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>

- December 2007 -

Table of Contents

List of Tables

PEtALS-SE-EIP

This component implements the main Enterprise Integration Patterns, as described in http://www.enterpriseintegrationpatterns.com.

Chapter 1. Features

The integration patterns provided are:

- Aggregator
- Bridge
- Dispatcher
- Router
- RoutingSlip
- 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 extending :

`org.ow2.petals.se.eip.patterns.Pattern`:

```
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);

}
```

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

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

import java.net.URI;
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.exception.PEtALSCDKException;
import org.ow2.petals.component.framework.util.Exchange;
import org.ow2.petals.component.framework.util.Extensions;
import org.ow2.petals.jbi.descriptor.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 void send(final Exchange exchange) throws MessagingException;
  public Exchange createConsumeExchange(Consumes consumes, URI mep)throws MessagingException ;
  public Extensions getExtensions();
}
```

Extends the JBI.XML file to reference your pattern :

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
...
<petals:params>
  <petals:param name="aggregator">org.ow2.petals.se.eip.patterns.Aggregator</petals:param>
  <petals:param name="router">org.ow2.petals.se.eip.patterns.Router</petals:param>
  <petals:param name="dispatcher">org.ow2.petals.se.eip.patterns.Dispatcher</petals:param>
  <petals:param name="routing-slip">org.ow2.petals.se.eip.patterns.RoutingSlip</petals:param>
  <petals:param name="bridge">org.ow2.petals.se.eip.patterns.Bridge</petals:param>
  <petals:param name="wire-tap">org.ow2.petals.se.eip.patterns.WireTap</petals:param>
</petals:params>
...
</jbi:jbi>
```

Table 2.1. component installation configuration attributes

Attribute	Description	Default	Required
your-pattern-name	java class implementing your pattern. The name of the pattern will be the one you give as parameter name		No

Chapter 3. Service Configuration

3.1. Processing a pattern

PROVIDE SERVICE : do some services calls 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 CONSUMES sections count depends on the pattern.

The EIP Component returns to the Consumer the OUT response from the service it called, if the pattern is InOut or InOptOut.

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

3.1.1. aggregator 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 OUT responses from the services, and aggregate them.

The aggregate is returned to the original Consumer, as the OUT message of its original Exchange.

Caution

CONSUMES sections count is 1-n.

Caution

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

The response looks like :

```
<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>
```

3.1.2. router pattern

The EIP Component evaluates an expression on the incoming IN message of the Exchange. The condition is defined in the `condition` parameter of the Service Unit.

If the evaluation result is `true`, the exchange is forwarded to the first service referenced in the CONSUMES section. Otherwise, it is forwarded to the second one. The response is sent back if it exists.

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 count is 2.

3.1.3. dispatcher pattern

The EIP Component acts as for the aggregator pattern, except that it just forwards InOnly or RobustInOnly exchange. No response is returned.

Caution

CONSUMES sections count is 0-n.

Caution

message exchange pattern of the incoming exchange and the consumed services is InOnly or RobustInOnly

3.1.4. routing-slip pattern

The EIP Component chains all the services referenced in the CONSUMES sections, in the order 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 (Robust)InOnly.

Each service called must be In(Opt)Out service, except the last one that has to respect the message exchange pattern of the original incoming exchange.

If the original exchange is In(Opt)Out, the OUT response from the last service is sent back to the original Consumer.

Caution

CONSUMES sections count is 1-n.

Caution

message exchange pattern of the last service has to be the same than the incoming exchange. All other services have to be In(Opt)Out.

3.1.5. wire-tap pattern

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

the parameter way defines which message of the exchange has to be copied.

Values are :

- request (copy the IN message)
- response (copy the OUT / Fault message)

- request-response (copy IN and OUT/Fault message)

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 service on which the copy of the message is sent.

Caution

CONSUMES sections count is 2.

Caution

message exchange pattern of the 'monitoring' service is InOnly.

3.1.6. bridge pattern

The EIP Component acts as an exchange pattern bridge, and allows you, for instance, to transform an InOnly call into an InOut call, if your consumer can only sends InOnly messages and your provider can only accepts InOut exchanges.

Just defined in a CONSUMES section the service you want to call, and the EIP Component will match the incoming and outgoing exchange pattern the best than it can.

Caution

CONSUMES sections count is 1.

Caution

OUT responses are lost if the incoming exchange is In(Robust)Only

3.1.7. Service Unit descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:petals="http://petals.ow2.org/extensions"
    xmlns:jbi="http://java.sun.com/xml/ns/jbi"
    xmlns:company="http://company.com"
    version="1.0">
 <jbi:services binding-component="true">

  <jbi:provides interface-name="company:MyProcess"
   service-name="company:MyProcessImpl"
   endpoint-name="EIPMyProcess">
   <petals:params>
    <petals:param name="eip">router</petals:param>
    <petals:param name="condition">name(/*)='helloworldRequest'</petals:param>
   </petals:params>
  </jbi:provides>

  <jbi:consumes interface-name="petals:Helloworld"/>
  <jbi:consumes interface-name="petals:Clock"/>

 </jbi:services>
</jbi:jbi>
```

Table 3.1. service-unit attributes to provide services

Attribute	Description	Default	Required
provides	Name of the JBI service that will be activated to expose the pattern into the JBI environment. interface (qname), service (qname) and endpoint (string) name are required.		Yes

Table 3.2. eip attributes for processing pattern

Attribute	Description	Default Value	Required
eip	The name of the pattern to call. Default pattern provided are : aggregator, router, dispatcher, routing-slip, wire-tap, bridge. If you provide other pattern, it is the name of your pattern you want to use.		Yes
condition	XML condition applied on the incoming message.		Only for router
way	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)		Only for wire-tap

3.1.8. 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 have to set all the services that will take part into the pattern processing. These services are referenced in CONSUMES sections.

The order of the CONSUMES is important.

3.2.1. Service Unit descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:petals="http://petals.ow2.org/extensions"
    xmlns:jbi="http://java.sun.com/xml/ns/jbi"
    xmlns:company="http://company.com"
    version="1.0">
 <jbi:services binding-component="true">

  <jbi:provides interface-name="company:MyProcess"/>

  <jbi:consumes interface-name="petals:Helloworld"
   service-name="petals:HelloworldService"
   endpoint-name="HelloworldEndpoint">
   <petals:mep>in-out</petals:mep>
   <petals:operation>sayHello</petals:operation>
   <petals:params>
   </petals:params>
  </jbi:consumes>
 </jbi:services>
</jbi>
```

Table 3.3. service-unit attributes to provide services

Attribute	Description	Default	Required
consumes	Name of the JBI service that will be called into the JBI environment. Only the interface (Qname) name can be provided (the container will choose a ServiceEndpoint for this interface), or you can only set service (qname) and endpoint (string) names, without the interface name.		Yes

Table 3.4. extensions

Attribute	Description	Default Value	Required
mep	The MEP to use for the message exchange. Possible values are 'InOnly', 'InOptionalOut', 'InOut', 'RobustInOnly'	InOnly	Yes
timeout	The send timeout in ms	[empty string]	No
operation	Operation to invoke on the foreign JBI endpoint		Yes

3.2.2. Usage

Each CONSUMES section defined in the descriptor will take part of the process.

For instance, if you define a "router" service (set a PROVIDES section with routing-slip pattern), the service referenced in the first CONSUMES section will be called if the router test is true. Otherwise, the service referenced in the second CONSUMES section will be called.

Chapter 4. Samples