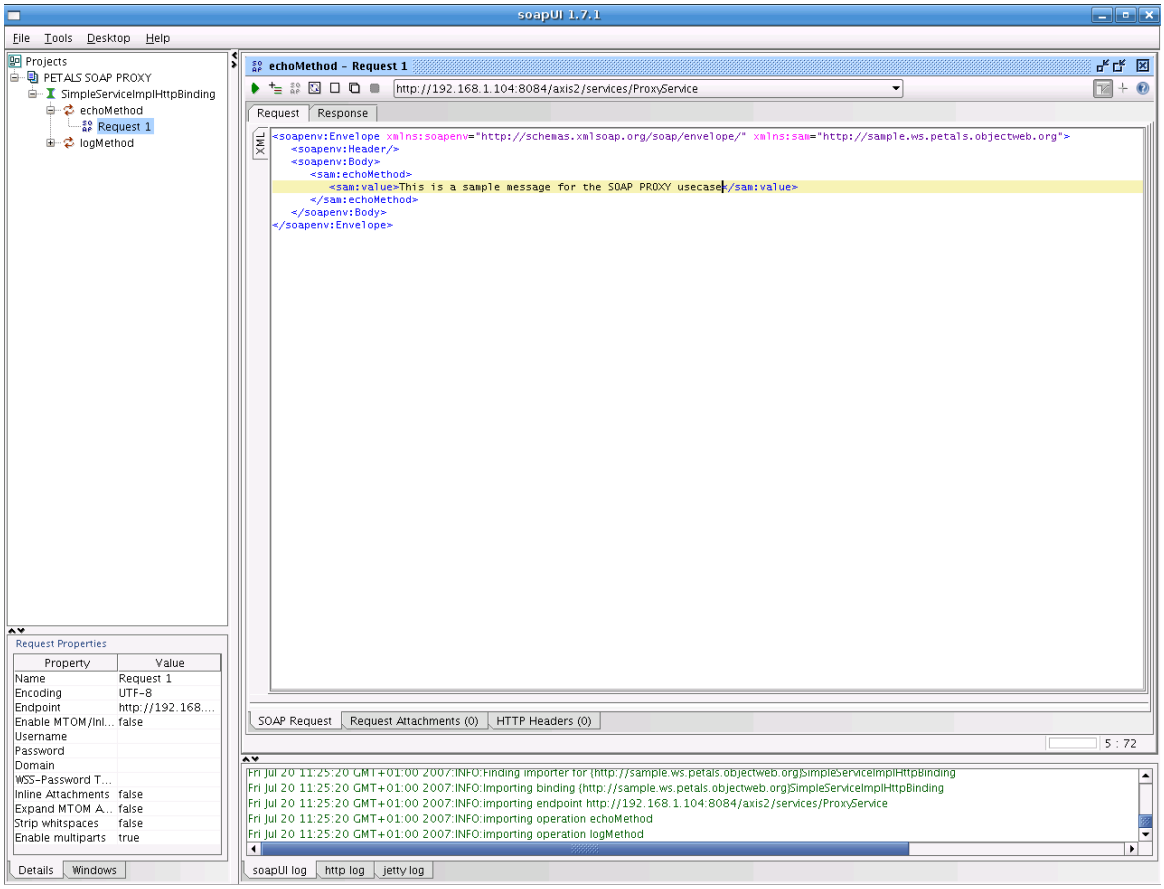
Figure 2.3. Create a new WSDL project



This will create the SOAP payload, you just have to put a message if you want. For example :

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:sam="http://sample.ws.petals.objectweb.org">
   <soapenv:Header/>
   <soapenv:Body>
      <sam:echoMethod>
         <sam:value>This is a sample message for the SOAP PROXY usecase</sam:value>
      </sam:echoMethod>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 2.4. Create the SOAP request in SOAPUI

