PEtALS JBI Component User Guide

"WS-Notification Broker" Service Engine

Author: Thierry DÉJEAN - **Petals ESB Team**
Contact: thierry.dejean@petalslink.com
Revision: 1
Last change: 20.07.2010

OW2 Consortium

Contents

PART 1

Introduction

The objective of this document is to explain how to install, configure and use the JBI component named “petals-se-WsNotificationBroker“. It also provides a short view of its architecture and its goals as well as some simple use-cases that illustrate main WS-Notification features.

The remainder of this document is organized as follows. The introduction describes the context and gives some background on WS-Notification features respect to PEtALS Entreprise Service Bus. Section 2 provides an overview of WS-Notification Broker component. Next two sections - section 3 and section 4 - describe how to install and configure the component. And the last section 5 provides some simple use-cases that show WS-Notification component's features in action

This document will not deal with ”WS-Notification” paradigm in details and the reader is supposed to be familiar with its concepts. If not, let's have a look to OASIS Web Services Notification TC website at : `http://www.oasis-open.org/committees/wsn/`.

1.1 The WS-Notification OASIS Standards

The WS-Notification OASIS Standards consist of is a set of three specifications that describe mechanisms and concepts associated to Eventing or Notification paradigm. These specification are :

- the **Topic** specification[8] : this specification defines the notion of topic which can be seen as a way to label and classify events that producers can send. Shortly, any actor - event consumer - who wants to received a specific event will previously subscribe on the topic which the event is associated to.
- the **Base Notification** specification[3] : this specification defines the main mechanism of “publish-subscribe“ and describes what is exactly a ”Notification Consumer“ as well as a ”Notification Producer“. it also provides some ”sub-mechanisms“ that can help to implement a more complex and optimized EDA systems (such like ”pull point“ or pausable subscription mechanisms).
- the **Brokered Notification** specification[2] : this specification is an extension of the previous one and mainly describes how to decouple Notification consumers to Notication Producer introducing a intermediary : a Notification Broker

No more details will be provided here concerning these specifications content and the reader who is interested will find more details directly in mentioned papers.

1.2 PEtALS ESB and EDA paradigm

Even Driven Architecture paradigm has become more and more important in nowadays Service Architectures and probably represents the next step after SOA one. Aware of this fact, it was clear that PEtALS Entreprise Service Bus should provide features that answer to this need. In other words the PEtALSESB container must provide a “publish-subscribe“ mechanism which is the base of EDA architecture. That is why the technical choice have been to implement ”WS-Notification“ OASIS Standards through a JBI Service Engine that make PEtALS container ”WS-Notification paradigm“ compliant. For practical, the petals-se-WsNotificationBroker JBI service engine component implements a Notification Broker as describing in Ws-BrokeredNotification OASIS specification[8].

1.3 Limitations And Extensions

1.3.1 Limitations

Actually, only main Ws-BrokeredNotification operations have been implemented. In details these operations. Available operations are listed in component wsdl file A.
Moreover unless suggested in OASIS specifications[8, 3], Notification actors - notification producer, notification consumer and notification broker, have not been implementated as WS-Resources[1] which means that related operations are actually not supported. WS-Resource* OASIS specifications reference are provided at the end of this document[4, 5, 6, 7]

1.3.2 Extensions

This wsdl file defines an extra portType named **SupportedTopicsSet**. This portType provides some operation definitions that are not part of the WS-Notification OASIS specifications. Nevertheless, it seems that such operations are essential to make Notification Broker usage easier during runtime.

In details, these operations are :

- The ***GetSupportedTopics(...)*** operation that let any actors to known the set of topics currently supported by the Notification broker.
- The ***AddNewSupportedTopic(...)*** operation that make any actors to update the set of topics currently supported by the Notification broker during runtime. The new topic to add must be describe as topic expression using ”Concrete“ or ”simple“ dialect[2].

[1] see related website at `www.oasis-open.org/committees/wsrf/`
[2] see WSN OASIS specifications for more details [3, 8, 2]

PART 2

Architecture Overview

2.1 UML Class Diagram

A “synthetic“ - means non complete - UML class diagram of the WS-Notification Broker service engine (the «**org.ow2.petals**» package) and its dependences with WS-Notification libraries (the package named «**com.ebmwebsourcing.wsstar.notification.service**»):

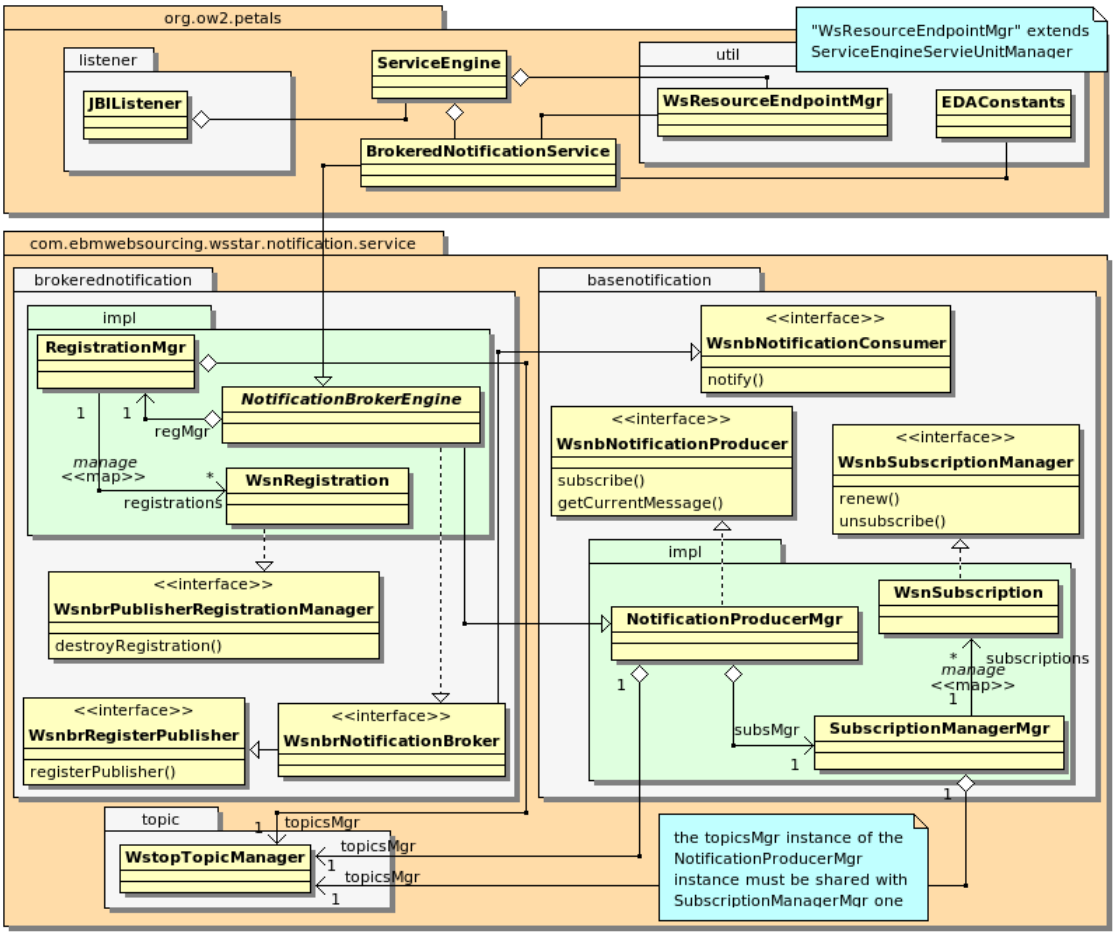


Figure 2.1: UML class diagram

2.2 The PEtALS WS-Notification libraries

The PEtALS WS-Notification libraries are a set of java libraries that implement WS-Notification specifications content as well as WS-Addressing W3C specification[1] WS-Notification specifications depend on. In details :

ws-addressing-model : this library provides Java classes that implement ws-addressing data types defined in related specification. Java classes have been generated using JAXB tools are xml schema associated to ws-addressing specification.

ws-resource-model : this library provides Java classes that implement ws-resource data types defined in related specifications. Java classes have been generated using JAXB tools are xml schema associated to ws-resource specifications. It depends on « ws-addressing-model» library.

ws-notification-model : this library provides Java classes that implement ws-notification data types defined in related specifications. Java classes have been generated using JAXB tools are xml schema associated to ws-notification specifications. it depends on «ws-addressing-model» and «ws-resource-model» libraries.

ws-notification-service : this library provides basic component that implement WS-Notification mechanisms such as creation and management of consumers subscriptions resource, topics management (associate and store links between subscriptions and known topics, registrations creation and management, ... and so on. In fact all mechanisms described in both «WS-BaseNotification» and «WS-BrokeredNotification» are implemented in this library. And the **BrokerNotificationService.java** java class, in WsNotificationBroker service engine source code simply realize - extends in OO paradigm - the java abstract class **BrokeredNotificationService.java**, as shown in previous UML diagram class (fig.2.1).

2.3 The PEtALS WS-Notification Broker Component

2.3.1 CDK architecture part

As explain previously, the PEtALS WS-Notification broker component business logic part - WS-Notification mechanisms - is implemented through the «**BrokeredNotificationService.java**» class. Other part of the component architecture - based on the Component Development Kit - will not by discuss in this document. For more details about these concerns, please consult petalslink's wiki at `http://doc.petalslink.com/` or contact the petals Team[1]

2.3.2 Classe Descriptions

JBIListener.java : implements «AbstractJBIListener» CDK's class. Provides mechanisms that make the component able to perform received JBI Messages. Determine which operation must be performed, extract WS-Notification business logic payload and delegate treatments to WS-Notification business logic stack implemented by the «BrokeredNotificationService» class. See PEtALS CDK's documentation for more details

ServiceEngine.java : implements «AbstractServiceEngine» CDK's class. Initialize WS-Notification business Logic mechanisms. Provides common mechanisms of JBI Service Engine Components. See PEtALS CDK's documentation for more details

BrokeredNotificationService.java : implements WS-Notification business logic mechanisms that perform WS-Notification standard operations «Subscribe», «Unsubscribe», «RegisterPublisher», «DestroyRegistration», «Notify», ... Based on WS-Notification libraries.

WsResourceEndpointMgr.java : provides mechanisms used to create and register new «WS-Resource» endpoints - those associated to subscription and registration resources - on the PEtALS bus. If no ServiceUnitManager is already linked to the component, this class must implements the CDK's class «ServiceEngineServiceUnitManager». Otherwise, if a ServiceUnitManager is already linked to the component, one of its attribute must be an instance of «WsResourceEndpointMgr» (see petals-se-WsnProducer-new source code).

EDAConstant.java : provides some common Constant widely used in WS-Notification context

[1] mail : petals[at]lists.ebmwebsourcing.com

2.3.3 WS-Notification main mechanisms

Following sub-sections provide ”high level“ overviews - based on UML sequence diagrams - of five main WS-Notification operations performing, respect to the WS-Notification Broker actor. Please be aware that these may not follow rigorously the UML syntax and their aim is just to provide a global understand of how the WS-Notification Broker actor.

