PEtALS Platform Distribution

This document is the user guide of the PEtALS platform distribution. This release targets PEtALS users who want to build their services architecture on distributed infrastructure

PEtALS Team
Roland NAUDIN <roland.naudin@ebmwebsourcing.com>
Christophe HAMERLING <christophe.hamerling@ebmwebsourcing.com>

- March 2009 -

Table of Contents

List of Figures

PEtALS

PEtALS is the highly distributed OW2 Java Business Integration (JBI) bus.(See JSR-208 specifications for further details on JBI).

Along the time, PEtALS has increased its coverage amongst high-value domains like clustering, robustness, availibility, performance... Moreover, PEtALS relies entirely on its OW2 partner technology, the Fractal component model, which brings to its architecture a strong modularity. Please visit the Fractal web site for further details at http://fractal.objectweb.org.

Since the version 2.1, the PEtALS team have decided to exploit Fractal leverage by delivering various PEtALS distributions. Each distribution is packaged and customized to be addressed to a specific audience.

This distribution targets the usage of PEtALS in a distributed infrastructure. It provides security and transaction support. It permits various optimizations on data transfers and tuning on the administration and exploitation of your JBI environment.

Chapter 1. Presentation

1.1. What is PEtALS?

PEtALS helps you to integrate your Enterprise Business Units in order to provide a coherent and rational global solution. With PEtALS, you can design your new applications by building a composition of existing ones and new ones.

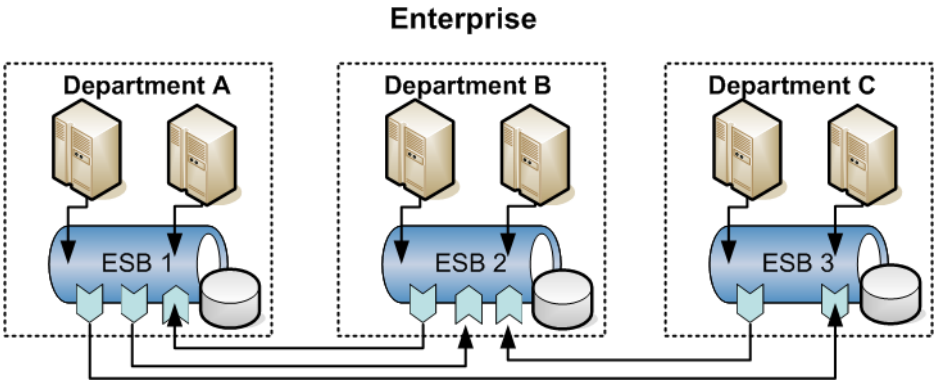
All your applications expose their business logic by exposing services. Thus, each new composed application becomes an add-on to the pool of reusable applications.

This concept is well known as **Service Oriented Architecture - SOA**.

PEtALS offers a solid backbone for your enterprise Information System and acts as a Bus, a place where all your data are exchanged. PEtALS connects services to each others. PEtALS is an Enteprise Service Bus (ESB).

Please visit our Web site for further details at http://petals.ow2.org.

Figure 1.1. PEtALS is an ESB fully JBI compliant



1.2. Acronyms

Along the document, we are using common JBI acronyms :

- **SL** : a Shared Libraries artifact.
- **BC** : a Binding Component artifact.
- **SE** : a Service Engine component artifact.
- **SA** : a Service Assembly artifact.
- **SU** : a Service Unit artifact.
- **MEP** : Message Exchange Pattern, JBI defines 4 possible MEP to invoke a service.

1.3. JBI Components

In order to build your business solutions, PEtALS proposes a large bunch of components. Each component is downloadable independently on the PEtALS website.

1.3.1. Binding Components

The BCs are bridges to interact with external services, commonly via a communication protocol.

They provide transfers of incoming requests from external service consumers to the JBI environment, as well as the transfer of outgoing requests to external service providers.

The current proposed BCs are :

- **Filetransfer** : transfers files from/to local directories.
- **FTP** : transfers files from/to remote directories.
- **TCP/IP** : transfers files from/to TCP/IP servers.
- **HTTP** : transfers files from/to HTTP servers.
- **Mail** : interacts with e-mail servers to receive/send emails.
- **XMPP** : transfers files from/to XMPP servers.
- **JMS** : interacts as JMS client with any JMS provider to receive/send JMS messages.
- **Soap** : invokes external Web Services and expose JBI services as Web Services.
- **XQuare** : interacts with databases to store and retrieve data.
- **Ejb** : exposes remote EJBs as JBI service.

1.3.2. Service Engines

The SEs are components providing services to the JBI environment. A large domain of services can be designed: routing, orchestration, transformation... You can provide your own business services too.