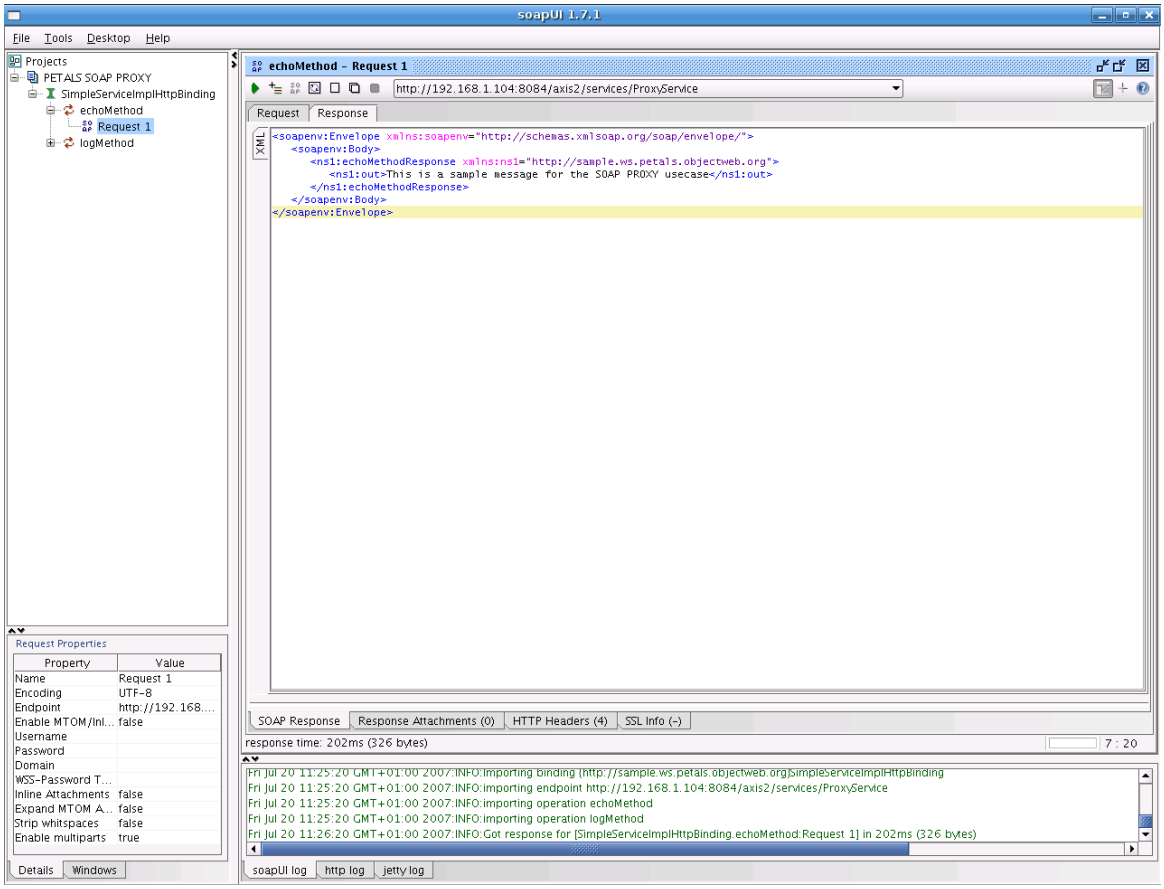


The SOAP request message is :

Send the SOAP message, the SOAP response should be like :

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns1:echoMethodResponse xmlns:ns1="http://sample.ws.petals.objectweb.org">
         <ns1:out>This is a sample message for the SOAP PROXY usecase</ns1:out>
      </ns1:echoMethodResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 2.5. SOAP message response in SOAPUI



The message transits like :

1. SOAP message is sent to PEtALS Web Service
2. The petals-bc-soap translates the SOAP message into a JBI one
3. The JBI message is sent to the Web Service provider endpoint
4. The Web Service provider endpoint translates the SOAP messahe into a JBI message
5. The SOAP message is sent to the external Web Service hosted on the J2EE container
6. The external Web Service handles the incoming message, process its service and sends a response back
7. The SOAP message transits into the JBI environment and response is sent to the client

You can also use the basic java client provided in the client directory.

