

JBossESB 4.2.1 GA

jBPM Integration Guide

JBESB-JBPMG-2/4/08



Legal Notices

The information contained in this documentation is subject to change without notice.

JBoss Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. JBoss Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Java™ and J2EE is a U.S. trademark of Sun Microsystems, Inc. Microsoft® and Windows NT® are registered trademarks of Microsoft Corporation. Oracle® is a registered U.S. trademark and Oracle9™, Oracle9 Server™ Oracle9 Enterprise Edition™ are trademarks of Oracle Corporation. Unix is used here as a generic term covering all versions of the UNIX® operating system. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Copyright

JBoss, Home of Professional Open Source Copyright 2006, JBoss Inc., and individual contributors as indicated by the @authors tag. All rights reserved.

See the copyright.txt in the distribution for a full listing of individual contributors. This copyrighted material is made available to anyone wishing to use, modify, copy, or redistribute it subject to the terms and conditions of the GNU General Public License, v. 2.0. This program is distributed in the hope that it will be useful, but WITHOUT A WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

See the GNU General Public License for more details. You should have received a copy of the GNU General Public License, v. 2.0 along with this distribution; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Software Version

JBossESB 4.2.1 GA

Restricted Rights Legend

Use, duplication, or disclosure is subject to restrictions as set forth in contract subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause 52.227-FAR14.

© Copyright 2008 JBoss Inc.

Contents

Table of Contents

Contents.....	iii	jBPM to JBossESB.....	15
		EsbNotifier.....	15
		EsbActionHandler.....	17
		Exception Handling jBPM -> JBossESB.....	18
		Scenario 1. Time-out.....	18
		Scenario 2. Exception Transition.....	19
		Scenario 3. Exception Decision.....	20
About This Guide.....	4	Service Orchestration.....	21
What This Guide Contains.....	4	Introduction.....	21
Audience.....	4	Orchestration Diagram.....	21
Prerequisites.....	4	Process Deployment and Instantiation.....	28
Organization.....	4	Conclusion.....	30
Documentation Conventions.....	4	Known Issues.....	32
Additional Documentation.....	5	References.....	33
Contacting Us.....	6	Index.....	34
jBPM Integration.....	7		
Introduction.....	7		
Integration Configuration.....	7		
jBPM configuration.....	9		
Creation and Deployment of a Process			
Definition.....	10		
JBossESB to jBPM.....	13		
Exception Handling JBossESB to jBPM	15		

About This Guide

What This Guide Contains

The jBPM Integration Guide document captures how jBPM integrates with JBossESB.

Audience

This guide is most relevant to engineers who are responsible for using JBossESB 4.2.1 GA installations and want to know how jBPM can be used in JBossESB.

Prerequisites

None.

Organization

This guide contains the following chapters:

- **Chapter 1, jBPM-Integration:** A description of how jBPM is integration into JBossESB, and how it can be used.
- **Chapter 2, Service Orchestration:** An overview on how jBPM can be used to orchestrate JBossESB Services.

Documentation Conventions

The following conventions are used in this guide:

Convention	Description
<i>Italic</i>	In paragraph text, italic identifies the titles of documents that are being referenced. When used in conjunction with the Code text described below, italics identify a variable that should be replaced by the user with an actual value.
Bold	Emphasizes items of particular importance.
Code	Text that represents programming code.
Function Function	A path to a function or dialog box within an interface. For example, “Select File Open.” indicates that you should select the Open function from the File menu.
() and	<p>Parentheses enclose optional items in command syntax. The vertical bar separates syntax items in a list of choices. For example, any of the following three items can be entered in this syntax:</p> <pre>persistPolicy (Never OnTimer OnUpdate NoMoreOftenThan)</pre>
Note:	A note highlights important supplemental information.
Caution:	A caution highlights procedures or information that is necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

Table 1 Formatting Conventions

Additional Documentation

In addition to this guide, the following guides are available in the JBossESB 4.2.1 GA documentation set:

1. **JBossESB 4.2.1 GA *Trailblazer Guide***: Provides guidance for using the trailblazer example.
2. **JBossESB 4.2.1 GA *Programmer's Guide***: Provides guidance for developing applications using JBossESB.
3. **JBossESB 4.2.1 GA *Getting Started Guide***: Provides a quick start reference to configuring and using the ESB.
4. **JBossESB 4.2.1 GA *Administration Guide***: How to manage JBossESB.
5. **JBossESB 4.2.1 GA *Release Notes***: Information on the differences between this release and previous releases.
6. **JBossESB 4.2.1 GA *Services Guides***: Various documents related to the services available with the ESB.