The current proposed SEs are :

- **EIP** : provides routing and chaining of services based on Enterprise Integration Patterns.
- **Orchestra** : provides orchestration of services based on BPEL language.
- **Drools** : process rules on different language to provide routing services.
- **Bonita** : provides services to interacts with the Bonita Workflows.
- **XSLT** : transforms XML documents to XML or another forms, based on XSL sheets.
- **Validation** : validates XML documents against XML schemas.
- **CSV** : transforms CSV documents to XML documents.
- **Transcoder** : transcodes XML fields against database fields.
- **Quartz** : provides scheduling services.
- **Script** : supports scripting to provide your own business logic.
- **Pojo** : exposes Java classes as services to provide your own business logic.
- **JSR-181** : exposes Java classes as services to provide your own business logic.
- **RMI** : brings the access of the JBI component context to a remote RMI client, in order to provide your remote business logic.

1.3.3. Samples

Several sample components are delivered with PEtALS to ease the tests when building your solutions:

- **SampleClient** : a simple graphical client to invoke the services exposed in the bus.
- **HelloWorld** : the famous HelloWold paradigm exposed as a service...
- **Clock** : provides a clock service.
- **PerfConsumer/PerfProvider** : two components running together to test container performances.

Chapter 2. PEtALS container

2.1. Pre-Requisites

- **Java** : to run PEtALS, you need at least a Java JRE 1.5 distribution.

  The Sun JVM can be downloaded at : http://java.sun.com/javase/downloads/index_jdk5.jsp.

- **Ant** : to run the provided use cases, you need to install an Ant distribution.

  Apache Ant can be downloaded at : http://ant.apache.org/bindownload.cgi.

2.2. Installation

PEtALS does not require any installer to be installed. Just unzip the provided archive where you want to install it.

Generally, PEtALS find by itself its directory location. On some systems, the installation path can not be automatically found, in such case you must set the *PETALS_HOME* environment variable :

- On Unix/Linux systems :

```
# export PETALS_HOME=your_installation_path
```

- On Windows system :

```
# set PETALS_HOME=your_installation_path
```

2.3. Directory structure

- ant : includes a sample file containing exemples to illustrate the usage of JBI specialized Ant tasks.
- `bin` : includes scripts to launch PEtALS on Unix (*.sh) or Windows (*.bat) systems.
- `conf` : includes PEtALS configuration files.
- `install` : auto-loader directory; put a JBI component (SE or BC) or a SA to have them automaticaly installed, deployed and started in the PEtALS container.
- `installed` : components and services assemblies are automaticaly copied into this directory after a succesful installation/deployment.
- `lib` : includes libraries required by PEtALS system.
- `licenses` : includes the licenses of all the libraries used in PEtALS container
- `logs` : includes logs generated during PEtALS execution.
- `lost+found` : contains lost JBI elements (components or SAs no more referred in the PEtALS repository).
- `repository` : includes the resources;libraries, config files... of the installed/deployed components, SLs and SAs.
- `schema` : includes useful XML schemas.
- `work` : used by PEtALS engine to store internal resources during runtime.

2.4. Starting PEtALS

1. Go to `$PETALS_HOME/bin`

2. Launch `startup.sh` or `startup.bat` :

```
# ./startup.sh
```

You can launch PEtALS in console mode, to be able to interact with it via the terminal. In this case, use the option -C in the terminal :

```
# ./startup.sh -C
```

2.5. Stopping PEtALS

There is several ways to stop PEtALS :

1. If PEtALS is launched in a simple terminal, type **<ctrl>**+ **c** to stop PEtALS.
2. If PEtALS is launched in console mode, just type **q** or **x** and **<enter>** in the terminal :

```
# q
```

By typing **q**, PEtALS is stopped; components, SAs ant SLs life-cycle states are hold.

By typing **x**, PEtALS is shut down; components, SAs and SLs are uninstalled/undeployed.

3. If PEtALS is launched in background, launch the `stop.sh` script :

```
# ./stop.sh
```

4. If PEtALS is launched in background mode, launch the `shutdown.sh` script :

```
# ./shutdown.sh
```


Note

Stopping PEtALS means to stop the running components and SAs without changing their life-cycle state. The endpoints exposed by the components are still registered in the distributed JNDI directory of the PEtALS domain. At your next start, the components and SAs would be recovered to their previous state, and all persisted requests would be processed. Shutting down PEtALS means to undeploy and uninstall all the components and SAs loaded in the container AND unregister the stopped node from the PEtALS domain. It is useful to clean up your container.

Chapter 3. Configuration

3.1. Introduction

All the PEtALs configuration resources are located in the `conf` directory.