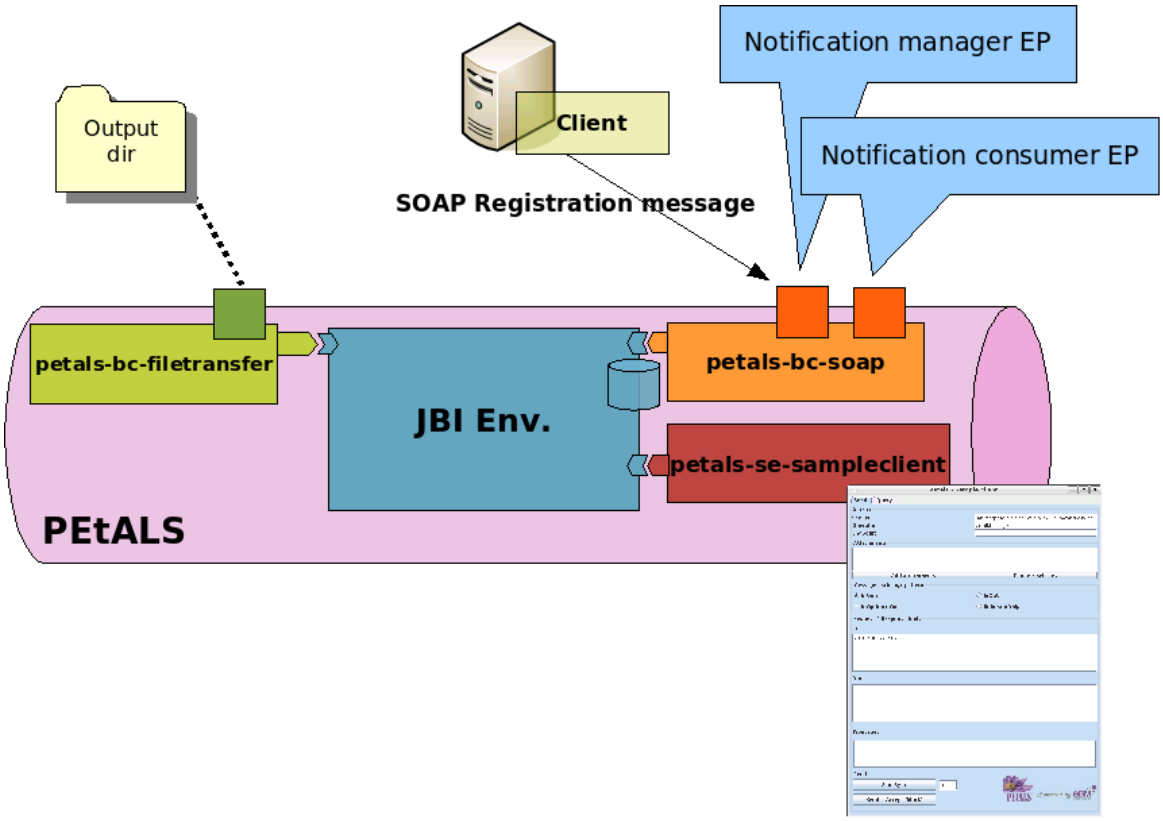
2.3. Handling Web Service notifications

This usecase explain how you can use PEtALS as Web Service notifications producer.

The current usecase provides a Web Service notification mechanisms so that external Web Service notification consumers can subscribe to notification. To get more details on this feature, please refer to the petals-bc-soap documentation.

The JBI oriented representation of this usecase is :

Figure 2.6. Producing Web Service notification with PEtALS



2.3.1. Setting up the platform

1. Start PEtALS (if not already started)
2. Install components (petals-bc-soap, petals-se-sampleclient), deploy service assembly and start these artifacts. This can be done with the Ant script:

   ant

The deployed Service Assembly contains :

1. A service unit which will be deployed on the petals-bc-soap to create a Web Service Topic (provides an endpoint).
2. A service unit which will be deployed on the petals-bc-filetransfer to provide a file transfer service (write message to disk)
3. A service unit which will be deployed on the petals-bc-soap to consume the file transfer activated endpoint : Expose a JBI service as external Web Service

A sample client is also provided and is used to create notification subscriptions.

2.3.2. Usage

On the user side, you must :

1. Launch the client which will register notification consumers into the Web Service Notification manager (hosted on the SOAP binding component). This client also launches a mini webserver which will receive Web Service Notifications:

   ant client

The client resgiters an URL which is the one activated by a Service Unit on the petals-bc-soap. All the notification messages will be sent to this URL. You should see messages like this in you console :

On the client side :

```
     [echo] Launching client...
     [java] 2007-08-20 15:24:57.488::INFO:  Logging to STDERR via org.mortbay.log.StdErrLog
     [java] 2007-08-20 15:24:57.582::INFO:  jetty-6.1.4
     [java] 2007-08-20 15:24:58.667::INFO:  Started SocketConnector@0.0.0.0:7878
     [java] Subscribing to topic null on WSN producer EPR : http://localhost:8084/wsn/producer
     [java] Notification will be sent to EPR : http://localhost:7878/notif/Consumer
     [java] Waiting to complete, type 'q' to quit application...
```

On the PEtALS console :

```
[petals.container.components.petals-bc-soap]  Receiving a new request on WSN subscription service
```

2. As JBI service client, we use the sample client to send a JBI message to the topic activated service. Select the topic activated endpoint with the sample client component, and send a message.

Figure 2.7. Publish a message in the topic with the sample client component

The message transits like this :

1. The message is published to the Web Service topic on the petals-bc-soap
2. The notification manager sends the message to all the Web Service notification consumers
3. The notification consumer receives a notification message