Contacting Us

Questions or comments about JBossESB 4.2.1 GA should be directed to our support team.

jBPM Integration

Introduction

JBoss jBPM is a powerful workflow and BPM (Business Process Management) engine. It enables the creation of business processes that coordinate between people, applications and services. With its modular architecture, JBoss jBPM combines easy development of workflow applications with a flexible and scalable process engine. The JBoss jBPM process designer graphically represents the business process steps to facilitate a strong link between the business analyst and the technical developer. This document assumes that you are familiar with jBPM. If you are not you should read the jBPM documentation [TB-JBPM-USER] first. JBossESB integrates the jBPM so that it can be used for two purposes:

1. **Service Orchestration:** ESB services can be orchestrated using jBPM. You can create a jBPM process definition which makes calls into ESB services.
2. **Human Task Management :** jBPM allows you to incorporate human task management integrated with machine based services.

Integration Configuration

The jbpms.esb deployment that ships with the ESB includes the full jBPM runtime and the jBPM console. The runtime and the console share a common jBPM database. The ESB DatabaseInitializer mbean creates this database on startup. The configuration for this mbean is found in the file jbpms.esb/jbpms-service.xml.

```
<classpath codebase="deploy" archives="jbpms.esb"/>
<classpath codebase="deploy/jbossesb.sar/lib" archives="jbossesb-rosetta.jar"/>
<mbean code=
  "org.jboss.internal.soa.esb.dependencies.DatabaseInitializer"
  name="jboss.esb:service=JBPMDatabaseInitializer">
  <attribute name="Datasource">java:/JbpmsDS</attribute>
  <attribute name="ExistsSql">
    select * from JBPM_ID_USER</attribute>
  <attribute name="SqlFiles">
    jbpms-sql/jbpms.jpdl.hsqldb.sql, jbpms-sql/import.sql
  </attribute>
  <depends>
    jboss.jca:service=DataSourceBinding, name=JbpmsDS
  </depends>
</mbean>
<mbean code=
  "org.jboss.soa.esb.services.jbpms.configuration.JbpmsService"
  name="jboss.esb:service=JbpmsService">
</mbean>
```

The first Mbean configuration element contains the configuration for the DatabaseInitializer. By default the attributes are configured as follows:

- “Datasource” - use a datasource called JbpmDS,
- “ExistsSql” - check if the database exists by running the sql: “Select * from JBPM_ID_USER”
- “SqlFiles” - if the database does not exist it will attempt to run the files jbpm.jpdl.hsqldb.sql and import.sql. These files reside in the jbpm.esb/jbpm-sql directory and can be modified if needed. Note that slightly different ddl files are provided for the various databases.

The DatabaseInitializer mbean is configured in jbpm-service.xml to wait for the JbpmDS to be deployed, before deploying itself. The second mbean “JbpmService” ties the lifecycle of the jBPM job executor to the jbpm.esb lifecycle - it starts a job executor instance on startup and stops it on shutdown. The JbpmDS datasource is defined in the jbpm-ds.xml and by default it uses a HSQL database. In production you will want change to a production strength database. All jbpm.esb deployments should share the same database instance so that the various ESB nodes have access to the same processes definitions and instances.

The jBPM console is a web application accessible at <http://localhost:8080/jbpm-console> when you start the server. The login screen is shown in Fig. 1.

Log In as an Example User		
Choose an example user from the list to log in as:		
User Name	Password	Group(s)
manager	manager	user manager admin
user	user	user
shipper	shipper	user
admin	admin	user admin

To remove this login name list, edit the web.xml file and locate the section entitled "Example Login page".