- `topology.xml` : defines the configuration and resources involved in the PEtALS topology.
- `server.properties` : defines the local configuration, customization and resources of your JBI container.
- `logger.properties` : defines the logging configuration of your container.
- `log4j.properties` : defines the logging configuration of third part libraries using log4j logger.
- router.cfg : configures the modules used by the PEtALS container router.
- `stack.xml` : configures JGROUPS communication stacks, which is used for the domain management, reserved to PEtALS specialists.
- `julia.cfg` : contains the FRACTAL configuration, reserved to PEtALS specialists.
- `login.properties` : security file. See Security section for further details.
- `user-passwords.properties` : security file. See Security section for further details.
- `groups.properties` : security file. See Security section for further details.
- `authorization.cfg` : security file. See Security section for further details.

3.2. PEtALS topology - topology.xml

PEtALS is aiming distributed environments. To fullfill this orientation, PEtALS servers of a common domain are defined in a shared file; the `topology.xml` file.

PEtALS servers that communicate each other are gathered in a `Domain`. Each configuration of these servers is referenced in the file,under an unique `Domain` element. Servers declared under different domains will never communicate each other.

As PEtALS introduces the concept of *private* and *public* services, servers are referenced under `Sub-domains`. Thus, a private business service exposed on a server is only visible by servers of the same sub-domain.

The topology configuration looks like this sample:

```
<topology>
  <domain name="PETALS domain" ...>
    <multicast.../>
    <jndi.../>

    <sub-domain name="SubDomain 1">
      <container name="10"...>
      <container name="11"...>
    </sub-domain>

    <sub-domain name="SubDomain 2">
      <container name="20"...>
      <container name="21"...>
    </sub-domain>

  </domain>
</topology>
```

Caution

In a `static` domain, the `topology.xml` file must be the same for all the containers belonging to a PEtALS domain. That is to say, it must be copied under all the `$PETALS_HOME/conf` folders of the servers

Note

You can find samples of this file in the `$PETALS_HOME/conf` folder.

Note

More information are available in the `$PETALS_HOME/schemas/petalsTopology.xsd` schema.

3.2.1. Domain configuration

The PEtALS topology is identified by a PEtALS `Domain`. This configuration part defines the domain behavior and the common resources of the domain.

Here is an example of domain definition:

```
<domain mode="dynamic" name="PEtALS2">
    <multicast>
        <ip>224.7.65.52</tns:ip>
        <port>8000</port>
    </multicast>
    <jndi>

 <factory>org.objectweb.petals.communication.jndi.client.naming.NamingContextFactory</factory>
        <host>localhost</host>
        <port>9999</port>
    </jndi>

    <sub-domain.../>

</domain>
```

`domain` — *mode* attribute: The mode of the domain, possible values are *static*, *dynamic* and *standalone*. This field defines the behavior of the PEtALS domain, if it can expand or shrink, or it can accept only one node. See Chapter 7, *PEtALS domain* for further details.

name attribute: The name of the domain. It is the identifier of a PEtALS domain.

`multicast` — Optional element. Defines the use of multicast IP messages within the PEtALS domain. If the company network allows multicast communication, the definition of this element is recommanded. See section 6 for further details.

- `ip` — the ip to use to send and receive multicast messages.
- `port` — the port to use to send and receive multicast messages.

`jndi` — Optional element. Defines an external JNDI server to use for the registry. All the nodes belonging to the domain access to this JNDI resource. See Chapter 7, *PEtALS domain* for further details.

- `factory` — The JNDI factory class name.
- `host` — The host name or IP of the JNDI server.
- `port` — The port to connect to the JNDI server.

3.2.2. Sub Domain configuration

A Domain is composed of one ore several Sub-domains. This configuration part defines the sud-domain identifiers and resources it uses within the topology.

```
...
  <sub-domain name="PEtALS subdomain">
    <configuration>
        <network-timed-synchronized>true</network-timed-synchronized>
    </configuration>

    <container.../>
    <container.../>
  <sub-domain/>
...
```

`sub-domain` *name* attribute: The name of the sub-domain. See Chapter 7, *PEtALS domain* for further details.

`configuration` The configuration of the PEtALS sub-domain.

`network-timed-synchronized` Set this parameter to true when hosts running the PEtALS containers of your sub-domain are network timed-synchronized, using NTP for example. In this case, an optimization is made on timeout management: if the mesage is received by the remote PEtALS instance after the timeout delay, the message is discarded by JORAM service and will be not processed by the remote PEtALS instance.

3.2.3. Container configuration

A `Sub-domain` is composed of one or several PEtALS servers or `Containers`. This configuration part defines the container identifiers and resources it uses within the topology.