- **notify operation performing :**

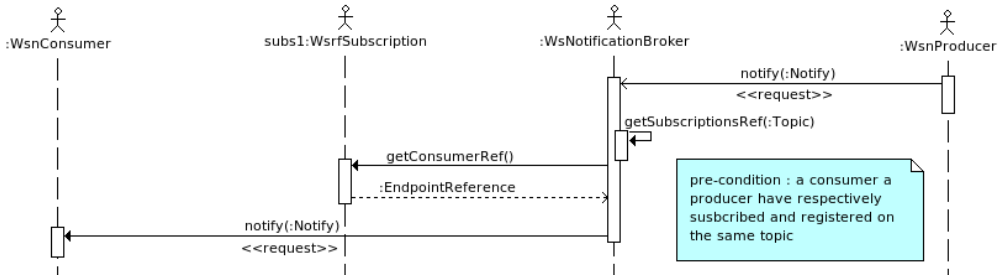


Figure 2.2: Notify UML sequence diagram

- **subscribe operation performing :**

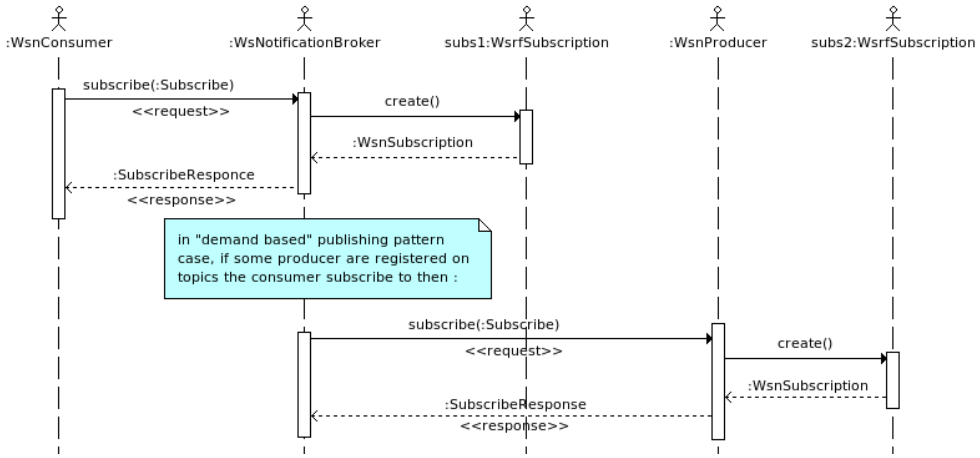


Figure 2.3: Subscribe UML sequence diagram

- **registerPublisher operation performing :**

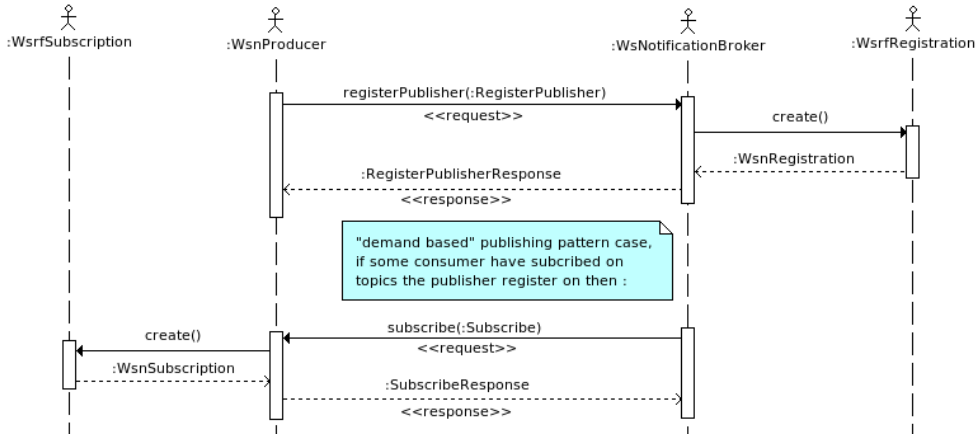


Figure 2.4: RegisterPublisher UML sequence diagram

- **unsubscribe operation performing :**

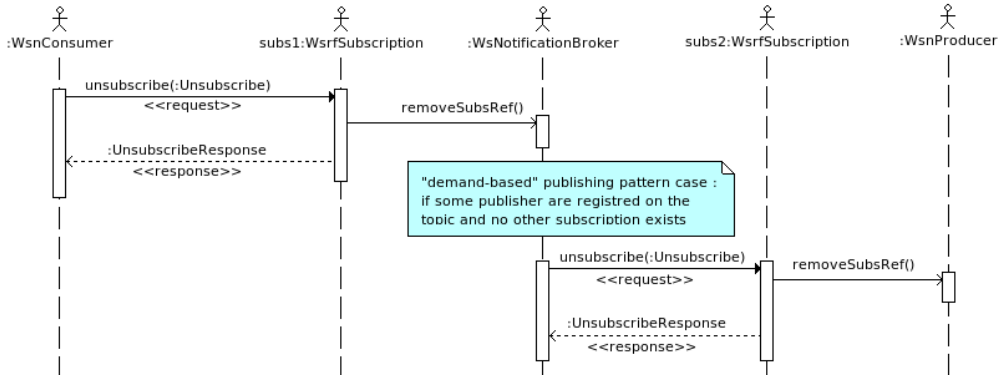


Figure 2.5: Unsubscribe UML sequence diagram

- **destroyRegistration operation performing :**

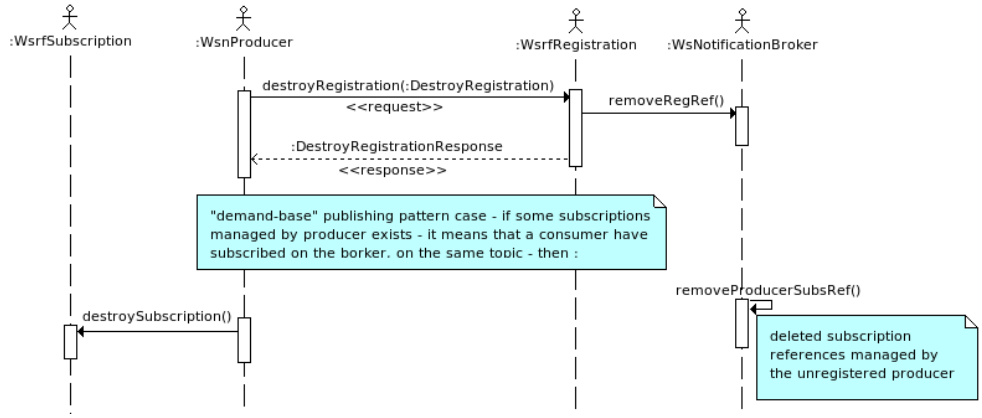


Figure 2.6: DestroyRegistration UML sequence diagram

PART 3

Configuration

3.1 Generic configuration parameters of JBI component

Parameter	Description	Default	Required	Scope
acceptor-pool-size	The size of the thread pool used to accept Message Exchange from the NMR. Once a message is accepted, its processing is delegated to the processor pool thread.	3	Yes	Runtime
processor-pool-size	The default size of the thread pool used to process Message Exchanges. Once a message is accepted, its processing is delegated to one of the thread of this pool.	10	Yes	Runtime
processor-max-pool-size	The maximum size of the thread pool used to process Message Exchanges. The difference between this size and the **processor-pool-size** represents the dynamic threads that can be created and destroyed during overhead processing time.	50	No	Runtime
ignored-status	When the component receives an acknowledgement message exchange, it can skip the processing of these message according to the type of the acknowledgment. If you decide to not ignore some acknowledgement, the component listeners must take care of them. Accepted values : • DONE_AND_ERROR_IGNORED • DONE_IGNORED • ERROR_IGNORED • NOTHING_IGNORED	DONE_AND_ERROR_IGNORED	Yes	Component
properties-file	Name of the file containing properties used as reference by other parameters. Parameters reference the property name in the following pattern $myPropertyName. At runtime, the expression is replaced by the value of the property. The value of this parameter is : • an URL • a file relative to the PEtALS installation path • an empty value to stipulate a non-using file	-	No	Installation
external-listener-class-name	Qualified name of the class extending ***AbstractExternalListener***. ☞ Only for BC components	-	No	Component
jbi-listener-class-name	Qualified name of the class extending ***AbstractJBIListener***.	-	Yes	Component

Definition of CDK parameter scope :

- *Component* : The parameter has been defined during the development of the component. A user of the component can not change its value.
- *Installation* : The parameter can be set during the installation of the component, by using the installation MBean (see JBI specifications for details about the installation sequence). If the parameter is optional and has not been defined during the development of the component, it is not available at installation time.
- *Runtime* : The paramater can be set during the installation of the component and during runtime. The runtime configuration can be changed using the CDK custom MBean named “RuntimeConfiguration“. If the parameter is optional and has not been defined during the development of the component, it is not available at installation and runtime times.

3.2 Specific WS-Notification Broker configuration

3.2.1 Specific fields of jbi.xml file

The WS-Notification Broker component expect extra parameters in its jbi.xml configuration file. These parameters are listed, with their description, in the following table.

Parameter	Description	Default	Required	Scope
publishing-pattern	this parameter define the broker comportment respect to publishing patterns, as describe in WS-BrokeredNotification specification[2] (page 12/43). Possible values are pattern's names : • simple • brokered • demand-based	demand-based	Yes	Component

3.2.2 Additional configuration elements

The other specific configuration element of the WS-Notification Broker component is its "supportedTopicSet.xml" configuration file which define its default set of supported - known - topics which notification producers and notification consumers will be allowed to respectively register to and subscribe to.
A sample of such file is provided in the annexe part B. This document is an xml representation of a TopicSet type object as defined in WS-Topics specification[8]. The to configure it is base on using a Environment variable named **SUPPORTED_TOPICS_SET** whose value is the "supportedTopicSet.xml" file path.


If not set, then a default embedded "supportedTopicSet.xml" file wile be used.

PART 4

Installation

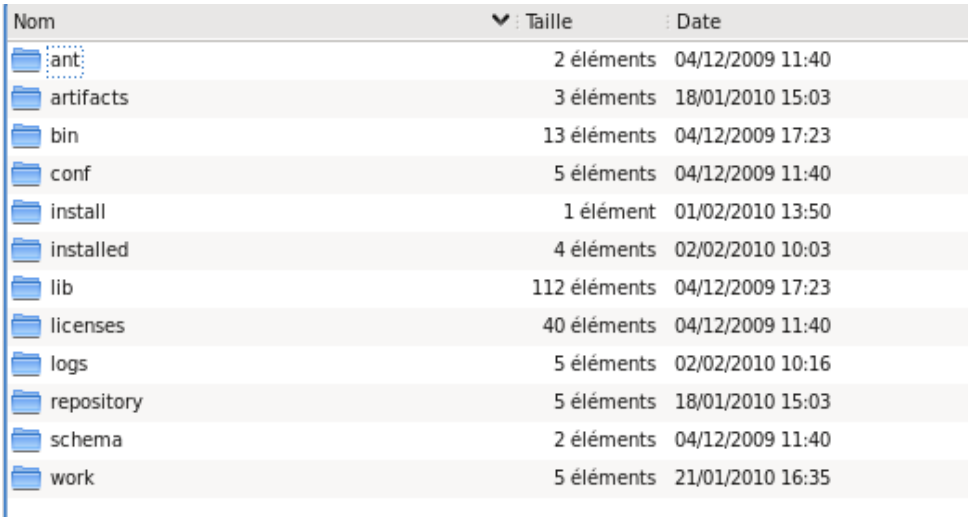
4.1 PEtALS Entreprise Service Bus installation

Please refer to Official PEtALS Entreprise Service Bus Documentation on petalslink's wiki at `http://doc.petalslink.com/`. Some Pdf format documantation can be found at `http://petals.ow2.org/documentation.html`

4.2 WS-Notification Broker service engine installation

Unless standart JBI component installation process, the component petals-se-WsNotificationBroker does not require any service units to be configured. As soons as it is installed and deployed in PEtALS container, relared endpoint are stored to the Services Registry.