Figure 1. The jBPM Console

Please check the jBPM documentation [TB-JBPM-USER] to change the security settings for this application, which will involve change some settings in the conf/login-config.xml. The console can be used for deploying and monitoring jBPM processes, but is can also be used for human task management. For the different users a customized task list will be shown and they can work on these tasks. The quickstart bpm_orchestration4 [JBESB-QS] demonstrates this feature.

The jbpm.esb/META-INF directory contains the deployment.xml and the jboss-esb.xml. The deployment.xml specifies the resources this esb archive depends on:


```

<jbossesb-deployment>
  <depends>jboss.esb:deployment=jbossesb.esb</depends>
  <depends>jboss.jca:service=DataSourceBinding,name=JbpmDS</depends>
</jbossesb-deployment>

```

which are the jbossesb.esb and the JbpmDS datasource. This information is used to determine the deployment order.

The jboss-esb.xml deploys one internal service called “JBpmCallbackService”:

```

<services>
  <service category="JBossESB-Internal"
    name="JBpmCallbackService"
    description="Service which makes Callbacks into jBPM">
    <listeners>
      <jms-listener name="JMS-DCQListener"
        busidref="jBPMCallbackBus"
        maxThreads="1"
      />
    </listeners>
    <actions mep="OneWay">
      <action name="action" class="
        org.jboss.soa.esb.services.jbpm.actions.JBpmCallback"/>
    </actions>
  </service>
</services>

```

This service listens to the jBPMCallbackBus, which by default is a JMS Queue on either a JBossMQ (jbmq-queue-service.xml) or a JbossMessaging (jbm-queue-service.xml) messaging provider. Make sure only one of these files gets deployed in your jbpm.esb archive. If you want to use your own provider simple modify the provider section in the jboss-esb.xml to reference your JMS provider.

```

<providers>
  <!-- change the following element to jms-jca-provider to
    enable transactional context -->
  <jms-provider name="CallbackQueue-JMS-Provider"
    connection-factory="ConnectionFactory">
    <jms-bus busid="jBPMCallbackBus">
      <jms-message-filter
        dest-type="QUEUE"
        dest-name="queue/CallbackQueue"
      />
    </jms-bus>
  </jms-provider>
</providers>

```

For more details on what the JbpmCallbackService does, please see the “jBPM to ESB” section later on in this chapter.

jBPM configuration

The configuration of jBPM itself is managed by three files, the jbpm.cfg.xml and the hibernate.cfg.xml and the jbpm.mail.templates.xml.

By default the jbpm.cfg.xml is set to use the JTA transaction manager, as defined in the section:

```

<service name="persistence">
  <factory>
    <bean class="
      org.jbpm.persistence.jta.JtaDbPersistenceServiceFactory">

```

```

        <field name="isTransactionEnabled"><false/></field>
        <field name="isCurrentSessionEnabled"><true/></field>
        <!--field name="sessionFactoryJndiName">
            <string value="java:/myHibSessFactJndiName" />
        </field-->
    </bean>
</factory>
</service>

```

Other settings are left to the default jBPM settings.

The hibernate.cfg.xml is also slightly modified to use the JTA transaction manager

```

<!-- JTA transaction properties (begin) ===
===== JTA transaction properties (end) -->
<property name="hibernate.transaction.factory_class">
    org.hibernate.transaction.JTATransactionFactory</property>
<property name="hibernate.transaction.manager_lookup_class">
    org.hibernate.transaction.JBossTransactionManagerLookup</property>

```

Hibernate is not used to create the database schema, instead we use our own DatabaseInitiazer mbean, as mentioned in the previous section.

The jbpmmail.templates.xml is left empty by default. For each more details on each of these configuration files please see the jBPM documentation.

Note that the configuration files that usually ship with the jbpmm-console.war have been removed so that all configuration is centralized in the configuration files in the root of the jbpmm.esb archive.

Creation and Deployment of a Process Definition

To create a Process Definition we recommend using the eclipse based jBPM Process Designer Plugin [KA-JBPM-GPD]. You can either download and install it to eclipse yourself, or use JBoss Developer Studio. Figure 2 shows the graphical editor.