Here is an example of a container configuration:

```
...
    <container name="0">
        <host>192.168.1.148</host>
        <description>description of the features of Container1</description>
        <configuration>

            <user>petalsUser</user>
            <password>petalsPassword</password>

            <network-service>
                <port>7720</port>
            </network-service>

            <jmx-service>
                <rmi-port>3000</rmi-port>
            </jmx-service>

            <joram-service>
                <domain-port>7740</domain-port>
                <tcp-port>7760</tcp-port>
            </joram-service>

            <dream-service>
                <tcp-port>7800</tcp-port>
                <ssl-port>7801</ssl-port>
            </dream-service>

        </tns:configuration>
    </container>
...
```

`container` *name* attribute: The name of the container. Used as identifier for transport communications.

Caution

Must be unique in the whole Domain. Must be an integer.

`host` The host name or ip address of the container.

`description` Optional. The description of the container.

`user` The login name, used for JMX or JORAM authentication

`password` The password, used for JMX or JORAM authentication

`network-service` The network service, used to discover new / started / stopped / removed PEtALS node.

- `port` The binding port. Used to receive dedicated messages from the domain.

`jmx-service` The JMX service, used to manage MBean objects.

- `rmiport` The port used to by the JMX RMI connector. The JMX Server would be reachable on this port at the URI `service:jmx:rmi:///jndi/rmi://host:rmiport/jmxRmiConnector`.

`joram-service` The JORAM service. Used by the PEtALS Transport layer to exchange JBI messages.

- `domain-port` The port used by the JORAM agent to communication with the JORAM domain.
- `tcp-port` The port used to able remote client to connect to the JORAM agent. Specify this port only in *dynamic* mode.

`dream-service` The DREAM sercvice. Used by the PEtALS Transport layer to exchange JBI messages.

- `tcp-port` The port used by DREAM to exchange message via TCP communication.
- `ssl-port` The port used by DREAM to exchange message via SSL communication.

3.3. Server properties - server.properties

`petals.container.name` The name of the container. Must match a container name defined in the topology configuration.

`petals.repository.path` Optional. The URL path to the PEtALS repository. PEtALS holds its JBI configuration in this repository and can recover its configuration from it.

`petals.exchange.validation` Optional. This property is used to activate the control of exchange acceptance by target component when the NMR routes messages (see `isExchangeWithConsumerOkay()` and `isExchangeWithProviderOkay()` methods in JBI Component interface for further details).

`petals.task.timeout` Optional. Maximum duration of the processing of a life-cycle operation on a JBI components and SAs (start, stop, ...). After this duration, if the processing of the operation is not finished, it is interrupted. It prevents from hanging threads.

`petals.classloaders.isolated` Optional. Activate/Unactivate the isolation of the ClassLoaders created for Shared Libraries and Components from the PEtALS container one. It can be useful to avoid concurrent libraries loading issues.

`petals.autoloader` Optional. Activate/Unactivate the autoloader service.It can be useful in production environment to unactivate it.

`petals.transport.qos`	This property set up the quality of service used by the transporter. Three QOS policy are settable: *reliable*, *fast* or *secure*. The *reliable* policy assures the delivery of the messageExchanges when attempting exchange transfers. If a target component is unreachable by any reasson (crash, network failure...),the messageExchanges are persisted and would be delivered at the component recovery. No effort is provided on the duration of the transfers. The *fast* policy assures an optimized transfer of message Exchange. According to the location of the destination, a component on a local or on a remote PEtALS container, the fastest transport protocol is used. The *secure* policy assures an encrypted transfer of message Exchange, by using an SSL communication.
`petals.routing.strategy`	This property defines the routing strategy. Two kind of strategy can be define: *highest* or *random*. The following parameters, separated by commas, represent respectively the weighting for local endpoint, the weighting for a remote active endpoint and the weighting for a remote inactive endpoint. The *random* strategy chooses an endpoint randomly in function of the defined weightings. Every endpoint has a chance to be elected, but the more the weight is strong, the more the endpoint can be elected. The *highest* strategy chooses an endpoint randomly amongst the endpoints having the strongest weigth.
`petals.ssl.key.password`	This property must be set if you want to use SSL communication. It defines the key password to retrieve the private key.
`petals.ssl.keystore.file`	This property must be set if you want to use SSL communication. It defines the keystore file where the keys are stored.
`petals.ssl.keystore.password`	This property must be set if you want to use SSL communication. It defines the keystore password.
`petals.ssl.truststore.file`	This property must be set if you want to use SSL communication. It defines the truststore file where the signatures are verified.
`petals.ssl.truststore.password`	This property must be set if you want to use SSL communication. It defines the truststore password.
`dream.session.cache.size`	This property set up the size of the DREAM session cache. A DREAM session is used to send a message to another PEtALS node. It is recommended to set the size to the number of others PEtALS nodes contained in your domain plus 1.
`dream.connection.timeout`	This property set up the DREAM timeout in ms to attempt to establish/reestablish a TCP/SSL connection.
`dream.connection.retry`	This property set up the DREAM number of retries to attempt to establish/ reestablish a TCP/SSL connection.