The only rule is that it must be deployed and started before any other component that intends to use petals ESB WS-Notification features. As any other PEtALS components, the petals-se-WsNotificationBroker broker service engine can be installed and deployed by hand. To do that you simply have to copy and paste the "petals-se-notification-X.X.zip" zip archive of the component in the sub-folder of your petals ESB platform called «**install/**». Then, few seconds later, the embedded petals ESB autoloader mechanism will deploy and start the component. This mechanism is enabled in the default configuration of the bus.



Nom	Taille	Date
ant	2 éléments	04/12/2009 11:40
artifacts	3 éléments	18/01/2010 15:03
bin	13 éléments	04/12/2009 17:23
conf	5 éléments	04/12/2009 11:40
install	1 élément	01/02/2010 13:50
installed	4 éléments	02/02/2010 10:03
lib	112 éléments	04/12/2009 17:23
licenses	40 éléments	04/12/2009 11:40
logs	5 éléments	02/02/2010 10:16
repository	5 éléments	18/01/2010 15:03
schema	2 éléments	04/12/2009 11:40
work	5 éléments	21/01/2010 16:35

Figure 4.1: Petals ESB folder details

☞ To remove "by hand" an installed PEtALS ESB component and/or its service assembly, you simply have to remove the related archive from the "Installed/" folder

Otherwise you can use PEtALS Entreprise Service Bus webconsole tools. See associated documentation for more details[1].

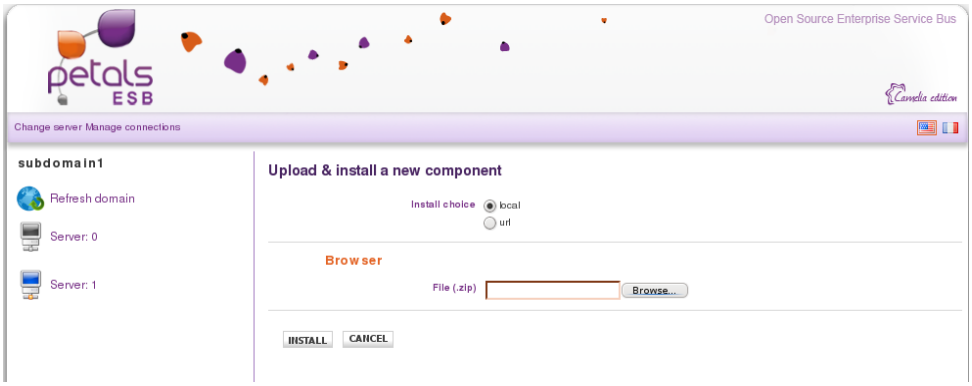


Figure 4.2: Webconsole Install Component administration page of Petals ESB

4.3 Others JBI components

In order to run use-cases described in next section 5 that illustrate, through the WsNotificationBroker service engine, PEtALS WS-Notification features, some extra components need to installed. These components are :

petals-se-WsnConsumer-new service engine :PEtALS service engine component that acts as a Notification Consumer. As soon as its related service assembly is deployed and started, it sends a "Subscribe" request to the Broker (the petas-se-WsNotificationBroker that must have been previously deployed). A "Unsusbcribe" request is sent when its service assembly is stopped.

petals-se-WsnProducer-new service engine :PEtALS service engine component that acts as a Notification Producer. As soon as its related service assembly is deployed and started, it sends a "RegisterPublisher" request to the Broker (the petas-se-WsNotificationBroker that must have been previously deployed). A "DestroyRegistration" request is sent when its service assembly is stopped. When the component is registred, it can receive a "Subscribe" request from the "Broker" as described in the "demand-based" publishing pattern (see page 13/43 of the "WS-BrokeredNotification" specification). Then, once its has been registred, it periodically checks if some Broker's subscriptions exists on its own supported topics and if so, it will send a notification message to it.

[1]Once again, related documentation can be found Petalslink's wiki at `http://doc.petalslink.com/` or in the section «**Documentation** of PEtALS Entreprise Service Buswebsite at `http://petals.ow2.org/documentation.html`

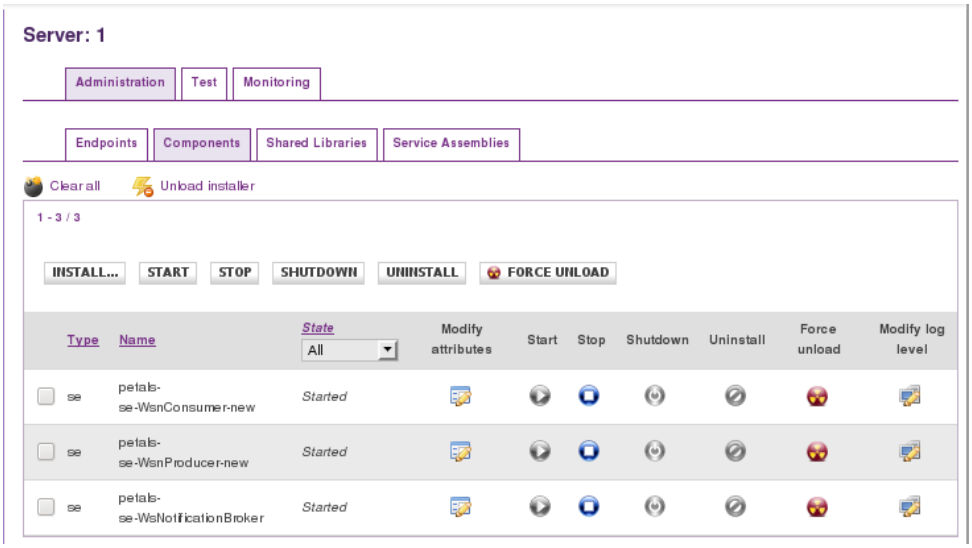


Figure 4.3: Webconsole component administration page of Petals ESB

And theirs associated configurations[2] :

sa-se-WsnConsumer service engine : service assembly that provides WsnConsumer service engine standard configuration (endpoint, ...) as well as more specific configuration part such as the xml representation[3] - embedded file named **subscribe-payload.xml** - of the «Subscribe» request payload to use.

sa-se-WsnProducer service engine :service assembly that provides WsnProducer service engine standard configuration (endpoint, ...) as well as more specific configuration part such as the xml representation[4] - embedded file named **registerOnStart.xml** - of the «RegisterPublisher» request payload to use.

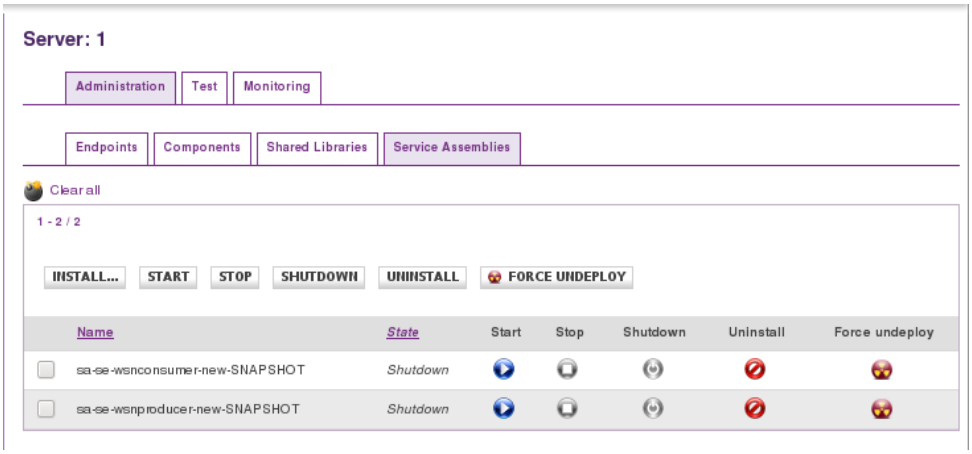


Figure 4.4: Webconsole service assemblies administration page of Petals ESB

Extra-components and their services assemblies can by installed by the same way as the WsNotificationBroker service engine. Only take care that the WsNotificationBroker service engine has been installed and started first (see previous section.4.2)

[2] Service Assembly taht contains Service Units in JBI language
[3] the path of the xml file is set as paramter of the associated «jbi.xml» configuration file of the embedded service unit
[4] the path of the xml file is set as paramter of the associated «jbi.xml» configuration file of the embedded service unit

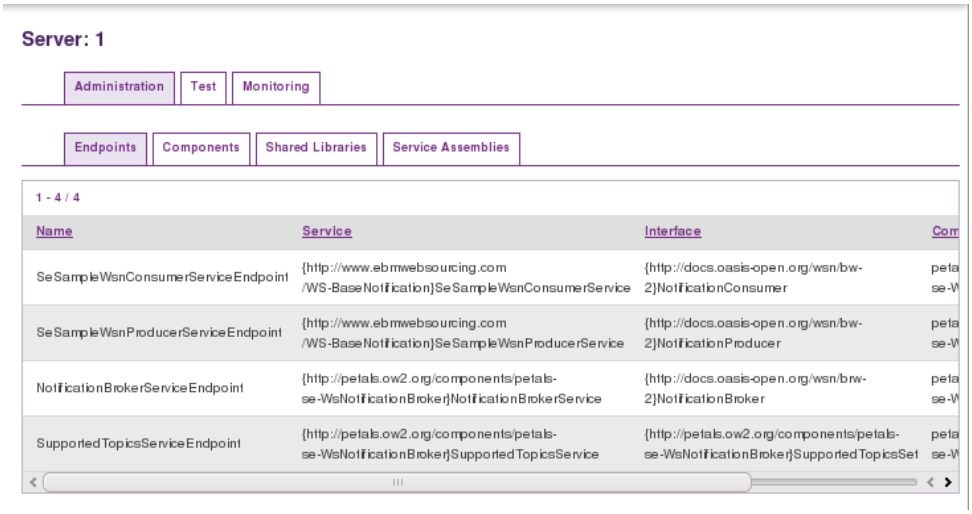


Figure 4.5: Webconsole endpoints administration page of Petals ESB

PART 5

Ws-Notification Broker in action

In this section some simple use-cases will be run using PEtALS webconsole interface and especially its «administration» part that make user able to start or stop components and service assemblies. Of course this is the only way to run use-cases but it is probably the most friendly one. Other way is to use «by hand» method which means copy and past service assemblies archives in «**install/**» folder of PEtALS to install and start them or remove existing one from «**installed/**» folder of PEtALS to stop and uninstall them.

5.1 «subcribe/unsubscribe» use-case

5.1.1 scenario

This use-case illustrate the upper parts of UML sequence diagrams presented in section 2.3.3 (fig.2.3 and fig.2.5). A Notification consumer sends a «subscribe» request to the broker, then it sends a «unsubscribe» request to the subscription resource newly created, according to subscription reference returned by the broker.