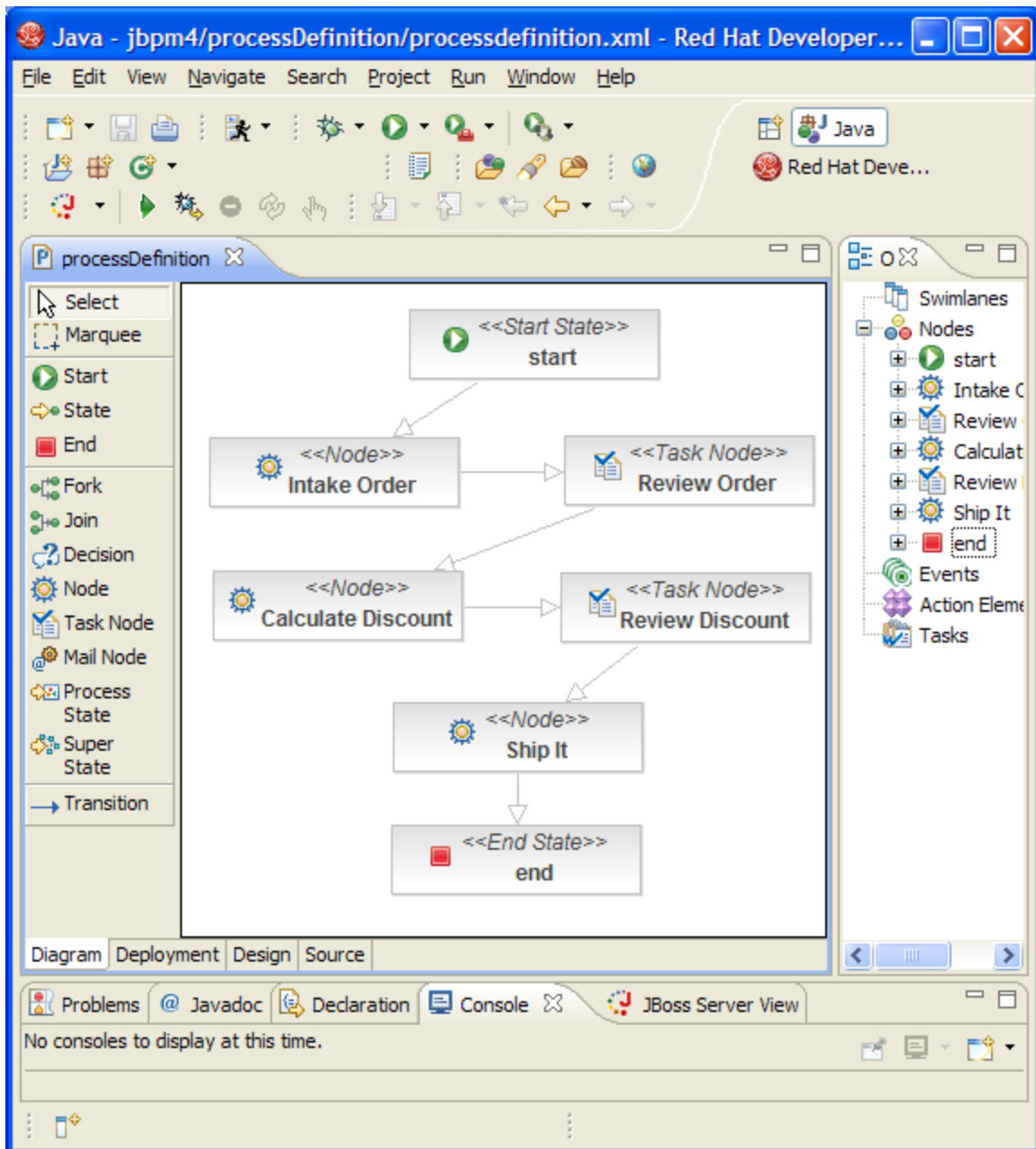


Figure 2. jBPM Graphical Editor

The graphical editor allows you to create a process definition visually. Nodes and transitions between nodes can be added, modified or removed. The process definition saves as an XML document which can be stored on a file system and deployed to a jBPM instance (database). Each time you deploy the process instance jBPM will version it and will keep the older copies. This allows processes that are in flight to complete using the process instance they were started on. New process instances will use the latest version of the process definition.