3.4. Loggers configuration - loggers.properties

PEtALS is relying on FRACTAL component framework which uses Monolog.

In this file, we configure monolog to wrap the JavaLog loggers.

For each PEtALS Fractal component loggers, 2 handlers are used:

- a dedicated console handler to provide logs in the standard output of your console.
- a shared file handler to provide logs in a common file stored in the `logs` directory.

The global default log level is set to `INFO`.

The shared file handler is filtering `DEBUG` logs, whereas the console handlers are not.

If you want to see the DEBUG logs on your console:

- you can activate the DEBUG level for PEtALS by changing the PEtALS global log level

```
logger.Petals.level DEBUG
```

- you can activate the DEBUG level for some PEtALS components by changing their log level (uncomment pertinent lines in the file)

```
#logger.Petals.JBI-Management.SystemRecoveryServiceImpl.level DEBUG
logger.Petals.JBI-Management.InstallationServiceImpl.level DEBUG
logger.Petals.JBI-Management.DeploymentServiceImpl.level DEBUG
#logger.Petals.JBI-Management.AutoLoaderServiceImpl.level DEBUG
```

At the end of the file, the configuration for the DREAM and JORAM transporters is defined.

As for PEtALS Fractal component, 2 handlers are defines for both DREAM and JORAM.

For these loggers, the log level is set to `ERROR`.

You can activate the `DEBUG` level alike the PEtALS loggers.

3.5. Router configuration - router.cfg

The PEtALS router is used to elect potential endpoints matching a service invocation.

The router architecture has been structured in modules. Modules to used are defined in the `router.cfg` file like this :

```
# Define the modules to be loaded in the router. They are processed in the following defined order
# Lines starting with '#' characters are comments ones
#org.ow2.petals.jbi.messaging.routing.module.AuthorizationModule
org.ow2.petals.jbi.messaging.routing.module.AddressResolverModule
#org.ow2.petals.jbi.messaging.routing.module.TraceRouterModule
org.ow2.petals.jbi.messaging.routing.module.RouterSenderModule
```

Each line is a module which will be loaded at router initialization. When a message need to be routed, it goes through all the router modules in the order they are defined in the configuration file.

Chapter 4. Management

4.1. Install/Uninstall components

The installation and uninstallation of components is fully described in the JBI specification at the sections 6.4 and 6.5.

4.1.1. Install/Uninstall with repertories

PEtALS embeds an autoloader service which takes care of all the default installation steps until the start of the component.

Components can be installed directly by copying the zip archive in the `install` directory of PEtALS. PEtALS will detect the new file and proceed to the installation and the start of the component.

To uninstall a component, remove the zip archive of the `installed` directory of PEtALS. PEtALS will detect the removal and proceed to the stop/shutdown/uninstallation of the component.

4.1.2. Install/Uninstall with the web administration console

The PEtALS web application proposes an administration console to install/uninstall your components. To use it, you need a servlet container like Tomcat.

Check the web console documentation for further details.

4.2. Install/Uninstall SLs

The installation and uninstallation of SLs is fully described in the JBI specification at the sections 6.4.

in the same manner of for the component, you can use repertories or the web console to do it.

4.3. Deploy/Undeploy SAs

The deployment and undeployment of SAs is fully described in the JBI specification at the sections 6.6.

in the same manner of for the component, you can use repertories or the web console to do it.

4.4. Monitoring PEtALS

The PEtALS web application proposes a monitoring console. To use it, you need a servlet container like Tomcat.

With the monitoring console, you can :

- visualize PEtALS nodes information of the domain
- control and set monitoring parameters for each PEtALS node
- visualize informations and statistics about the exchanges in the domain
- visualize your PEtALS container information
- control and set the monitoring parameters
- visualize informations and statistics about the exchanges

check the web console documentation for more information.

4.5. Advanced Management

JBI has designed the management if JBI engine on the JMX technology. Fur further details on JMX, please refers to official documentation.

PEtALS provides its management MBeans under the `PETALS` JMX domain.

There is 6 'top level' JBI management services :

1. **Administration Service**

   This service allows to retrieve information about JMX ObjectName, life-cycle state... on loaded JBI artifacts. For further information, please check the JavaDoc of the `AdminServiceMBean` interface.

