PEtALS-SE-EIP

This document explain how to install configure and use the petals-se-eip JBI component.

PEtALS Team
Adrien LOUIS <adrien.louis@ebmwebsourcing.com>
Roland NAUDIN <roland.naudin@ebmwebsourcing.com>
- July 2008 -
# Table of Contents

PEtALS-SE-EIP ........................................................................................................... 4
1. Features .................................................................................................................. 5
2. Component Configuration ....................................................................................... 6
3. Service Configuration .............................................................................................. 9
   3.1. Processing a pattern .......................................................................................... 9
       3.1.1. Aggregator Pattern .................................................................................. 9
       3.1.2. Scatter-gather Pattern ............................................................................. 9
       3.1.3. Router Pattern .......................................................................................... 10
       3.1.4. Dynamic Router Pattern ....................................................................... 10
       3.1.5. Dispatcher Pattern .................................................................................. 10
       3.1.6. Routing-slip Pattern .............................................................................. 11
       3.1.7. Wire-tap Pattern .................................................................................... 11
       3.1.8. Bridge Pattern ....................................................................................... 12
       3.1.9. Service Unit descriptor .......................................................................... 13
       3.1.10. Usage ..................................................................................................... 15
   3.2. Call services during the pattern process .......................................................... 15
       3.2.1. Service Unit descriptor ........................................................................... 15
       3.2.2. Usage ...................................................................................................... 15
List of Tables

2.1. Configuration of the component (CDK) ................................................................. 7
2.2. Configuration of the component (EIP) ................................................................. 8
3.1. Configuration of a Service Unit to provide a service (JBI) ...................................... 13
3.2. Configuration of a Service Unit to provide a service (CDK) .................................... 14
3.3. Configuration of a Service Unit to provide a service (EIP) ...................................... 14
3.4. Configuration of a Service Unit to consume a service (JBI) .................................... 15
This component implements the main Enterprise Integration Patterns, as described in [http://www.enterpriseintegrationpatterns.com](http://www.enterpriseintegrationpatterns.com).

It is based on the PEtALS CDK.
Chapter 1. Features

The provided integration patterns are:

• Aggregator
• Bridge
• Dispatcher
• Router
• DynamicRouter
• RoutingSlip
• ScatterGather
• WireTap

The EIP component can be easily extended to provide more patterns.
Chapter 2. Component Configuration

The component can be extended to provide more integration patterns.

To add a new pattern, provide a Java class implementing:

```java
package org.ow2.petals.se.eip.patterns;
import org.ow2.petals.component.framework.util.Exchange;
import org.ow2.petals.se.eip.ExchangeContext;
public interface Pattern {
    public void processPattern(Exchange exchange, ExchangeContext context);
    public void init();
}
```
or by extending the abstract class:

```java
package org.ow2.petals.se.eip.patterns;

... public abstract class AbstractPattern implements Pattern {
    ... public abstract void process(Exchange exchange, ExchangeContext context) throws MessagingException;
    protected abstract boolean validateMEP(URI mep);
    protected abstract String getPatternName();
    ...
}
```

Use the `ExchangeContext` to help you processing your orchestration:

```java
package org.ow2.petals.se.eip;
import java.util.List;
import java.util.logging.Logger;
import javax.jbi.messaging.MessagingException;
import javax.jbi.servicedesc.ServiceEndpoint;
import org.ow2.petals.component.framework.Constants.MEPConstants;
import org.ow2.petals.component.framework.api.configuration.ConfigurationExtensions;
import org.ow2.petals.component.framework.api.exception.PetalsCDKException;
import org.ow2.petals.component.framework.api.message.Exchange;
import org.ow2.petals.component.framework.jbidescriptor.generated.Consumes;
public interface ExchangeContext {
    public Logger getLogger();
    public List<Consumes> getSUConsumes(ServiceEndpoint endpoint);
    public boolean sendSync(final Exchange exchange) throws MessagingException;
    public void sendAsync(final Exchange exchange) throws MessagingException;
    public Exchange accept(Exchange exchange) throws InterruptedException, PetalsCDKException;
    public Exchange createConsumeExchange(Consumes consumes) throws MessagingException;
    public Exchange createConsumeExchange(Consumes consumes, MEPConstants mep) throws MessagingException;
    public ConfigurationExtensions getExtensions();
}
```