5.1.2 actors concerned

The WS-Notification actors concerned by this use-case are :

- **Notification Consumer** which sends a «subscribe» request the broker and sends later a «unsubscribe» request to the previously created subscription resource whose reference have been returned by the broker
- **Notification Broker** which perform the request and return the reference of the newly created subscription resource
- **SubscriptionManager** resource - the subscription created by the Broker - which will perform the «unsubscribe» request

5.1.3 how to run the use case

Using the PEtALS webconsole administration interface, services assemblies page, start the «sa-se-wsnconsumer» service assembly to make the consumer send a «subscribe»request to the broker, and stop it to make it send a «unsubscribe» request to subscription resource newly created

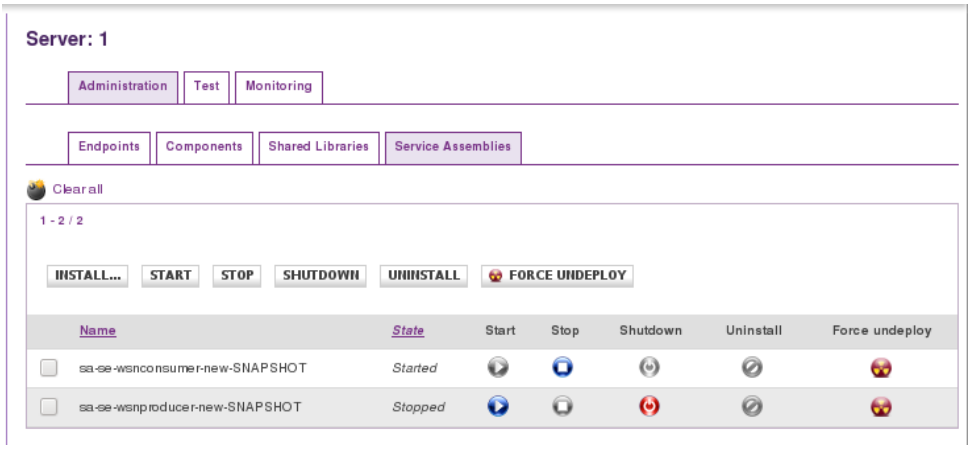


Figure 5.1: sa-se-wsnconsumer service assembly has been started

5.1.4 expected results

In the PEtALS webconsole administration interface, endpoints page, a new endpoint must have been added after the «sa-se-wsnconsumer» started[1]

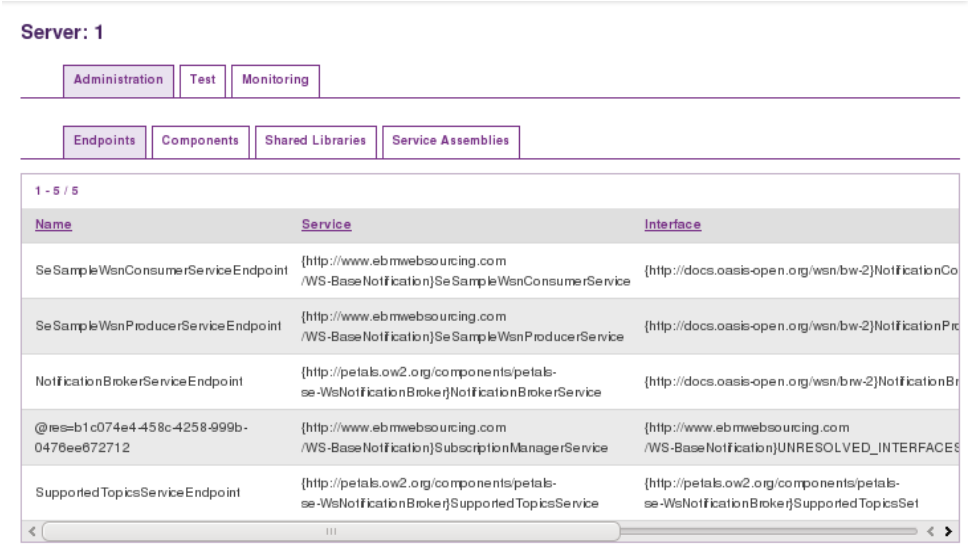


Figure 5.2: New «Subscription resource» endpoint registered on endpoints administration page

And this endpoint will be removed as soon as the «sa-se-wsnconsumer» is stopped and no longer appear in the PEtALS webconsole administration interface, endpoints page.

[1] Notification resources endpoint name format is : "**@res=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx**"

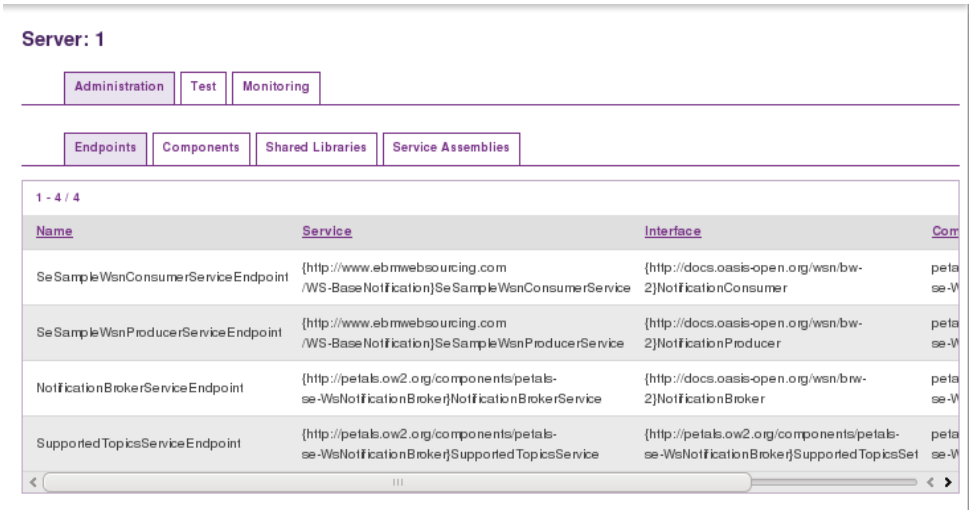


Figure 5.3: The new «Subscription resource» endpoint is no longer registered

5.2 «registerPublisher/destroyRegistration» use-case

5.2.1 scenario

This use-case illustrate the upper parts of UML sequence diagrams presented in section 2.3.3 (fig.2.4 and fig.2.6). A Notification producer sends a «registerPublisher» request to the broker, and starts checking, each 30 seconds, if some consumers - keep in mind that a broker is also a consumer ;) - have subscribes to its topics. Then it sends a «destroyRegistration» request to the registration resource newly created, according to publisher's registration reference returned by the broker.

5.2.2 actors concerned

The WS-Notification actors concerned by this use-case are :

- **Notification Producer** which sends a «registerPublisher» request the broker and sends later a «destroyRegistration» request to the previously created publisher's registration resource whose reference have been returned by the broker
- **Notification Broker** which perform the request and return the reference of the newly created subscription resource
- **PublisherRegistrationManager** resource - the publisher's registration created by the Broker - which will perform the «destroyRegistration» request

5.2.3 how to run the use case

Using the PEtALS webconsole administration interface, services assemblies page, start the «sa-se-wsnproducer» service assembly to make the producer send a «registerPublisher» request to the broker, and stop it to make it send a «destroyRegistration» request to registration resource newly created

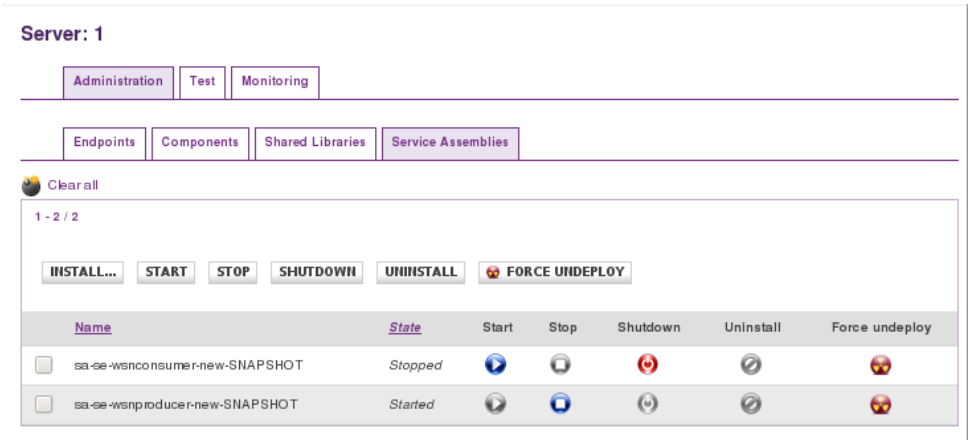


Figure 5.4: sa-se-wsnproducer service assembly has been started

5.2.4 expected results

In the PEtALS webconsole administration interface, endpoints page, a new endpoint must have been added after the «sa-se-wsnproducer» started.

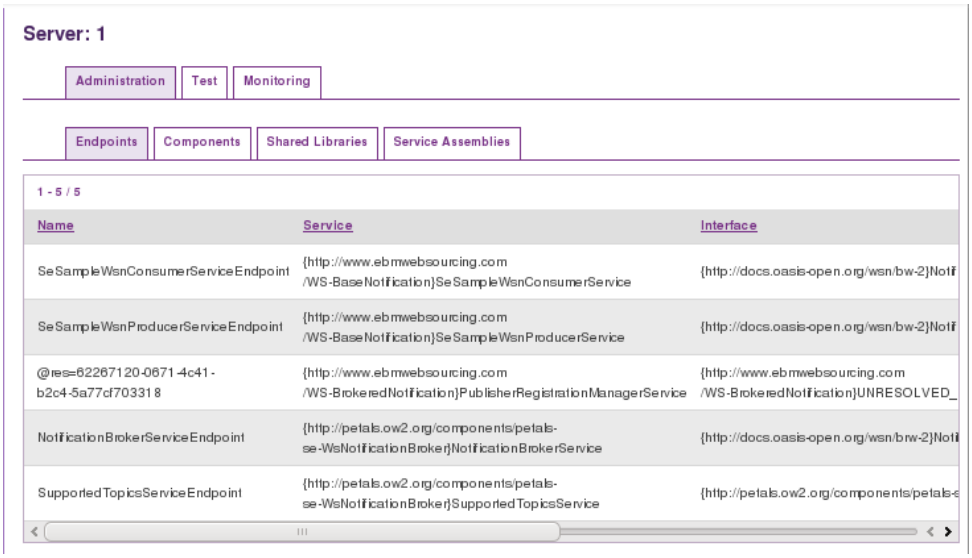


Figure 5.5: New «Subscription resource» endpoint registered on endpoints administration page

And this endpoint will be removed as soon as the «sa-se-wsnproducer» is stopped and no longer appear in the PEtALS webconsole administration interface, endpoints page.