2. **Deployment Service**

   This service allows to deploy/undeploy SAs and to manage their life-cycle (start, stop, shutdown). It allows to list SUs hold by a SA or SA deployed in a given component. For further information, please check the JavaDoc of the `DeploymentServiceMBean` interface. .

3. **Installation Service**

   This service allows to install/uninstall SLs and components and to manage (create, register, retrieve, destroy) `InstallerMBean` instances. Installers are used to perform the installation/uninstallation tasks on components. For more information check the javaDoc of the `InstallationServiceMBean` interface.

4. **Logger Service**

   This service allows to manage the loggers and their log handlers.

5. **Monitoring Service**

   This service allows to activate or deactivate the monitoring mode in PEtALS.

6. **PEtALS Administration Service**

   This service allows to execute Petals specific administration tasks like shutting down the PEtALS container or retrieve the domain topology.

4.5.1. JMX Management

An RMI JMX connector is launched within PEtALS container. The MBean methods can be accessed by RMI client, such as the JDK JConsole tool. The JConsole is commonly located in the `$JAVA_HOME/bin` directory of your environment.

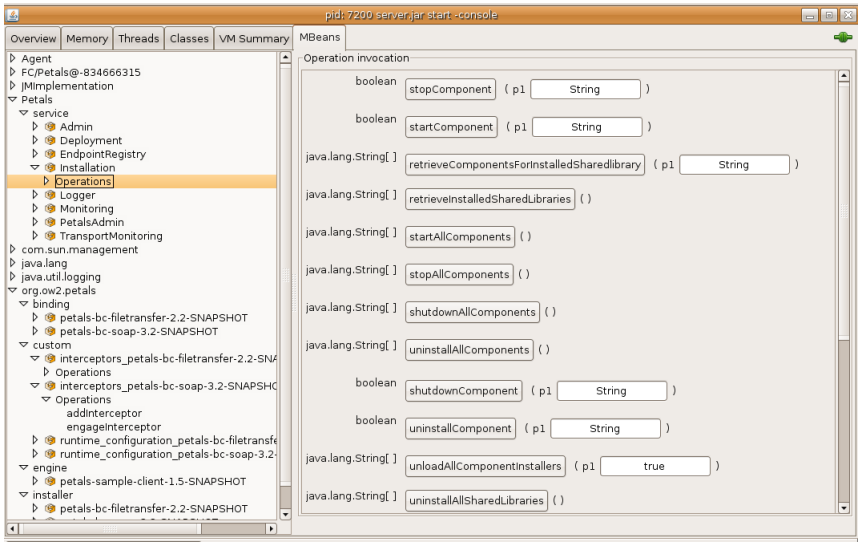
At startup, JConsole prompts for a server to connect to. Go to the `Advanced` tab and connect to the URI `service:jmx:rmi://jndi/rmi://myhost:myport/jmxRmiConnector`.

Note

The URI pattern is always the same, but `myhost` and `myhost` must be the ones defined in your topology file.

Once connected, you can find the MBeans exposed on the JMX server, including the PEtALS management services under `Petals` domain :

Figure 4.1. The JConsole management view



4.5.2. Ant Management

JBI specifies a set of dedicated ant task to be able to manage JBI container with batch/script programs. The full list of Ant tasks and their relatives parameters is described at section 6.10 of the JBI specification.

Moreover you can find a sample of an Ant build file in `$PETALS_HOME/ant` that illustrate the possibilities of management.

4.5.2.1. Install a component

To install a component just run this ant task :

```
<jbi-install-component file="file:///mydirectory/myfile.zip" port="myport" host="myhost"
 user="myuser" password="mypassworld" />
```

This will load a new `InstallerMBean` and execute the `install` method on this one. Then the component is completly installed.

4.5.2.2. Start a component

To start a component just run this ant task :

```
<jbi-start-component name="mycomponentname" port="myport" host="myhost" user="myuser"
 password="mypassworld" />
```

4.5.3. Console mode Management

The user can interact with PEtALS on a command-line console by starting PEtALS with the argument `-console` (`-c`).

Once PETALS is started, commands can be wrote to the console:

- hotdeploy -ZIP filePath- (or hd): install and start a component, a service assembly or a shared library
- hotundeploy -ZIP fileName- (or hu): shutdown and uninstall a component, a service assembly or a shared library
- path (or p) : display current file system path
- components (or c): display the installed components
- info (or i): display the container version

- setpath -Path- (or sp): change current file system path
- jndi (or d) : display the JNDI directory
- help (or h) : display this help
- stop (or q) : stop Petals
- shutdown (or x) : shut down Petals

Chapter 5. Optimizations

5.1. JBI Exchange Optimizations

5.1.1. Optimization setup

The following optimization can be exploited by JBI components by setting the related MessageExchange property to the MessageExchange transfered.