You should see the following message on the client side :

```
     [java] >
     [java] ### RECEIVING REQUEST :
     [java] --MIMEBoundaryurn_uuid_8DB38D8DB2DEF5D9DF1187620267293
     [java] Content-Type: application/xop+xml; charset=UTF-8; type="text/xml"
     [java] Content-Transfer-Encoding: binary
     [java] Content-ID: <0.urn:uuid:8DB38D8DB2DEF5D9DF1187620267294@apache.org>

     [java] <?xml version='1.0' encoding='UTF-8'?>
             <soapenv:Envelope xmlns:wsa="http://www.w3.org/2005/08/addressing"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
               <soapenv:Header>
                 <wsa:To>http://localhost:7878/notif/Consumer</wsa:To>

 <wsa:From><wsa:Address>http://192.168.1.104:8084/wsn/producer</wsa:Address></wsa:From>
                 <wsa:MessageID>urn:uuid:8DB38D8DB2DEF5D9DF1187620267291</wsa:MessageID>

 <wsa:Action>http://docs.oasis-open.org/wsn/bw-2/NotificationConsumer/NotifyRequest</wsa:Action>
               </soapenv:Header>
               <soapenv:Body>
                 <wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
                   <wsnt:NotificationMessage>
                     <wsnt:SubscriptionReference>
                       <wsa:Address>http://localhost:7878/notif/Consumer</wsa:Address>
                     </wsnt:SubscriptionReference>
                     <wsnt:Topic
 Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete">TopicSample</wsnt:Topic>

 <wsnt:ProducerReference><wsa:Address>http://192.168.1.104:8084/wsn/producer</wsa:Address></
wsnt:ProducerReference>
                     <wsnt:Message>
                       <text>This is a notification message from the sample client service</text>
                     </wsnt:Message>
                   </wsnt:NotificationMessage>
                 </wsnt:Notify>
               </soapenv:Body>
             </soapenv:Envelope>
     [java] --MIMEBoundaryurn_uuid_8DB38D8DB2DEF5D9DF1187620267293--
     [java] ################################################
```

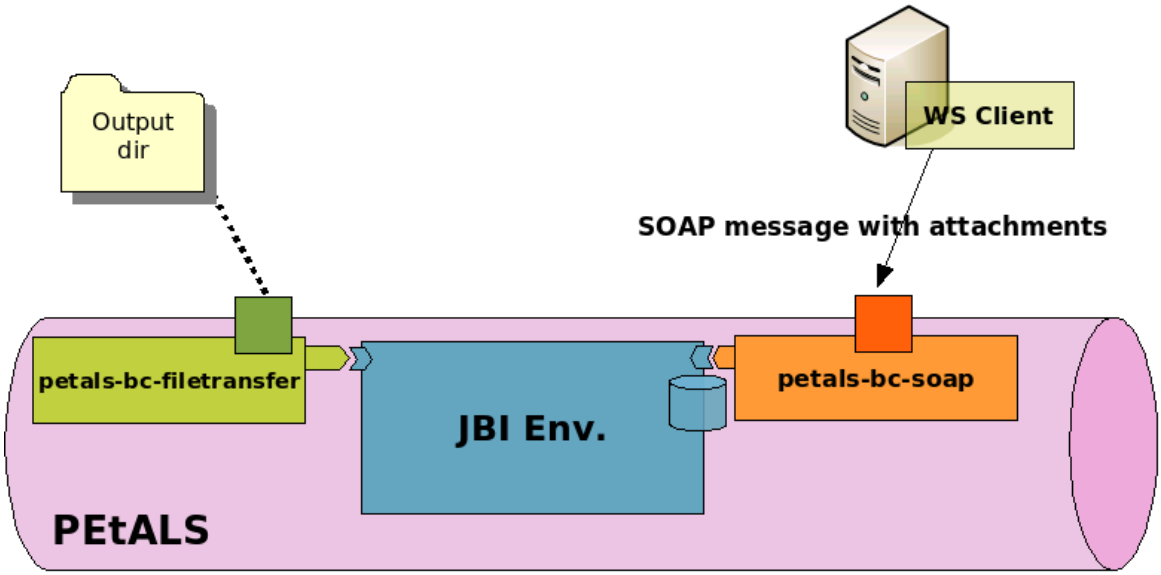
The element under the `<wsnt:Message>` element is the content of the original JBI message sent from the sample client.

2.4. Handling SOAP attachments

This usecase explains how the petals-bc-soap can handle SOAP attachments for incoming and outgoing SOAP messages. To get more details, please read the petals-bc-soap user's guide.

The JBI representation of the current usecase is :

Figure 2.8. Handling SOAP attachments with PEtALS



2.4.1. Setting up the platform

1. Start PEtALS (if not already started)
2. Install components (petals-bc-soap, petals-bc-filetransfer), deploy service assembly and start these artifacts. This can be done with the Ant script:

   ant

The deployed service assembly contains:

1. A service unit which will be deployed on the petals-bc-filetransfer which provides a file transfer service. This service unit activates a *FileTransferMTOMEndpoint* endpoint.
2. A service unit which will be deployed on the petals-bc-soap which consumes *FileTransferMTOMEndpoint* endpoint activated by the previous service unit. This service unit also creates a new Axis2 service named *MTOMService*.