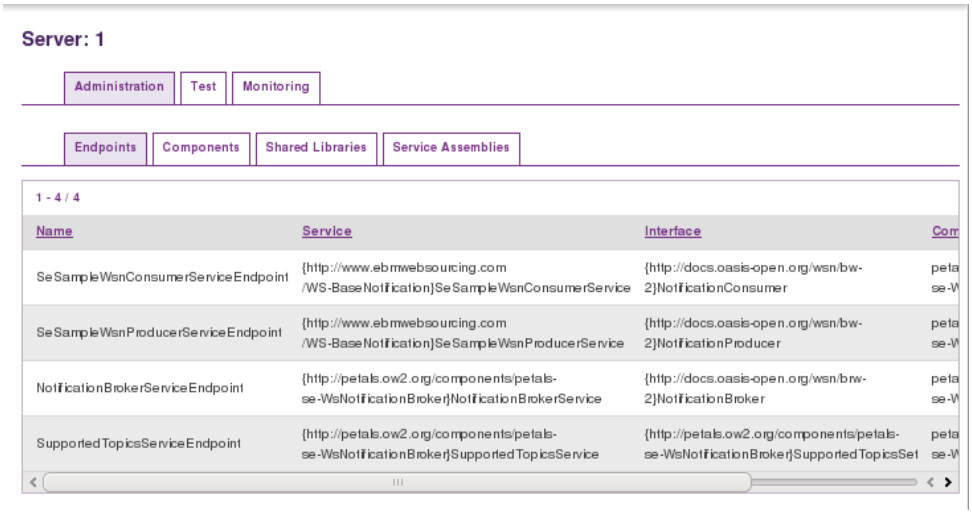


Figure 5.6: The new «Registration resource» endpoint is no longer registered

5.3 «Notify» use-case

5.3.1 scenario

This use-case illustrate the UML sequence diagrams presented in section 2.3.3 (fig.2.2). First, a Notification producer sends a «registerPublisher» request to the broker and start checking each 30s if some consumers have subscribed to its topics - keep in mind that a broker is also a consumer ;). Then a Notification consumer sends a «subscribe» request to the broker which will find the existing publisher's registration and will sends, a s a consequence, a «subscribe» request to the Notification producer (fully described by UML diagram fig.2.3). From now, each 30 seconds, the producer will found broker subscription and sends it a «notify» request. This request will then forwarded to the Notification consumer.

5.3.2 actors concerned

The WS-Notification actors concerned by this use-case are :

- **Notification Producer** which sends a «registerPublisher» request the broker and sends later «notify» requests to the broker, as long as the broker does not unsubscribe to subscription created by the producer.
- **Notification Broker** which perform the requests and return the reference of newly created subscription and registration resources. It will also receive from the producer and sends to the consumer, some «notify» requests
- **PublisherRegistrationManager** resource - the publisher's registration created by the Broker - which will perform the «destroyRegistration» request
- **Notification Consumer** which sends a «subscribe» request the broker and perform «notify» requests sended by the broker.
- **SubscriptionManager** resources - subscriptions created by the Broker and by the producer (as describe in fig.2.3).

5.3.3 how to run the use case

Using the PEtALS webconsole administration interface, services assemblies page, start both «sa-se-wsnproducer» and «sa-se-wsnconsumer» service assemblies. As soon as the Notification producer will find the broker subscription, it will start sending it a «notify» request. Simply stop the service assemblies - in fact just stopping one is enough - to make the producer stop sending «notify» request

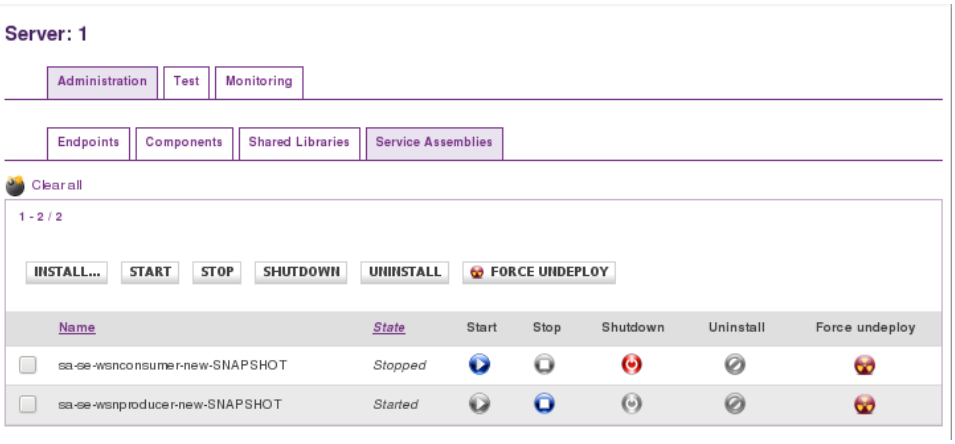


Figure 5.7: all services assemblies - both producer and consumer one - are startted

5.3.4 expected results

☞ In order to see explicit notification message in the console logs, PEtALS ESB must have been configured to display logs for component level. In other words, the following line must be uncommented in «loggers.properties» configuration file : *"logger.Petals.Container.Components.level DEBUG"*

Results are only viewable in the console where the PEtALS Entreprise Service Bus container has been launched. In log message, you should see «notify» requests send by the «petals-se-WsnProducer» component, received then sends by the «petals-se-WsNotificationBroker» component and finally received by the «petals-se-WsnConsumer» component.

```
[...]

[Petals.Container.Components.petals-se-WsnProducer-new]-INFO 2010-08-02 18:26:35,308 Generate notification
on existing Broker subscription ...
[Petals.Container.Components.petals-se-WsnProducer-new]-FINE 2010-08-02 18:26:35,309
        ============ - DEBUG - Subscription found : 862a9ad0-bf55-4395-a134-b670ada845fd
[Petals.Container.Components.petals-se-WsnProducer-new]-FINE 2010-08-02 18:26:35,340
        ==========
[...]

[Petals.Container.Components.petals-se-WsNotificationBroker]-FINE 2010-08-02 18:26:35,350
exchange to process internally received
[Petals.Container.Components.petals-se-WsNotificationBroker]-FINE 2010-08-02 18:26:35,351
I am the "notification_broker" and i have received a notification from a registred notification producer !
[Petals.Container.Components.petals-se-WsNotificationBroker]-FINE 2010-08-02 18:26:35,351
I must forward this notification to all consumer that have subscribed to it

[...]

[Petals.Container.Components.petals-se-WsnConsumer-new]-FINE 2010-08-02 18:26:35,399
Accepting a JBI message with Id : petals:uid:08EA730453C9689DBC2291541372637414
[Petals.Container.Components.petals-se-WsnConsumer-new]-FINEST 2010-08-02 18:26:35,400
Process an exchange as PROVIDER with id : petals:uid:08EA730453C9689DBC2291541372637414
```

```
[Petals.Container.Components.petals-se-WsnConsumer-new]-FINE 2010-08-02 18:26:35,400
Process an exchange managed directly by the component
[Petals.Container.Components.petals-se-WsnConsumer-new]-FINE 2010-08-02 18:26:35,404
             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[Petals.Container.Components.petals-se-WsnConsumer-new]-FINE 2010-08-02 18:26:35,404
 Notification Message received on "onJBIMessage" method. Nothing to do
[Petals.Container.Components.petals-se-WsnConsumer-new]-FINE 2010-08-02 18:26:35,404
 exchange to process internally received
[Petals.Container.Components.petals-se-WsnConsumer-new]-INFO 2010-08-02 18:26:35,436
 hey ! I am the "Notification consumer" component and i have received a notification from the Broker !
[Petals.Container.Components.petals-se-WsnConsumer-new]-INFO 2010-08-02 18:26:35,458
 NotificationMessageHolder content received :
 <?xml version="1.0" encoding="UTF-8"?>
<NotifyContent xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">this The content message
 of the pseudo notification sent by the WsnProducer !!!
</NotifyContent>

[...]
```

PART 6

Externalize Ws-Notification Services

This section will explain how to externalize - mains expose outside the PEtALS Entreprise Service Bus - services provided by the “petals-se-WsNotificationBroker” service engine component and then make a PEtALS ESB container act as a “Ws-Notification” actor respect to other external actors.

6.1 Prerequisites and Restrictions

6.1.1 description

This solution is based on the PEtALS binding component “petals-bc-soap” - more exactly a modified version of it - and it supposes that all requests are sent using “SOAP over http” transport protocol.

6.1.2 additional required petals components

Components[1] required to externalize “WS-Notification Broker” services are :

- **petals-bc-soap-notif** : a modified version[2] of “petals-bc-soap” PEtALS binding component commonly used to externalize an internal JBI Service and make it reachable from outside using “SOAP over HTTP” transport protocol[3].

- **sa-soap-se-WsNotificationBroker** : the service assembly used to configure the resource “petals-bc-soap-notif” PEtALS binding component in order to make “WS-Notification Broker” services available outside the Bus.

- **External SOAP client** (Optional) : a any SOAP[4] client able to create and send SOAP message to a given URL.

6.2 Installation

These additional PEtALS components - and not the external SOAP client of course - can be installed and started, using the PEtALS webconsole, as described in previous sections of this document.

Once installed the “petals-bc-soap-notif” component should be listed in the webconsole component Administration page (fig. - 6.1)

[1] these extra components are actually only located in the “sandbox/tdejean” folder on the PEtALS SVN repository and also - link - on the forge of the European research project named “Synergy”

[2] It automatically externalizes any WS-Resource created and registered, at runtime, in PEtALS bus as a result of a “Subscribe” or “RegisterPublisher” request

[3] For more details, see related documentation on petalslink wiki : `http://doc.petalslink.com/`

[4] SOAPUI is a good choice even if it is not the only one. Its web site : `http://www.soapui.org`

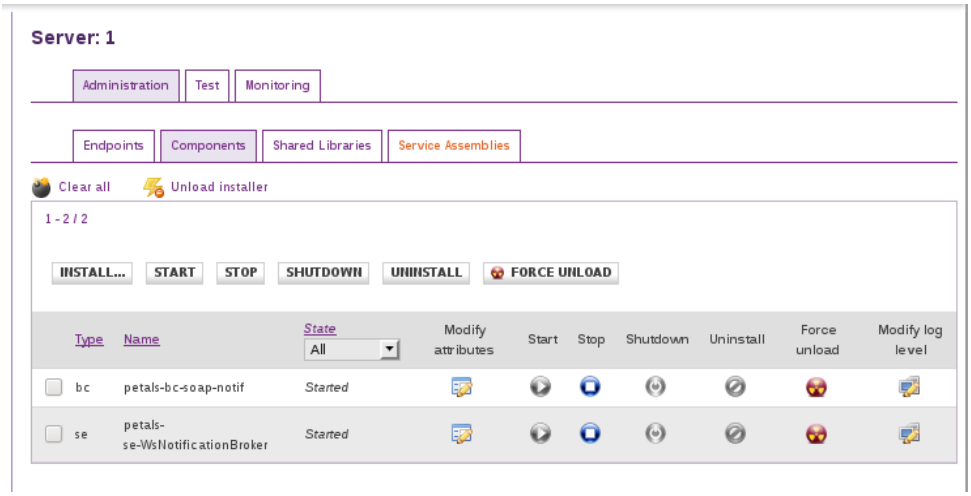


Figure 6.1: BC-SOAP-Notif installation on webconsole component administration page

As well as the “sa-soap-se-WsNotificationBroker” service assembly that should appear in the list of the managed service assembly page(fig - 6.2)