Extends the JBI.xml file of the component to reference your pattern:
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ...>
  <eip:aggregator>org.ow2.petals.se.eip.patterns.Aggregator</eip:aggregator>
  <eip:router>org.ow2.petals.se.eip.patterns.Router</eip:router>
  <eip:bridge>org.ow2.petals.se.eip.patterns.Bridge</eip:bridge>
  <eip:wire-tap>org.ow2.petals.se.eip.patterns.WireTap</eip:wire-tap>
  <eip:scatter-gather>org.ow2.petals.se.eip.patterns.ScatterGather</eip:scatter-gather>
  ...
</jbi:jbi>

### Table 2.1. Configuration of the component (CDK)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptor-pool-size</td>
<td>The size of the thread pool used to accept Message Exchange from the NMR.</td>
<td>5</td>
<td>Yes</td>
<td>Runtime</td>
</tr>
<tr>
<td></td>
<td>Once a message is accepted, its processing is delegated to the processor pool thread.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>processor-pool-size</td>
<td>The size of the thread pool used to process Message Exchanges.</td>
<td>10</td>
<td>Yes</td>
<td>Runtime</td>
</tr>
<tr>
<td></td>
<td>Once a message is accepted, its processing is delegated to one of the thread of this pool.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>performance-notifications</td>
<td>Enable the performance notifications in the component. The CDK proposes to a performance notification feature to the component implementor. If you enable this feature, you must use the related method accessible in the AbstractComponent class.</td>
<td>-</td>
<td>No</td>
<td>Runtime</td>
</tr>
<tr>
<td>performance-step</td>
<td>When the performance notification feature is enabled, it is possible to define a step on the notifications. When there is an heavy message traffic, it is recommanded to increase this step to avoid performance disturbance.</td>
<td>-</td>
<td>No</td>
<td>Runtime</td>
</tr>
</tbody>
</table>
| 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 | -       | No       | Installation |
| 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 or NOTHING_IGNORED | -       | 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.
Component Configuration

- **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 parameter 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.

**Table 2.2. Configuration of the component (EIP)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>your-pattern-name</td>
<td>java class implementing your pattern. The name of the pattern at runtime will be the one you give as parameter name</td>
<td>-</td>
<td>No</td>
</tr>
</tbody>
</table>
Chapter 3. Service Configuration

3.1. Processing a pattern

PROVIDE SERVICE : Process several services invocations and orchestrate the results

A Service Unit contains one and only one provides section, which describes the pattern that will be processed when a message is received.

The Service Unit contains also one or more consumes sections, which reference services to call during the pattern execution. The order of the consumes sections is important, as it is the one which is used by the pattern during its execution.

The number of consumes sections depends on the pattern implemented.

If the MEP InOut or InOptOut are supported by an implemented EIP, the component returns to the consumer an OUT response built according to the pattern feature.

If an invoked service returns a Fault or an Error status, the process ends and the Fault or Error is sent back to the consumer.

If an operation is specified in a consumes sections, this operation is used to invoke the bound service, otherwise the operation of the incoming message is relayed.

3.1.1. Aggregator Pattern

The EIP Component receives incoming messages and identifies the messages that are correlated to a SU deployed.

The correlation can be either retrieved from an XPath expression specified in the SU parameter aggregator-correlation or by a property from the incoming exchange with the name 'aggregator-correlation'.

Once a message received matches the XPath expression specified in the SU parameter aggregator-complete, the pattern collects information from each previously correlated message and sends a single, aggregated message to the service referenced in the consumes section.

The result of the invoked service is reported to the sender of the 'complete' message.

Caution

consumes sections cardinality is [1-1].

Caution

message exchange pattern of the incoming exchange is InOnly or RobustInOnly.

Caution

messages order is kept from the incoming sequence to the outgoing message.