A Web Service sample client is also provided to send SOAP message with attachments to the exposed Web Service on `http://<HOST>:<PORT>/axis2/services/MTOMService`. This client uses the Axis2 client library to send the SOAP message with an attachment.

2.4.2. Usage

On the user side, after installation and deployment (see previous chapter), you just have to :

1. Launch the Web Service client which will send a SOAP message with attachments to the Web Service:

   ant client

This invokation will send the mtom.txt file from the data directory as SOAP attachment (If you want to send another file, you can modify the Ant script).

The message transits in the different layers and components like this :

1. The SOAP message is handled by the petals-bc-soap activated Web Service
2. The SOAP message and its attachments are translated into a JBI message
3. The JBI message is sent to the file transfer activated endpoint

4. The JBI message and its attachments are written to files

To check that the usecase is valid you should have files in the directory you have chosen in the filetransfer service unit.

2.5. Securing access to an exposed Web Service

This usecase explains how you can secure Web Service exposed by PEtALS through the SOAP binding component. This feature is available since the version 2.0 of the SOAP binding component. To get more information on this feature, read the component documentation.

The current usecase provide user/password security on an exposed Web Service.

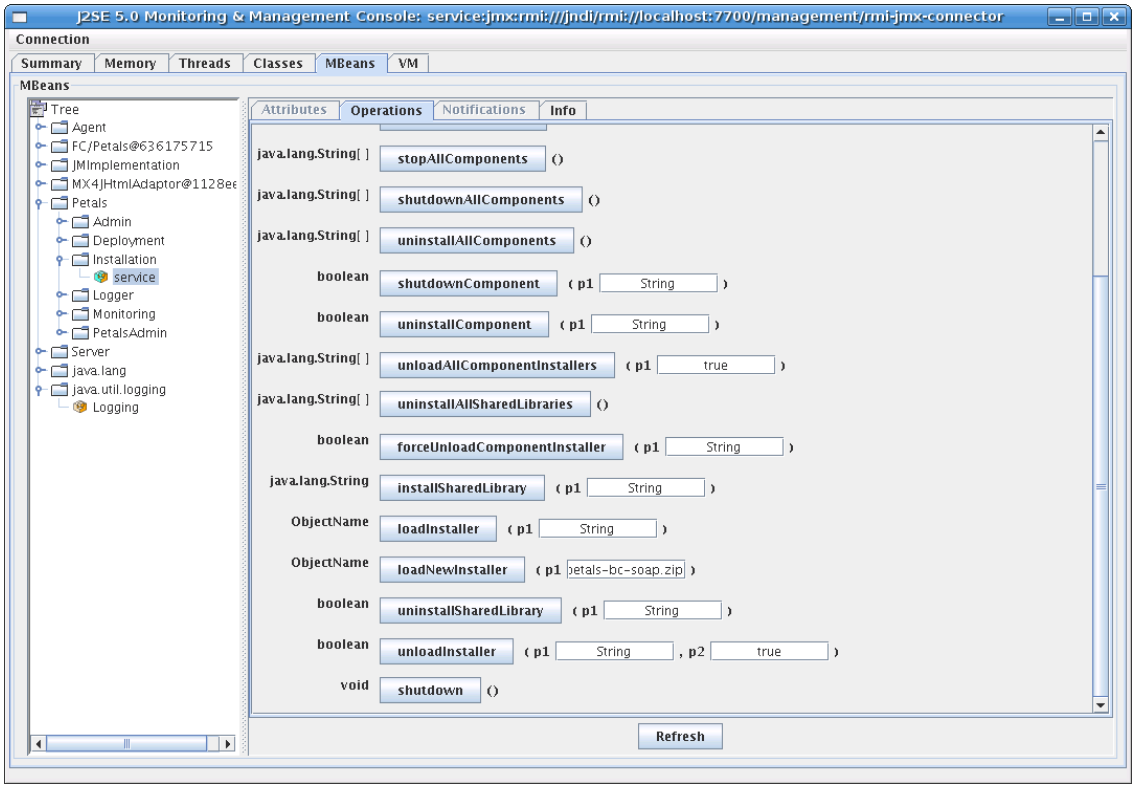
2.5.1. Setting up the platform

To set this configuration, you must:

1. Install the petals-bc-soap component (via JMX):

   Go into the Installer MBean view and enter the petals-bc-soap component file URL (must start with file://). The value is something like `file:///<USECASE>/deployables/petals-bc-soap.zip` where <USECASE> is the usecase directory.