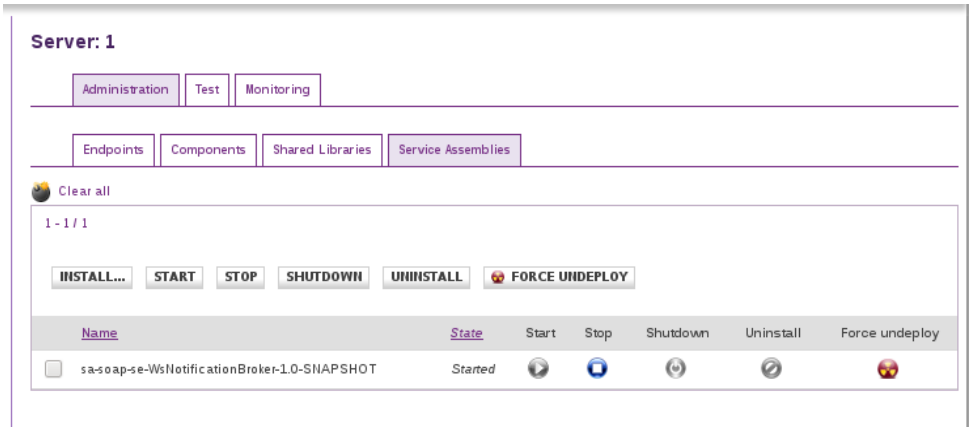


Figure 6.2: BC-SOAP-Notif installation on webconsole component administration page

To validate the installation process, the web homepage provided by the “petals-bc-soap-notif” component (fig. - 6.3) must be available when you request it[5]

[5] common url should be : http://localhost:8085/. Otherwise check the bc-soap-notif jbi.xml file to determinate the configured port number

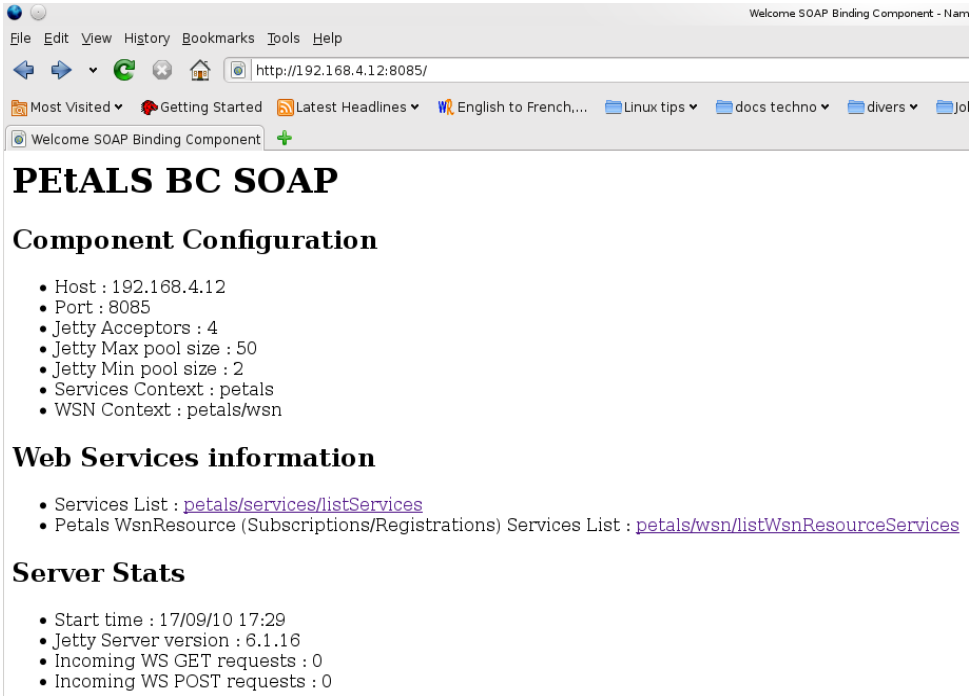


Figure 6.3: The web homepage provided by the bc-soap component when started

6.3 Request WS-Notification service from outside

6.3.1 List of reachable WS-Notification services and resources

Available services list

From the bc-soap component web homepage you can show which services are available and can be requested from any Soap client, according to their associated wsdl file description. Currently two services shoul be listed - and so reachable - which are “SupportedTopicsService” service and the “NotificationBrokerService” service (fig. 6.4).

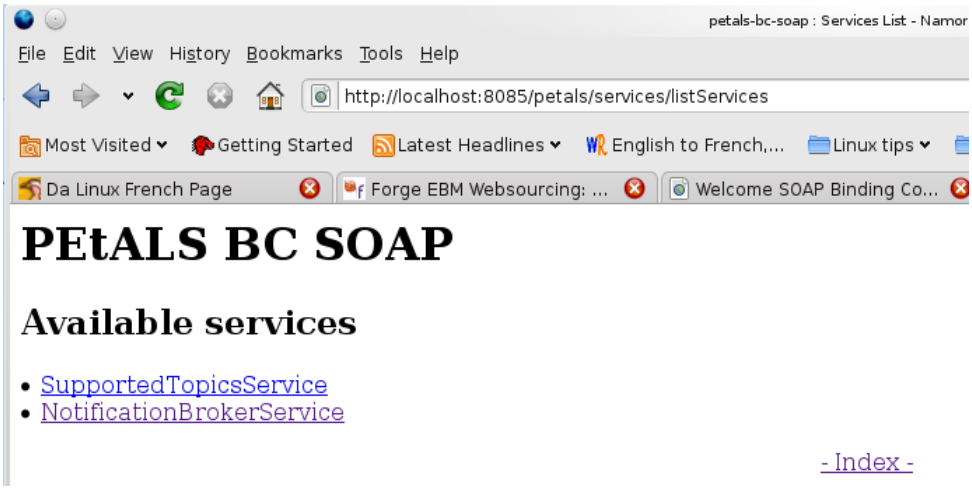


Figure 6.4: The bc-soap “available service” web page

- **The “SupportedTopicsService”** service provides the supported Topics set currently supported by PEtALS ESB - through its petals-se-WsNotificationBroker service engine component - and which a Notifica-

tionConsumer can subscribe to[6].

- **The "NotificationBrokerService"** service provides the common Ws-Notification operation provided by a Notification Broker : Subscribe, Unsubscribe, Notify, RegisterPublisher, DestroyRegistration, ... See related OASIS Specifications for more details

Subscription/registration WS-Notification resources list

And A second page, also available from the bc-soap component web homepage show which Subscription/Registration are currently "managed" by the PEtALS ESB and can then be removed by sending to them appropriate requests (Unsubscrabe/DestroyRegistration)

Figure 6.5: The bc-soap "available ws-notification resources" web page

6.3.2 A Simple test : "subscribe/unsubscribe" from an external NotificationConsumer Actor

Using the SOAPUI client software, a "subscribe" request the an "Unsubscribe" request are sent to the PEtALS ESB using the "BC-SOAP" exposed url address. The creation and the destruction of the Subscription resource can be check using both the bc-soap web homepage (external view of PEtALS services) and the PEtALSwebconsole (internal view of PEtALS services).

Create and send "Subscribe" request from SOAPUI

Create the payload and send the "subscribe" request according the related wsdl description and the currently supported topics (fig. 6.6)

[6] Even if the "getSupportedTopics" does not expect any specific payload content, you must add an empty xml node "<empty/>" in the body part of the sent SOAP message

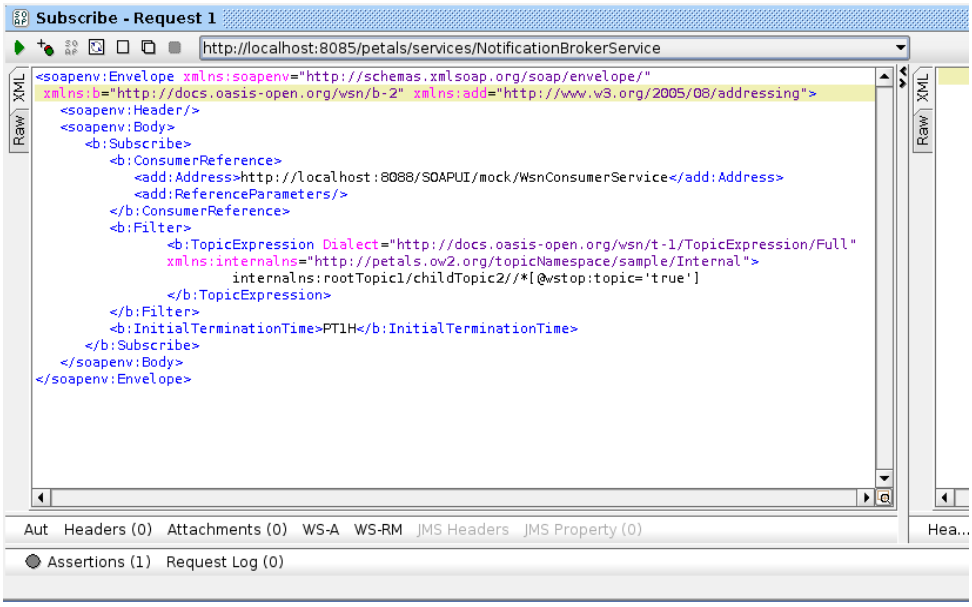


Figure 6.6: Performing a “Subscribe” request from SOAPUI software (soap client)

If no error have been made ins the request a simple “UnsubscribeResponse” payload message must be returned (fig. 6.7)

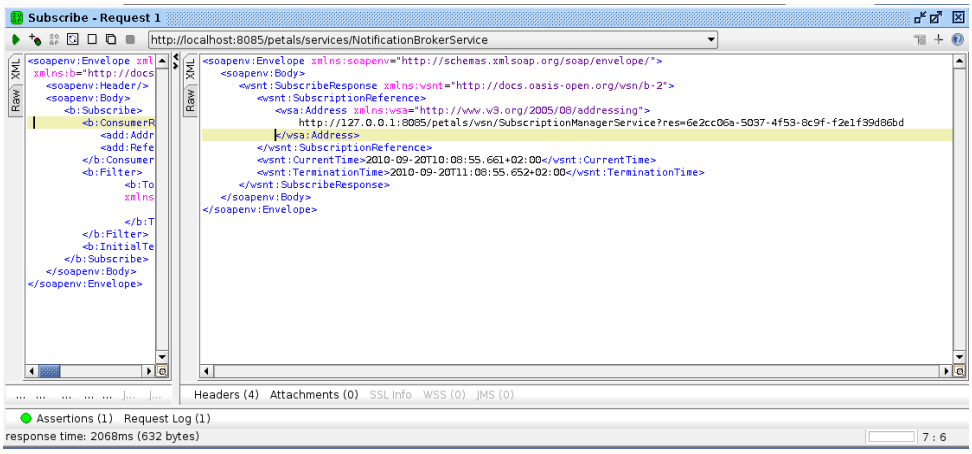


Figure 6.7: Returned “Subscribe” response from PEtALS ESB (through bc-soap component)

Check the creation of the associated WS-Resource

Once the “subscribe” has been performed and the associated response returned the subscription resource - its associated endpoint - is now registered in the PEtALS ESB registry and automatically externalize through the “petals-bc-soap” component as a webservice whose interface is the one defined as the Subscription-Manager Interface[7]

A subscription notification resource is now listed in the bc-soap “available resource” webpage (fig. 6.8)

[7] See p.31 of OASIS specification[3]