The aggregated message looks like:

```xml
<aggregator-root-element>
  <incoming message 1.../>
  ...
  <incoming message N/>
</aggregator-root-element>
```

3.1.2. Scatter-gather Pattern

The EIP Component forwards the incoming IN message of the Exchange to all the services referenced in the consumes sections (these services has to be InOut). The pattern waits for all the responses from the services, and aggregates them.
The aggregation is returned to the original consumer, as the OUT message of its original Exchange.

If a services called har response with a Fault, the fault is reported to the original Exchange, and all the others responses are ignored. The fault is them returned to the original consumer.

Caution

`consumes` sections cardinality is [1-n].

Caution

message exchange pattern of the incoming exchange and of the consumed services is `InOut`.

The response looks like:

```xml
<aggregate-result>
  <response from the 1st service referenced in the Service Unit.../>
  <response from the 2nd service referenced in the Service Unit.../>
  <response from the 3rd service referenced in the Service Unit.../>
  ...
</aggregate-result>
```

### 3.1.3. Router Pattern

Also known as Content-Based Router pattern.

The EIP Component evaluates expressions on the incoming IN message of the Exchange. Conditions can be multiple and are defined into the SU parameter `router-condition`.

Conditions are evaluated against the message until a `true` result. Then the exchange is forwarded to the service referenced in the `consumes` section matching the position of the condition.

E.g., the second condition is evaluated, and results to be `true`, the exchange is forwarded to the service referenced in the second `consumes` section.

If none of the conditions are `true`, the exchange is forwarded to the service referenced in the last `consumes` section (default).

Some example of conditions:

- `sum(/items/item/value) > 100` : the sum of all the values of the 'item' elements is greater than 100
- `name(/*)='helloworldRequest'` : the name of the root element is 'helloworldRequest'

Caution

`consumes` sections cardinality is the number of conditions plus 1 (the last one is the default service).

Caution

The last `consumes` section is the default service to invoke if no condition has been fulfilled.

Caution

message exchange pattern of the incoming exchange and of the consumed services is `InOnly`, `RobustInOnly` or `InOut`.

### 3.1.4. Dynamic Router Pattern

Inspired from the EIP Dynamic Router pattern.
This pattern is routing the incoming IN message toward a matching service, as for the router pattern. The difference is on the source of evaluation. Instead of evaluating the incoming message directly, this pattern invokes a first service which returns the message to evaluate. Conditions are defined into the SU parameter `dynamicrouter-condition`.

The first `consumes` section is invoked to get the message to evaluate.

Conditions are evaluated against the message until a `true` result. Then the exchange is forwarded to the service referenced in the `consumes` section matching the position of the condition.

E.g., the second condition is evaluated, and results to be `true`, the exchange is forwarded to the service referenced in the second `consumes` section.

If none of the conditions are `true`, the exchange is forwarded to the service referenced in the last `consumes` section (default).

Some example of conditions:

- `sum(/items/item/value) > 100`: the sum of all the values of the 'item' elements is greater than 100
- `name(/)="helloworldRequest"`: the name of the root element is 'helloworldRequest'

Caution

`consumes` sections cardinality is the number of conditions plus 2.

Caution

The first `consumes` section is the service to invoke to get the message to evaluate. Its pattern is `InOut`.

Caution

The last `consumes` section is the default service to invoke if no condition has been fulfilled.

Caution

message exchange pattern of the incoming exchange and of the consumed services is `InOnly`, `RobustInOnly` or `InOut`.

### 3.1.5. Dispatcher Pattern

The EIP Component dispatches the incoming IN message toward the configured service in `consumes` sections. No response message is returned.

Caution

`consumes` sections cardinality is `[1-n].`

Caution

message exchange pattern of the incoming exchange and the consumed services is `InOnly`.

### 3.1.6. Routing-slip Pattern

The EIP Component chains invocation of the referenced services in the `consumes` sections, in the order that they are declared.

The IN message of the incoming exchange is sent to the first service; the OUT response of this service is sent to the second service as an IN message, and so on.
The incoming exchange can be RobustInOnly or InOut.