To deploy a process definition the server needs to be up and running. Only then can you go to the 'Deployment' tab in the graphical designer to deploy a process archive (par). Figure 3 shows the "Deployment" tab view.

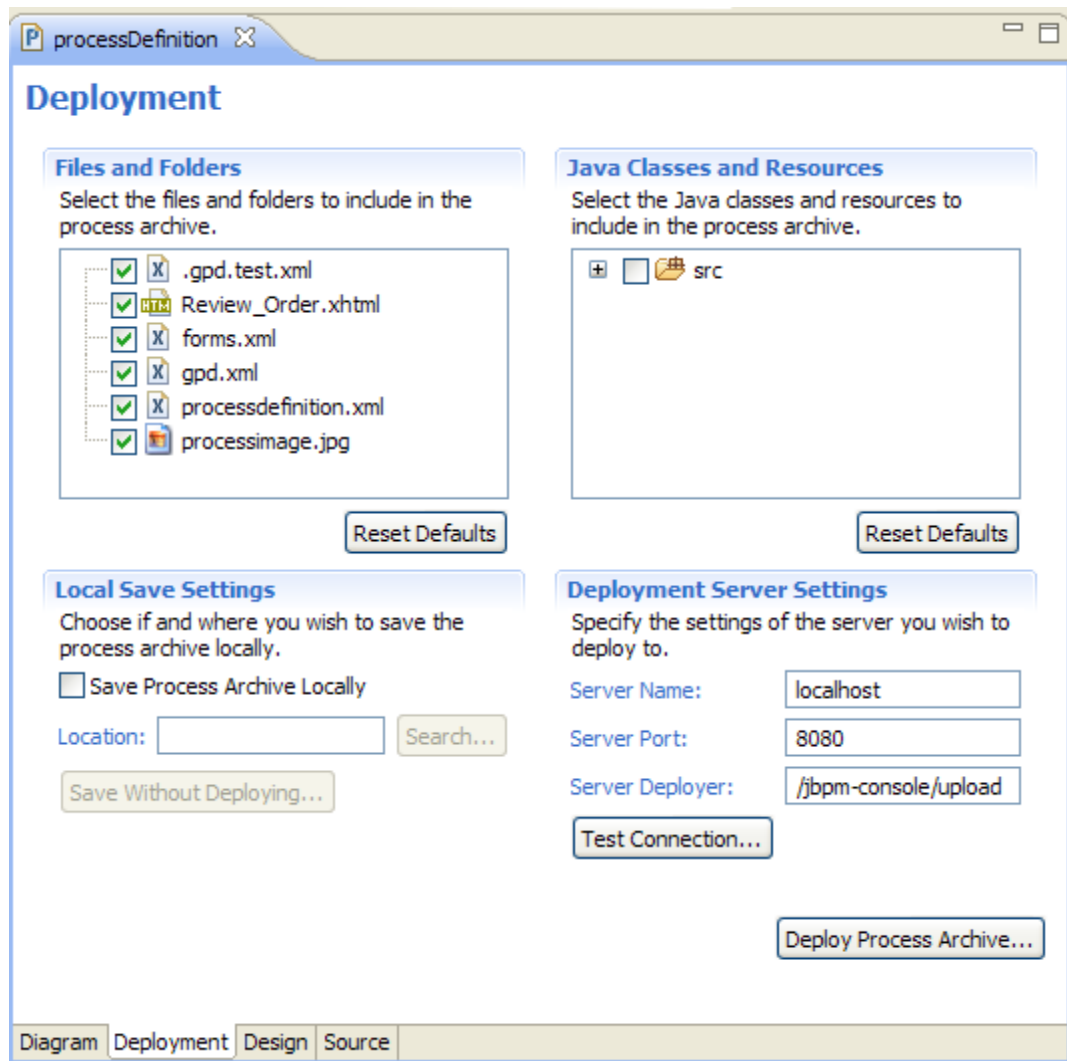


Figure 3. The Deployment View

In some cases it would suffice to deploy just the processdefinition.xml, but in most cases you will be deploying other type of artifacts as well, such as task forms. It is also possible to deploy Java classes in a jar, which means that they end up in the database where they will be stored and versioned. However it is strongly discouraged to do this in the ESB environment as you will risk running into class loading issues. Instead we recommend deploying your classes in the lib directory of the server. You can deploy a process definition

- straight from the eclipse plugin, by clicking on the “Test Connection..” button and, on success, by clicking on the “Deploy Process Archive” button,
- by saving the deployment to a jar file and using the jBPM console to deploy the archive, see Figure 4, or finally,
- by using the DeployProcessToServer jBPM ant task.