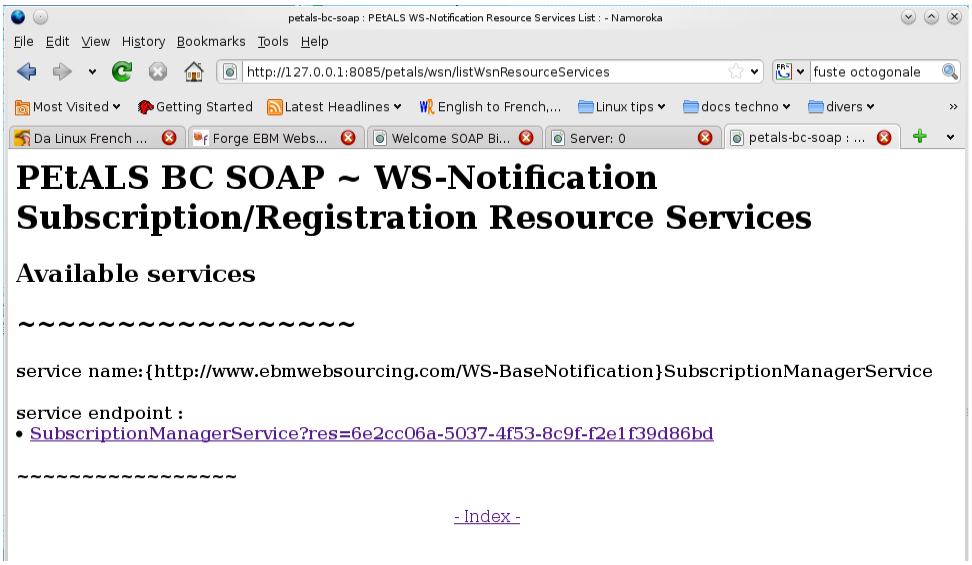


Figure 6.8: The subscription “ws-resource” is now listed - and reachable - in the bc-soap “available resource” page

And a new internal endpoint appears in the known endpoint list, in the related PEtALS ESB webconsole page (fig. 6.9)

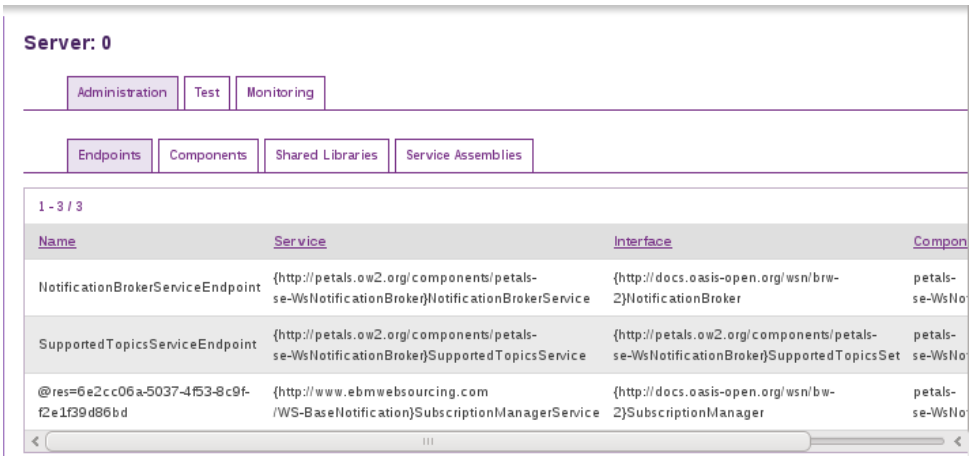


Figure 6.9: And the subscription “ws-resource” is also listed in the webconsole endpoint section administration page

Remove the subscription sending “unsubscribe” request

Then to end this use case, an “unsubscribe” request is now sent the Subscription “WS-Resource” using the soap address previously returned in the “subscribe” response payload.

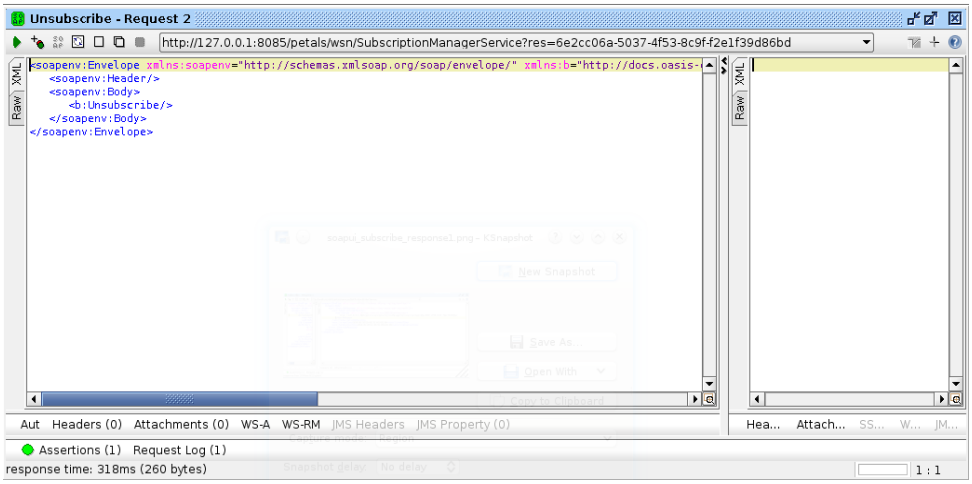


Figure 6.10: Performing a “Unsubscribe” request from SOAPUI software (soap client)

And expected response to the request must be a “UnsubcribeResponse” payload as describe in the OASIS specification[3]

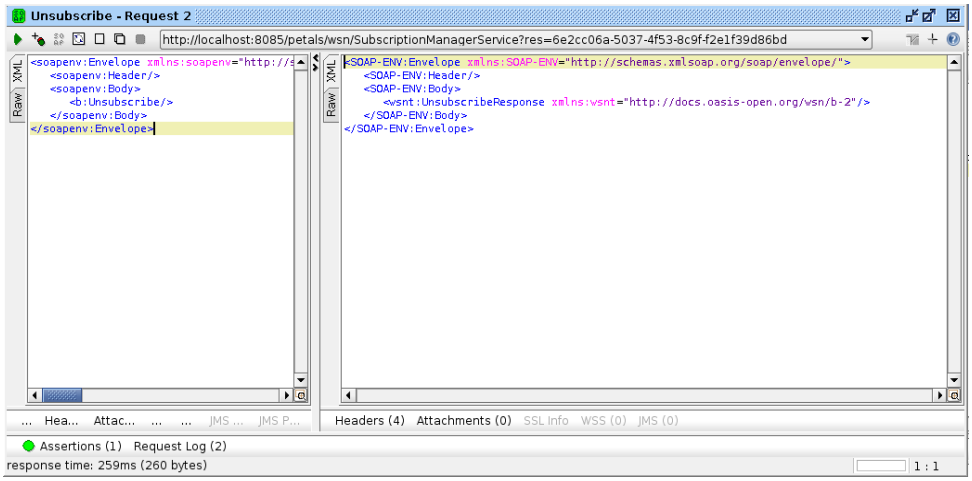


Figure 6.11: Returned “Unsubscribe” response from PEtALS ESB (through bc-soap component)

6.3.3 And Next ?

Well that ups to you. If you have well understood hwo does it work, you can now built your own use cases, using your own SOAP client. Have fun !

PART A

Component wsdl file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://petals.ow2.org/components/petals-se-WsNotificationBroker"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsn-bw="http://docs.oasis-open.org/wsn/bw-2"
 xmlns:wsn-brw="http://docs.oasis-open.org/wsn/brw-2"
 xmlns:wsn-br="http://docs.oasis-open.org/wsn/br-2"
 xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
 xmlns:wsn-b="http://docs.oasis-open.org/wsn/b-2"
 name="WS-BrokeredNotification"
 targetNamespace="http://petals.ow2.org/components/petals-se-WsNotificationBroker">

 <!-- ================================================== -->
 <!-- ================== imports ======================= -->
 <!-- ================================================== -->

 <wsdl:import location="http://docs.oasis-open.org/wsn/brw-2.wsdl"
   namespace="http://docs.oasis-open.org/wsn/brw-2" />
 <wsdl:import location="http://docs.oasis-open.org/wsn/bw-2.wsdl"
   namespace="http://docs.oasis-open.org/wsn/bw-2" />
 <!-- ================================================== -->
 <!-- ========= Message part definition ================ -->
 <!-- ================================================== -->

 <wsdl:message name="EmptyMessage" />

 <wsdl:message name="GetSupportedTopicsResponseMessage">
   <wsdl:part name="body" element="wstop:TopicSet" />
 </wsdl:message>

 <wsdl:message name="AddNewSupportedTopicMessage">
   <wsdl:part name="body" element="wsn-b:TopicExpression" />
 </wsdl:message>

 <!-- ================================================ -->
 <!-- =========  PortType definition ================= -->
 <!-- ================================================ -->

 <wsdl:portType name="SupportedTopicsSet">
  <wsdl:documentation>
    This port type defines a Web service that provides
    information about Topics currently supported by the Broker.
    Also allow to add new supported topic during runtime (EXPERIMENTAL)
  </wsdl:documentation>

  <wsdl:operation name="GetSupportedTopics">
   <wsdl:input message="tns:EmptyMessage" />
   <wsdl:output message="tns:GetSupportedTopicsResponseMessage" />
  </wsdl:operation>

  <wsdl:operation name="AddNewSupportedTopic">
   <wsdl:input message="tns:AddNewSupportedTopicMessage" />
  </wsdl:operation>
 </wsdl:portType>

 <!-- ============================================== -->
 <!-- ========== Binding definition ================ -->
 <!-- ============================================== -->

 <!-- NotificationBroker binding definition -->
 <wsdl:binding name="NotificationBrokerBinding" type="wsn-brw:NotificationBroker">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
```

```
  <wsdl:operation name="Notify">
   <soap:operation
     soapAction="http://docs.oasis-open.org/wsn/bw-2/NotificationConsumer/Notify"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
  </wsdl:operation>
  <wsdl:operation name="Subscribe">
   <soap:operation
     soapAction="http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/SubscribeRequest"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="ResourceUnknownFault">
   <soap:fault use="literal" name="ResourceUnknownFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidFilterFault">
   <soap:fault use="literal" name="InvalidFilterFault"/>
  </wsdl:fault>
  <wsdl:fault name="TopicExpressionDialectUnknownFault">
   <soap:fault use="literal" name="TopicExpressionDialectUnknownFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidTopicExpressionFault">
   <soap:fault use="literal" name="InvalidTopicExpressionFault"/>
  </wsdl:fault>
  <wsdl:fault name="TopicNotSupportedFault">
   <soap:fault use="literal" name="TopicNotSupportedFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidProducerPropertiesExpressionFault">
   <soap:fault use="literal" name="InvalidProducerPropertiesExpressionFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidMessageContentExpressionFault">
   <soap:fault use="literal" name="InvalidMessageContentExpressionFault"/>
  </wsdl:fault>
  <wsdl:fault name="UnacceptableInitialTerminationTimeFault">
   <soap:fault use="literal" name="UnacceptableInitialTerminationTimeFault"/>
  </wsdl:fault>
  <wsdl:fault name="UnrecognizedPolicyRequestFault">
   <soap:fault use="literal" name="UnrecognizedPolicyRequestFault"/>
  </wsdl:fault>
  <wsdl:fault name="UnsupportedPolicyRequestFault">
   <soap:fault use="literal" name="UnsupportedPolicyRequestFault"/>
  </wsdl:fault>
  <wsdl:fault name="NotifyMessageNotSupportedFault">
   <soap:fault use="literal" name="NotifyMessageNotSupportedFault"/>
  </wsdl:fault>
  <wsdl:fault name="SubscribeCreationFailedFault">
   <soap:fault use="literal" name="SubscribeCreationFailedFault"/>
  </wsdl:fault>
 </wsdl:operation>
 <wsdl:operation name="GetCurrentMessage">
  <soap:operation
    soapAction="http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/GetCurrentMessageRequest"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="ResourceUnknownFault">
   <soap:fault use="literal" name="ResourceUnknownFault"/>
  </wsdl:fault>
  <wsdl:fault name="TopicExpressionDialectUnknownFault">
   <soap:fault use="literal" name="TopicExpressionDialectUnknownFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidTopicExpressionFault">
   <soap:fault use="literal" name="InvalidTopicExpressionFault"/>
  </wsdl:fault>
  <wsdl:fault name="TopicNotSupportedFault">
   <soap:fault use="literal" name="TopicNotSupportedFault"/>
```