   Figure 2.9. Load the SOAP binding component through JMX

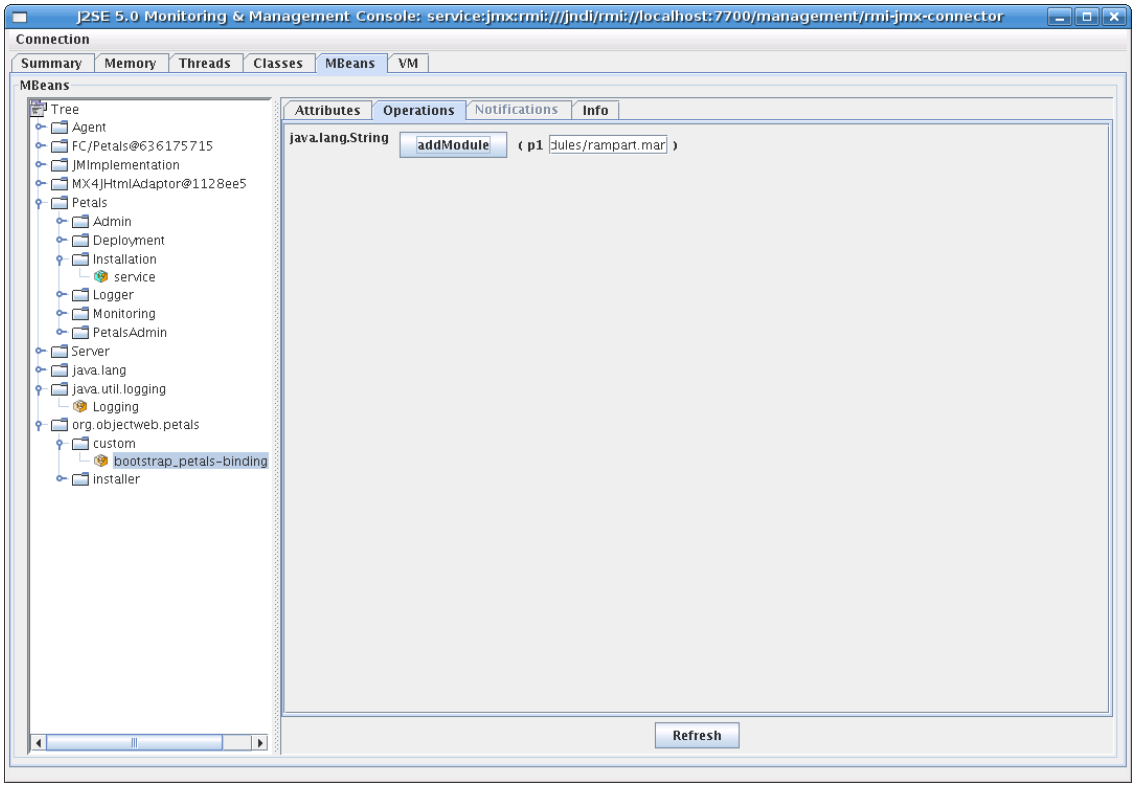


   You should see a popup window like this which notify that the component has been successfully installed.

2. Add the security module (rampart) during the bootstrap phase:

   The previous component installation has created a custom MBean (bootstrap_petals-binding-soap) which is used to invoke an addModule method. In the addMmodule field, enter the URL of the Rampart module. The value is something like file:///<USECASE>/axis2/modules/rampart.mar where <USECASE> is the usecase directory.