5.1.2. Acknowledgement bypassing

Each JBI MessageExchange pattern ends with a "DONE" or "ERROR" message. The sender has to accept such messages, otherwise they are accumulated in the NMR (**N**ormalized **M**essage **R**outer). This generate extra trafic, too, even in an In-Only exchange.

Setting the `org.ow2.petals.messaging.noack` property to `true` on the MessageExchange indicates that the `DONE` or `ERROR` message will not be sent through the NMR.

This feature is unactivated when using synchronous sends.

5.1.3. Source content compression

When large and redondant information is contained in the payload of a MessageExchange, it can be interestting to compress the payload to reduce the data volume transfered between two PEtALS nodes.

Setting the `org.ow2.petals.transport.compress` property to `true` on the MessageExchange indicates that the PEtALS transport layer will compress the Source content of the exchange messages.

Chapter 6. Security

Since the version 2.2, PEtALS is providing security features, based on JAAS. Check http://java.sun.com/javase/technologies/security for further details on JAAS.

6.1. Configuration

6.1.1. Login Module

PEtALS uses the Login Module mechanism to handle authentification. The default login modules are provided by the petals-jaas library. The login module can be loaded by specifiing a configuration file to be used at the container startup. The `login.properties` file define the options for the provided PropertiesLoginModule :

```
petals-domain {
    org.ow2.petals.jaas.PropertiesLoginModule
        sufficient
        org.ow2.petals.security.properties.user="users-passwords.properties"
        org.ow2.petals.security.properties.group="groups.properties";
};
```

- `org.ow2.petals.jaas.PropertiesLoginModule` is the class of the JAAS module to be used. This class is provided by the petals-jaas library and needs additional parameters for its configuration which are explained in the next lines.
- `org.ow2.petals.security.properties.user` value is the file where the users and passwords are defined. In the previous example, the file is `user-passwords.properties`. This file must be located at the same level than the `login.properties` one in the conf folder of your PEtALS distribution. Its content is defined like :

  ```
  petals=petals
  chamerling=password
  ```

  Left argument is the user login, right one is the password of the user.
- `org.ow2.petals.security.properties.group` value is the file where the groups are defined. In the previous example, the file is `groups.properties`. This file must be located at the same level than the `login.properties` one in the `conf` folder of your PEtALS distribution. Its content is defined like :

  ```
  #role=users as CSV
  admin=petals
  users=petals,chamerling
  ```

  Left argument is the group/role, right one are the users which are in the group as CSV.

6.1.2. Define Service - Endpoint authorizations

Once the login module has been configured, you can define the rules to access the services and endpoints in the `authorization.cfg` file under the `conf` folder of your PEtALS distribution.

```
# Authorization configuration, lines starting with '#' are ignored
#service={http://petals.ow2.org/
helloworld}HelloworldService;endpoint=HelloworldEndpoint;operation=sayHello;roles=*
service={http://petals.ow2.org/
helloworld}HelloworldService;endpoint=HelloworldEndpoint;operation=*;roles=users
```

An entry (line) is defined like this :

- service : The service qualified name
- endpoint : The endpoint name
- operation : The operation name

- roles : The roles which can access to the previous defined values

The operation and roles entries can have wildcard values which means 'all'.

With the previous sample, it is defined that users from the role 'users' can access to any operation of HelloworldService service / HelloworldEndpoint endpoint.

This authorizations are handled at the routing level. In order to be used, you must configure the router and add the AuthorizationModule router module to the router configuration file. More details are available on the router configuration section.

When the authorization router module process the JBI message, it will check if the message security subject is conform to the rules defined in the authorization configuration file. If not, an error will be returned to the service consumer.

6.2. Startup

You must start PEtALS with the `java.security.auth.login.config` system parameter to define the root path of the security module configuration files.

```
./startup.sh -Djava.security.auth.login.config=<PETALS_HOME>/conf/login.properties
```

Chapter 7. PEtALS domain

As you have seen , the PEtALS distributed environment is called a PEtALS *domain*.Three different domain mode are available to the PEtALS users.

The *standalone* mode restraints the definition of a single node in the PEtALS domain. Thus the container can be an enlighten version with some services bypassed, like the Network service and its discovery feature, the distributed PetALS JNDI resource, or some Transporters. Please use the *Standalone PEtALS distribution* for this mode.

The *dynamic* mode provides a PEtALS domain easily extendable and shrinkable. It is adapted to a development phase when nodes can be often added or removed from the domain. The weakness of this mode is that there is no control of the topology configuration.

The *static* mode provides a PEtALS delimited domain. it is adapted to an exploitation environment where the network resources (ports, IPs...) must be controlled, and the containers participating to the domain are clearly identified.