```
  </wsdl:fault>
  <wsdl:fault name="NoCurrentMessageOnTopicFault">
   <soap:fault use="literal" name="NoCurrentMessageOnTopicFault"/>
  </wsdl:fault>
  <wsdl:fault name="MultipleTopicsSpecifiedFault">
   <soap:fault use="literal" name="MultipleTopicsSpecifiedFault"/>
  </wsdl:fault>
 </wsdl:operation>
 <wsdl:operation name="RegisterPublisher">
  <soap:operation
    soapAction="http://docs.oasis-open.org/wsn/brw-2/RegisterPublisher/RegisterPublisherRequest"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
   <wsdl:fault name="ResourceUnknownFault">
    <soap:fault use="literal" name="ResourceUnknownFault"/>
   </wsdl:fault>
   <wsdl:fault name="InvalidTopicExpressionFault">
    <soap:fault use="literal" name="InvalidTopicExpressionFault"/>
   </wsdl:fault>
   <wsdl:fault name="TopicNotSupportedFault">
    <soap:fault use="literal" name="TopicNotSupportedFault"/>
   </wsdl:fault>
   <wsdl:fault name="PublisherRegistrationRejectedFault">
    <soap:fault use="literal" name="PublisherRegistrationRejectedFault"/>
   </wsdl:fault>
   <wsdl:fault name="PublisherRegistrationFailedFault">
    <soap:fault use="literal" name="PublisherRegistrationFailedFault"/>
   </wsdl:fault>
   <wsdl:fault name="UnacceptableInitialTerminationTimeFault">
    <soap:fault use="literal" name="UnacceptableInitialTerminationTimeFault"/>
   </wsdl:fault>
  </wsdl:operation>
 </wsdl:binding>

 <!-- SupportedTopicsSet portType Service binding definition -->
 <wsdl:binding name="SupportedTopicsSetBinding" type="tns:SupportedTopicsSet">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="GetSupportedTopics">
   <soap:operation
     soapAction="http://www.ebmwebsourcing.com/WS-BrokeredNotification/GetSupportedTopics"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="AddNewSupportedTopic">
   <soap:operation
     soapAction="http://www.ebmwebsourcing.com/WS-BrokeredNotification/AddNewSupportedTopic"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
  </wsdl:operation>
 </wsdl:binding>

 <!-- ============================================ -->
 <!-- =========== Service part definition ========= -->
 <!-- ============================================ -->

 <wsdl:service name="NotificationBrokerService">
  <wsdl:port name="NotificationBrokerServiceEndpoint" binding="tns:NotificationBrokerBinding">
   <soap:address location="NotificationBrokerServiceEndpoint"/>
  </wsdl:port>
 </wsdl:service>

 <wsdl:service name="SupportedTopicsService">
  <wsdl:port name="SupportedTopicsServiceEndpoint" binding="tns:SupportedTopicsSetBinding">
   <soap:address location="SupportedTopicsServiceEndpoint"/>
```

```
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

PART B

WS-Notification Broker "SupportedTopicsSet" file

```
<?xml version="1.0" encoding="UTF-8"?>
<wstop:TopicSet xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
 xmlns:internalns="http://petals.ow2.org/topicNamespace/sample/Internal"
 xmlns:externalns="http://petals.ow2.org/topicNamespace/sample/External"
 xmlns:petals="http://petals.ow2.org/topic"
 xmlns:genesis="http://petals.ow2.org/topicNamespace/GenesisDemonstrator"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <internalns:rootTopic1 wstop:topic="true">
  <internalns:childTopic1 wstop:topic="true"/>
  <internalns:childTopic2>
   <internalns:grandChildTopic21 wstop:topic="true"/>
  </internalns:childTopic2>
  <internalns:childTopic3 wstop:topic="true"/>
 </internalns:rootTopic1>
 <externalns:rootTopic2>
  <externalns:childTopic1>
   <externalns:grandChildTopic11 wstop:topic="true">
    <externalns:grandGrandChildTopic111 wstop:topic="true"/>
    <externalns:grandGrandChildTopic112 wstop:topic="true"/>
   </externalns:grandChildTopic11>
  </externalns:childTopic1>
  <externalns:childTopic2 wstop:topic="true">
   <externalns:grandChildTopic21 wstop:topic="true"/>
   <externalns:grandChildTopic22 wstop:topic="true"/>
  </externalns:childTopic2>
 </externalns:rootTopic2>
 <genesis:rootTopic>
  <genesis:childTopic1>
   <genesis:grandChildTopic1 wstop:topic="true">
    <genesis:grandGrandChildTopic1 wstop:topic="true"/>
    <genesis:grandGrandChildTopic2 wstop:topic="true"/>
   </genesis:grandChildTopic1>
  </genesis:childTopic1>
  <genesis:childTopic2 wstop:topic="true">
   <genesis:grandChildTopic1 wstop:topic="true"/>
   <genesis:grandChildTopic2 wstop:topic="true"/>
  </genesis:childTopic2>
 </genesis:rootTopic>
 <petals:component>
  <petals:cdk>
   <petals:producer>
    <petals:in wstop:topic="true"/>
    <petals:out wstop:topic="true"/>
    <petals:status wstop:topic="true"/>
    <petals:fault wstop:topic="true"/>
   </petals:producer>
  </petals:cdk>
  <petals:bc-soap>
   <petals:soapfault wstop:topic="true"/>
  </petals:bc-soap>
  <petals:bc-mail>
   <petals:smtpfault wstop:topic="true"/>
  </petals:bc-mail>
 </petals:component>
 <petals:kernel>
  <petals:routagefault wstop:topic="true"/>
  <petals:transporterfault wstop:topic="true"/>
 </petals:kernel>
</wstop:TopicSet>
```

PART C

WS-Notification actor request payloads xml files

C.1 WsnConsumer «subscribe» request payload

```
<?xml version="1.0" encoding="UTF-8"?>
<wsnt:Subscribe
 xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
 xmlns:wsa="http://www.w3.org/2005/08/addressing"
 targetnamespace ="http://docs.oasis-open.org/wsn/b-2">
 <wsnt:ConsumerReference>
  <wsa:Address>
   http://www.ebmwebsourcing.com/WS-BaseNotification::SeSampleWsnConsumerService@SeSampleWsnConsumerServiceEndpoint
  </wsa:Address>
 </wsnt:ConsumerReference>
 <wsnt:Filter>
  <wsnt:TopicExpression Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Full"
   xmlns:internalns="http://petals.ow2.org/topicNamespace/sample/Internal">
    internalns:rootTopic1/childTopic2//*[@wstop:topic='true']
  </wsnt:TopicExpression>
 </wsnt:Filter>
 <wsnt:InitialTerminationTime>PT1H</wsnt:InitialTerminationTime>
 <wsnt:SubscriptionPolicy/>
</wsnt:Subscribe>
```

C.2 WsnProducer «registerPublisher» request payload

```
<?xml version="1.0" encoding="UTF-8"?>
<wsn-br:RegisterPublisher
 xmlns:wsn-br="http://docs.oasis-open.org/wsn/br-2"
 xmlns:wsa="http://www.w3.org/2005/08/addressing"
 targetnamespace ="http://docs.oasis-open.org/wsn/br-2">
 <wsn-br:PublisherReference>
  <wsa:Address>
   http://www.ebmwebsourcing.com/WS-BaseNotification::SeSampleWsnProducerService@SeSampleWsnProducerServiceEndpoint
  </wsa:Address>
 </wsn-br:PublisherReference>
 <wsn-br:Topic Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Full"
  xmlns:internalns="http://petals.ow2.org/topicNamespace/sample/Internal">
   internalns:rootTopic1//*[@wstop:topic='true']
 </wsn-br:Topic>
 <wsn-br:Demand>false</wsn-br:Demand>
 <wsn-br:InitialTerminationTime>2009-12-25T00:00:00.00000Z</wsn-br:InitialTerminationTime>
</wsn-br:RegisterPublisher>
```

C.3 WsnProducer «notify» request payload

```
<?xml version="1.0" encoding="UTF-8"?>
<wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
 xmlns:wsa="http://www.w3.org/2005/08/addressing"
 targetnamespace="http://docs.oasis-open.org/wsn/b-2">
 <wsnt:NotificationMessage>
  <wsnt:Topic Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete"
   xmlns:internalns="http://petals.ow2.org/topicNamespace/sample/Internal">
    internalns:rootTopic1/childTopic2/grandChildTopic21
```

```
  </wsnt:Topic>
  <wsnt:Message>
   <NotifyMsgContent>
    <Details>
      this is the detail description of the notification sent !
    </Details>
   </NotifyMsgContent>
  </wsnt:Message>
 </wsnt:NotificationMessage>
</wsnt:Notify>
```

Bibliography

[1] D. Box, F. Curbera, et al. Web services addressing (WS-Addressing), 2004.

[2] D. Chappell and L. Liu. Web Services Brokered Notification 1.3 (WS-BrokeredNotification) OASIS Standard. Technical report, Tech. rep., OASIS, 2006.

[3] S. Graham, D. Hull, and B. Murray. Web Services Base Notification 1.3 (WS-BaseNotification) OASIS Standard. Technical report, Tech. rep., OASIS, 2006.

[4] S. Graham, A. Karmarkar, J. Mischkinsky, I. Robinson, and I. Sedukhin. Web services resource 1.2 (WS-Resource). *OASIS Standard, April*, 2006.

[5] S. Graham and J. Treadwell. Web Services Resource Properties 1.2 (WS-ResourceProperties) OASIS Standard. Technical report, Tech. rep., OASIS, 2006.

[6] L. Liu and S. Meder. Web Services Base Faults 1.2 (WS-BaseFaults). *OASIS Committee Specification*, 2006.

[7] L. Srinivasan and T. Banks. Web Services Resource Lifetime 1.2 (WS-ResourceLifetime) OASIS Standard. Technical report, Tech. rep., OASIS, 2006.

[8] W. Vambenepe, S. Graham, and P. Niblett. Web Services Topics 1.3 (WS-Topics) OASIS Standard 2. Technical report, Tech. rep., OASIS, 2006.