Figure 2.10. Add the Rampart module to the SOAP BC

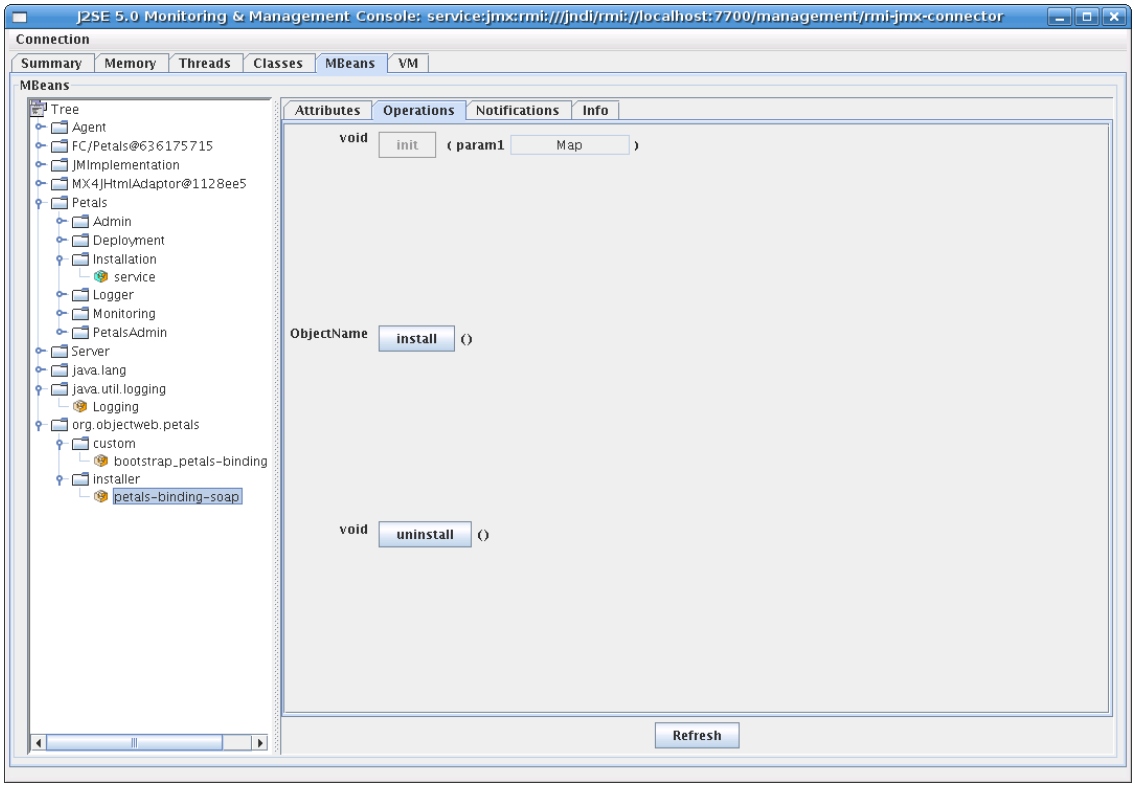


And click on the addModule button. You should see a popup window which notify you that the module has been successfully added to Axis2.

3. Install the petals-bc-soap

   Go to the Installer MBean and click on the install button:

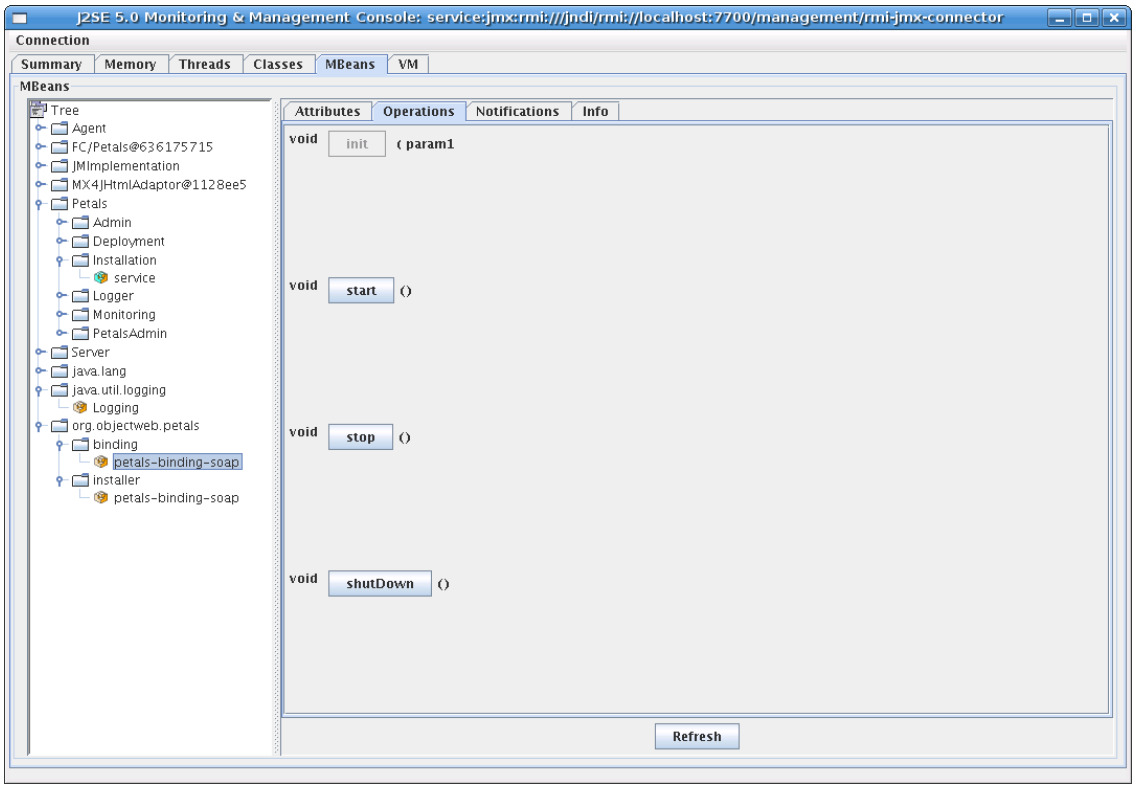
Figure 2.11. Install the SOAP BC



4. Start the petals-bc-soap component

   You can start the component in the ComponentLifeCycle MBean by clicking on the start button:

Figure 2.12. Start the SOAP BC



You should see a popup window like this which notify that the component has been started.