If a fault is returned by an invoked service, the fault is reported to the original exchange and the process is terminated.

Otherwise, the result of the last service is returned to the original exchange.

**Caution**

consumes sections cardinality is [1-n].

**Caution**

message exchange pattern of the last service is the one of the incoming exchange, and can be RobustInOnly or InOut. All other services are InOut.

### 3.1.7. Wire-tap Pattern

The EIP Component copy the IN or OUT/Fault message of the exchange between the consumer and the provider of the functional service to a 'monitoring' service.

The SU parameter wiretap-way determines which way of the invocation is relayed to the 'monitoring' service. At each way correspond a message of the exchange to copy.

Values are :

- **request** (copy the IN message)
- **response** (copy the OUT / Fault message)
- **request-response** (copy IN and OUT/Fault message)
- **request-on-response** (copy IN after OUT is received; not copied if Fault or Error)

The copied message is sent to the 'monitoring' service as an IN message using the InOnly exchange pattern.

The first consumes section references the provider, the second one references the 'monitoring' service.

**Caution**

consumes sections cardinality is 2.

**Caution**

message exchange pattern of the 'monitoring' service is InOnly. Message exchange pattern of the provider is InOnly, RobustInOnly or InOut.

### 3.1.8. Bridge Pattern

The EIP Component acts as an exchange pattern bridge, and allows you, for instance, to transform an InOnly invocation pattern into an InOut one, to be able to invoke service in a best effort way.

Define in a consumes section the service you want to call, and the EIP component will match the incoming and outgoing exchange pattern the best possible.

**Caution**

consumes sections cardinality is 1.
Caution

OUT response is lost if the incoming exchange is InOnly or RobustInOnly.

3.1.9. Service Unit descriptor

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi version="1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jbi="http://java.sun.com/xml/ns/jbi"
  xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
  xmlns:eip="http://petals.ow2.org/components/eip/version-2.2"
  xmlns:generatedNs="http://petals.ow2.org/EIP/dynamic-router">
  <jbi:services binding-component="false">
    <jbi:provides
      interface-name="generatedNs:dynamic-router1"
      service-name="generatedNs:dynamic-router1Service"
      endpoint-name="dynamic-router1Endpoint">
      <petalsCDK:wsdl xsi:nil="true" />
      <eip:eip>dynamic-router</eip:eip>
      <eip:dynamic-router-condition>sum(/items/item/value) > 100</eip:dynamic-router-condition>
      <eip:dynamic-router-condition>name(/*)='coucou'</eip:dynamic-router-condition>
    </jbi:provides>

    <jbi:consumes interface-name="generatedNs:dynamic-routerExpression"
      service-name="generatedNs:dynamic-routerExpressionService"
      endpoint-name="routerExpressionEndpoint">
      <petalsCDK:mep xsi:nil="true" />
    </jbi:consumes>

    <jbi:consumes interface-name="generatedNs:dynamic-routerProvider1"
      service-name="generatedNs:dynamic-routerProvider1Service"
      endpoint-name="routerProvider1Endpoint">
      <petalsCDK:mep xsi:nil="true" />
    </jbi:consumes>

    <jbi:consumes interface-name="generatedNs:dynamic-routerProvider2"
      service-name="generatedNs:dynamic-routerProvider2Service"
      endpoint-name="dynamic-routerProvider2Endpoint">
      <petalsCDK:mep xsi:nil="true" />
    </jbi:consumes>