7.1. Distributed environment

7.1.1. A distributed JNDI directory

PEtALS provides its own implementation of a JNDI directory, distributed amongst its domain nodes. Nevertheless, it is still possible to configure the PEtALS domain to use an external JNDI directory. To do so the host and port of the external JNDI directory must be specified in the topology configuration. With this configuration, all the nodes are connected to the same JNDI end point.

With the PEtALS JNDI directory, each PEtALS node contains a local replication of the part of the JNDI directory it is using. When a modification occurs on this part, it is synchronized amongst the replicated JNDI directories.

The PEtALS node have a local reference to the distributed directory, to avoid the socket based access of a standard JNDI directory. It fasten the JNDI communication and bring better performance during the JBI exchanges.

It is possible to 'externalized' the JNDI directory if clients want to access to it; the PetALS user must specify the container name and the port on which the JNDI server would communicate with the JNDI clients. The clients use the standard JNDI API to connect and dialog with the PEtALS JNDI directory.

7.1.2. JBI Service-Endpoints access

A ServiceEndpoint registered by a JBI component is stored in the JNDI directory. Thus, all other JBI components (local to the server or hosted on another server) can see and use it.

The ServiceEndpoint is holden in the JNDI directory until it is unregistered by its JBI component owner. If the container is stopped, the distributed JNDI directory still contains the ServiceEndpoint reference.

When this PETALS container is started again, the JBI component is restarted and it tries to register again this ServiceEndpoint. As this ServiceEndpoint is already referenced in the JNDI directory, this reference is returned to the component.

Note

The PEtALS JNDI directory is persisted on local files, so if all the PEtALS node are stopped, the ServiceEndpoint are still registered in the JNDI directory file. *This feature is not implemented yet*.

7.1.3. JBI MessageExchange delivery

The communication between the JBI components running on PEtALS node is transparent.

A JBI MessageExchange for a ServiceEndpoint is sent to the component that has registered it,the component can be collocated or not to the sender.

As a ServiceEndpoint is still active even if the server that hosts it is stopped, messages can be still sent to it.

If the *reliability* transport QOS has been selected, these messages are put in the JORAM persistence, and will be delivered when the server that hosts this ServiceEndpoint starts again. Even if all PETALS servers are stopped, as the pending messages are physically stored on the server that hosts the sender, the messages will be delivered at startup (PETALS uses the JORAM store and forward mechanism).

Otherwise, if the fast transport QOS has been selected, exceptions are raised up to the sender.

7.1.4. PEtALS network

The PEtALS nodes members of a PEtALS domain need to communicate each others to inform about their states or about node joining or leaving the domain. PEtALS proposes a service, the network service, to support these requirements.

The network service is based on *JGroups* technology, which permits to dispatch message amongst a group of network elements, detects new or failure elements, and many others functionalities.

In the PEtALS domain, there is one node elected as the coordinator, basically the oldest started. This coordinator is the actor which decides to accept or not a new starting node.

The PEtALS network behaves differently according to the domain mode used. In *static* mode, when a node is trying to join the domain, the coordinator verify its entire configuration against the topology configuration it hold. If a mismatch is found, it refuses the new node and notifies it with an explicit response. In *dynamic* mode, the coordinator verifies just if the new node doesn't try to use a ressource already used by another node (e.g a same port on a same host), and then accept it.

Note

The PEtALS JNDI directory relies on the network service to replicated each of its directories.

7.1.5. Stop, Restart and Shutdown a container

You can stop a PETALS container:

- by using the PetalsAdmin JMX service,
- by terminating the jvm process (by "Ctrl-C" or "kill" command),
- by using the command "stop" if the PEtALS instance has been started in the command-line mode,
- by using the script "stop".

At this point, the server is stopped but still referenced in the topology configuration (i.e. the other living nodes know that this server exists, even if it is stopped).

When you start again this server, it retrieves its configuration and notifies through the PETALS topology that it is started. It eventually sends or receives pending messages.

It is possible to 'shut down' a PEtALS server. It means that all the JBI SA, SL and components registered on the PEtALS container are undeployed and uninstalled, so all the ServiceEndpoints it did expose are unregistered from the JNDi directory. Moreover, in *dynamic* mode, the server is definitely removed from the PEtALS domain and no more registed in the shared JNDI directory. In *static* mode, you must change the configuration of other nodes to remove the entry associated to this server, after to have shutdowned it.

You can shutdown the server:

- by using the PetalsAdmin JMX service,
- by using the command "shutdown" if the PEtALS instance has been started in the command-line mode,

- by using the script "shutdown".

Chapter 8. Links

- The PEtALS Website : http://petals.ow2.org
- The JBI Specification : http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html