In addition, if you have started PEtALS in console mode, you should see a log message like this:

```
[petals.container.components.petals-binding-soap]       start
2007-08-08 15:35:15.357::INFO:  Logging to STDERR via org.mortbay.log.StdErrLog
[petals.container.components.petals-binding-soap]       Starting Jetty server...
[petals.container.components.petals-binding-soap]       Port : 8084 / Max poolsize : 255 / Min
 poolsize : 1/ Acceptors size : 4
2007-08-08 15:35:15.491::INFO:  jetty-6.1.4
2007-08-08 15:35:15.564::INFO:  Started SelectChannelConnector@0.0.0.0:8084
```

The SOAP binding component is now started...

5. Install and start the petals-se-helloworld component, deploy and start the service assembly:

   ant

The service assembly contains:

1. A service unit which will be deployed on the petals-se-helloworld which provides the helloworld service. This service unit activates a JBI endpoint.
2. A service unit which will be deployed on the petals-bc-soap which consumes the helloworld service. The deployment of this service unit will create a new Axis service in the component and will add a Web Service security context.

A Web Service client is also provided to send SOAP messages with and without security contexts to the exposed Web Service. This client uses the Axis2 client library to build and send the SOAP messages.

2.5.2. Usage

A client is provided within the usecase distribution. Launching this client will invoke the exposed Web Service in different ways:

1. Whithout the security header: The helloworld service will not be called since the security context is not valid. An error message will be displayed.

2. With the security header: The security context is valid. The helloworld service is called and the response is printed.

To launch the client, you have to invoke the Ant script like this:

ant client

A normal execution will print output messages on the terminal (error stack trace is a normal result):

```
client
Buildfile: build.xml

client:
     [java] Unsecure WS call...
     [java] log4j:WARN No appenders could be found for logger (org.apache.axis2.util.Loader).
     [java] log4j:WARN Please initialize the log4j system properly.
     [java] org.apache.axis2.AxisFault: WSDoAllReceiver: Incoming message does not contain required
 Security header
     [java]     at org.apache.axis2.util.Utils.getInboundFaultFromMessageContext(Utils.java:434)
     [java]     at
 org.apache.axis2.description.OutInAxisOperationClient.send(OutInAxisOperation.java:373)
     [java]     at
 org.apache.axis2.description.OutInAxisOperationClient.execute(OutInAxisOperation.java:294)
     [java]     at org.apache.axis2.client.ServiceClient.sendReceive(ServiceClient.java:520)
     [java]     at org.apache.axis2.client.ServiceClient.sendReceive(ServiceClient.java:500)
     [java]     at
 org.objectweb.petals.usecase.soapsecurity.HelloworldClient.callSayHello(HelloworldClient.java:131)
     [java]     at
 org.objectweb.petals.usecase.soapsecurity.HelloworldClient.main(HelloworldClient.java:72)
     [java]     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     [java]     at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
     [java]     at
 sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
     [java]     at java.lang.reflect.Method.invoke(Method.java:585)
     [java]     at org.apache.tools.ant.taskdefs.ExecuteJava.run(ExecuteJava.java:193)
     [java]     at org.apache.tools.ant.taskdefs.ExecuteJava.execute(ExecuteJava.java:130)
     [java]     at org.apache.tools.ant.taskdefs.Java.run(Java.java:705)
     [java]     at org.apache.tools.ant.taskdefs.Java.executeJava(Java.java:177)
     [java]     at org.apache.tools.ant.taskdefs.Java.execute(Java.java:83)
     [java]     at org.apache.tools.ant.UnknownElement.execute(UnknownElement.java:275)
     [java]     at org.apache.tools.ant.Task.perform(Task.java:364)
     [java]     at org.apache.tools.ant.Target.execute(Target.java:341)
     [java]     at org.apache.tools.ant.Target.performTasks(Target.java:369)
     [java]     at org.apache.tools.ant.Project.executeTarget(Project.java:1214)
     [java]     at org.apache.tools.ant.Project.executeTargets(Project.java:1062)
     [java]     at org.apache.tools.ant.Main.runBuild(Main.java:673)
     [java]     at org.apache.tools.ant.Main.startAnt(Main.java:188)
     [java]     at org.apache.tools.ant.launch.Launcher.run(Launcher.java:196)
     [java]     at org.apache.tools.ant.launch.Launcher.main(Launcher.java:55)
     [java] Secure WS call...
     [java] PWCBHandler pwcb.getIdentifer()=bob
     [java] <sayHelloResponse xmlns="http://org.objectweb.petals/"><sayHelloReturn>you told me :
 Helloworld secured service</sayHelloReturn></sayHelloResponse>

BUILD SUCCESSFUL
```