    <jbi:consumes interface-name="generatedNs:dynamic-routerProvider3"
      service-name="generatedNs:dynamic-routerProvider3Service"
      endpoint-name="dynamic-routerProvider3Endpoint">
      <petalsCDK:mep xsi:nil="true" />
    </jbi:consumes>
  </jbi:services>
</jbi:jbi>
```

Table 3.1. Configuration of a Service Unit to provide a service (JBI)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>provides</td>
<td>Describe the JBI service that will be exposed into the JBI bus. <em>Interface (qname), service (qname) and endpoint (string) attributes are required.</em></td>
<td>-</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Table 3.2. Configuration of a Service Unit to provide a service (CDK)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsdl</td>
<td>Path to the WSDL document describing services and operations exposed by the provided JBI endpoints defined in the SU. The value of this parameter is: • an URL • a file relative to the PEtALS installation path If no wsdl path is specified, a basic description is automatically provided by the CDK.</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout in milliseconds of a synchronous send. This parameter can be used in conjunction with the <code>sendSync(Exchange exchange)</code> method of the Listeners. Set 0 for an infinite timeout.</td>
<td>-</td>
<td>No</td>
</tr>
</tbody>
</table>

### Table 3.3. Configuration of a Service Unit to provide a service (EIP)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required by pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>eip</td>
<td>The name of the pattern to execute. Pattern provided are: aggregator, router, dynamic-router, dispatcher, routing-slip, wire-tap, bridge, scatter-gather. If you provide other patterns, set the name of your pattern to use.</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>router-condition</td>
<td>XPath condition applied on the incoming message.</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>dynamic-router-condition</td>
<td>XPath condition applied on the message returned by the first service.</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>wiretap-way</td>
<td>Exchange way on which the message should be copied and sent to the monitoring service. Values are request (copy IN), response (copy OUT/Fault), request-response (copy IN and OUT/Fault), request-on-response (copy IN after OUT is received; not copied if Fault or Error)</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>aggregator-complete</td>
<td>XPath condition applied to complete the sequence and trigger the invocation of the targeted service of the pattern with the aggregate message.</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>aggregator-correlation</td>
<td>XPath condition that is applied on the incoming message to correlate them together.</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>aggregator-root-element</td>
<td>The value which will be used as root element name of the aggregate message.</td>
<td>aggregator</td>
<td>No</td>
</tr>
<tr>
<td>scattergather-root-element</td>
<td>The value which will be used as root element name of the aggregate message.</td>
<td>scatter-gather</td>
<td>No</td>
</tr>
</tbody>
</table>
3.1.10. Usage

When deploying a service unit like in the previous code snippet, the JBI messages received will be processed by the Pattern and some calls to the services described in the other *consumes* sections will be called, depending on the pattern.

3.2. Call services during the pattern process

**CONSUME SERVICE**: Call a JBI service

In the same Service Unit than the *provides* section is defined, you can set all the services that will take a part of the pattern processing. These services are referenced in *consumes* sections.

The order of the *consumes* is important.

3.2.1. Service Unit descriptor

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi version="1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jbi="http://java.sun.com/xml/ns/jbi"
    xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
    xmlns:eip="http://petals.ow2.org/components/eip/version-2.2"
    xmlns:generatedNs="http://petals.ow2.org/EIP/bridge">
    <jbi:services binding-component="false">
        <jbi:provides
            interface-name="generatedNs:bridge1"
            service-name="generatedNs:bridge1Service"
            endpoint-name="bridge1Endpoint">
            <petalsCDK:wsdl xsi:nil="true"/>
            <eip:eip>bridge</eip:eip>
        </jbi:provides>
        <jbi:consumes
            interface-name="generatedNs:bridgeProvider"
            service-name="generatedNs:bridgeProviderService"
            endpoint-name="bridgeProviderEndpoint">
            <petalsCDK:mep">InOut</petalsCDK:mep>
            <eip:mep>bridge</eip:mep>
        </jbi:consumes>
    </jbi:services>
</jbi:jbi>
```

Table 3.4. Configuration of a Service Unit to consume a service (JBI)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumes</td>
<td>Name of the JBI service to invoke into the JBI bus. You can define only the interface (qname) to let the NMR choose a matching service, or the pair service(qname) and endpoint (string) to consume the localized service.</td>
<td>-</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note**

To be able to download WSDL imports in a long futur, these imports are cached by the CDK when installing services. This feature can be disabled with `disableWsdlImportsDownload`.

3.2.2. Usage

Each *consumes* section defined in the descriptor will take a part of the process, according to the pattern in used.