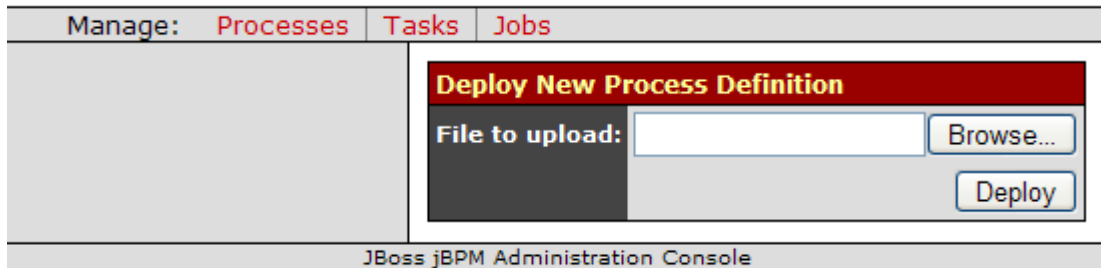


Figure 4. Someone with administrative privileges can deploy new process definition.

JBossESB to jBPM

JBossESB can make calls into jBPM using the BpmProcessor action. This action uses the jBPM command API to make calls into jBPM. The following jBPM commands have been implemented:

- **NewProcessInstanceCommand** - Start a new ProcessInstance given a process definition that was already deployed to jBPM. The NewProcessInstanceCommand leaves the Process Instance in the start state, which would be needed if there is an task associated to the Start node (i.e. some task on some actor's tasklist). In most cases however you would like the new Process Instance to move to the first node, which is where the next command comes in.
- **StartProcessInstanceCommand** - Identical to the NewProcessInstanceCommand, but additionally the new Process Instance is moved from the Start position into the first Node.
- **CancelProcessInstanceCommand** - Cancel a ProcessInstance. i.e. when an event comes in which should result in the cancellation of the entire ProcessInstance. This action requires some jBPM context variables to be set on the message, in particular the ProcessInstance Id. Details on that are discussed later.

The configuration for this action in the jboss-esb.xml looks like

```
<action name="create_new_process_instance"
  class="org.jboss.soa.esb.services.jbpm.actions.BpmProcessor">
  <property name="command" value="StartProcessInstanceCommand" />
  <property name="process-definition-name"
    value="processDefinition2"/>
  <property name="actor" value="FrankSinatra"/>
  <property name="esbToBpmVars">
    <!-- esb-name maps to getBody().get("eVar1") -->
    <mapping esb="eVar1" bpm="counter" default="45" />
    <mapping esb="BODY_CONTENT" bpm="theBody" />
  </property>
</action>
```

There are two required action attributes:

- **name** - required attribute. You are free to use any value for the name attribute as long as it is unique in the action pipeline.

- **class** - required attribute. This attributes needs to be set to “org.jboss.soa.esb.services.jbpm.actions.BpmProcessor”

Furthermore one can configure the following configuration properties:

- **command** – required property. Needs to be one of: NewProcessInstance-Command, StartProcessInstanceCommand or CancelProcessInstance-Command.
- **processdefinition** – required property for the New- and Start-ProcessInstanceCommands *if the process-definition-id property is not used*. The value of this property should reference a process definition that is already deployed to jBPM and of which you want to create a new instance. This property does not apply to the Signal- and CancelProcessInstance-Commands.
- **process-definition-id** – required property for the New- and Start-ProcessInstanceCommands *if the processdefinition property is not used*. The value of this property should reference a processdefinition id in jBPM of which you want to create a new instance. This property does not apply to the Signal- and CancelProcessInstanceCommands.
- **actor** – optional property to specify the jBPM actor id, which applies to the New- and StartProcessInstanceCommands only.
- **key** – optional property to specify the value of the jBPM key. For example one can pass a unique invoice id as the value for this key. On the jBPM side this key is as the “business” key id field. The key is a string based business key property on the process instance. The combination of business key + process definition must be unique if a business key is supplied. The key value can hold an MVEL expression to extract the desired value from the EsbMessage. For example if you have a named parameter called “businessKey” in the body of your message you would use “body.businessKey”. Note that this property is used for the New- and StartProcessInstanceCommands only.
- **transition-name** – optional property. This property only applies to the StartProcessInstance- and Signal Commands, and is of use only if there are *more than one* transition out of the current node. If this property is not specified the *default* transition out of the node is taken. The default transition is the *first* transition in the list of transition defined for that node in the jBPM processdefinition.xml.
- **esbToBpmVars** - optional property for the New- and StartProcessInstanceCommands. This property defines a list of variables that need to be extracted from the EsbMessage and set into jBPM context for the particular process instance. The list consists of mapping elements. Each mapping element can have the following attributes:
 - **esb** – required attribute which can contain an MVEL expression to extract a value anywhere from the EsbMessage.
 - **bpm** – optional attribute containing the name which be used on the jBPM side. If omitted the esb name is used.
 - **default** – optional attribute which can hold a default value if the esb MVEL expression does not find a value set in the EsbMessage.

Finally some variables can be set on the **body** of the EsbMessage:

- **jbpmProcessInstId** – required parameter which applies to the Cancel-ProcessInstanceCommand only. It is up to the user make sure this value is set as a named parameter on the EsbMessage body.

Exception Handling JBossESB to jBPM

For ESB calls into jBPM an exception of the type JbpmException can be thrown from the jBPM Command API. This exception is not handled by the integration and we let it propagate into the ESB Action Pipeline code. The action pipeline will log the error, send the message to the DeadLetterService (DLS), and send the an error message to the faultTo EPR, if a faultTo EPR is set on the message.

jBPM to JBossESB

The JBossESB to jBPM maybe interesting but the other way around is probably far more interesting jBPM to JBossESB communication provides us with the capability to use jBPM for service orchestration. Service Orchestration itself will be discussed in more detail in the next chapter and here we're focusing on the details of the integration first. The integration implements two jBPM action handler classes. The classes are “EsbActionHandler” and “EsbNotifier”. The EsbActionHandler is a request-reply type action, which drops a message on a Service and then waits for a response while the EsbNotifier only drops a message on a Service and continues its processing. The interaction with JBossESB is asynchronous in nature and does not block the process instance while the Service executes. First we'll discuss the EsbNotifier as it implements a subset of the configuration of EsbActionHandler class.

EsbNotifier

The EsbNotifier action should be attached to an outgoing transition. This way the jBPM processing can move along while the request to the ESB service is processed in the background. In the jBPM processdefinition.xml we would need attach the EsbNotifier to the outgoing transition. For example the configuration for a “Ship It” node could look like:

```
<node name="ShipIt">
  <transition name="ProcessingComplete" to="end">
    <action name="ShipItAction" class=
      "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbNotifier">
      <esbCategoryName>BPM_Orchestration4</esbCategoryName>
      <esbServiceName>ShippingService</esbServiceName>
      <bpmToEsbVars>
        <mapping bpm="entireCustomerAsObject" esb="customer" />
        <mapping bpm="entireOrderAsObject" esb="orderHeader" />
        <mapping bpm="entireOrderAsXML" esb="entireOrderAsXML" />
      </bpmToEsbVars>
    </action>
  </transition>
</node>
```

The following attributes can be specified:

- **name** – required attribute. User specified name of the action

- **class** – required attribute. Required to be set to `org.jboss.soa.esb.services.jbpm.actionhandlers.EsbNotifier`

The following subelements can be specified:

- **esbCategoryName** – required element. The category name of the ESB service
- **esbServiceName** – required element. The name of the ESB service.
- **globalProcessScope** - optional element. This boolean valued parameter sets the default scope in which the bpmToEsbVars are looked up. If the globalProcessScope is set to true the variables are looked for up the token hierarchy (= process-instance scope). If set to false it retrieves the variables in the scope of the token. If the given token does not have a variable for the given name, the variable is searched for up the token hierarchy. If omitted the globalProcessScope is set to false.
- **bpmToEsbVars** – optional element. This element takes a list of mapping subelements to map a jBPM context variable to a location in the EsbMessage. Each mapping element can have the following attributes:
 - **bpm** – required attribute. The name of the variable in jBPM context. The name can be MVEL type expression so you can extract a specific field from a larger object. The MVEL root is set to the jBPM “ContextInstance”, so for example you can use mapping like:

```
<mapping bpm="token.name" esb="TokenName" />
<mapping bpm="node.name" esb="NodeName" />
<mapping bpm="node.id" esb="esbNodeId" />
<mapping bpm="node.leavingTransitions[0].name"
        esb="transName" />
<mapping bpm="processInstance.id"
        esb="piId" />
<mapping bpm="processInstance.version"
        esb="piVersion" />
```

and one can reference jBPM context variable names directly.

- **esb** – optional attribute. The name of the variable on the EsbMessage. The name can be a MVEL type expression. By default the variable is set as a named parameter on the body of the EsbMessage. If you decide to omit the esb attribute, the value of the bpm attribute is used.
- **default** – optional attribute. If the variable is not found in jBPM context the value of this field is taken instead.
- **process-scope** – optional attribute. This boolean valued parameter can override the setting of the setting of the globalProcessScope for this mapping.

When working on variable mapping configuration it is recommended to turn on debug level logging.

EsbActionHandler

The EsbActionHandler is designed to work as a reply-response type call into JBossESB. The EsbActionHandler should be attached to the node. When this node is

entered this action will be called. The EsbActionHandler executes and leaves the node waiting for a transition signal. The signal can come from any other thread of execution, but under normal processing the signal will be send by the JBossESB callback Service. An example configuration for the EsbActionHandler could look like:

```
<node name="Intake Order">
  <action name="esbAction" class=
    "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>BPM_Orchestration4</esbCategoryName>
    <esbServiceName>IntakeService</esbServiceName>
    <bpmToEsbVars>
      <mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
    </bpmToEsbVars>
    <esbToBpmVars>
      <mapping esb="body.entireOrderAsXML" bpm="entireOrderAsXML" />
      <mapping esb="body.orderHeader" bpm="entireOrderAsObject" />
      <mapping esb="body.customer" bpm="entireCustomerAsObject" />
      <mapping esb="body.order_orderId" bpm="order_orderid" />
    </esbToBpmVars>
  </action>

  <transition name="" to="Review Order"></transition>
</node>
```

The configuration for the EsbActionHandler action *extends* the EsbNotifier configuration. The extensions are the following subelements:

- **esbToBpmVars** – optional element. *This subelement is identical to the esbToBpmVars property mention in the previous section* “JBossESB to jBPM” for the BpmProcessor configuration. The element defines a list of variables that need to be extracted from the EsbMessage and set into jBPM context for the particular process instance. The list consists of mapping elements. Each mapping element can have the following attributes:
 - **esb** – required attribute which can contain an MVEL expression to extract a value anywhere from the EsbMessage.
 - **bpm** – optional attribute containing the name which be used on the jBPM side. If omitted the esb name is used.
 - **default** – optional attribute which can hold a default value if the esb MVEL expression does not find a value set in the EsbMessage.
- **exceptionTransition** – optional element. The name of the transition that should be taken if an exception occurs while processing the Service. This requires the current node to have more then one outgoing transition where one of the transition handles “exception processing”.

Optionally you may want to specify a timeout value for this action. For this you can use a jBPM native Timer on the node. If for example you only want to wait 10 seconds for the Service to complete you could add

```
<timer name='timeout' duedate='10 seconds' transition='time-out'/>
```

to the node element. Now if no signal is received within 10 seconds of entering this node, the transition called “time-out” is taken.

There are two types of scenarios where exceptions can arise.

- The first type of exception is a `MessageDeliveryException` which is thrown by the `ServiceInvoker`. If this occurs it means that delivery of the message to the ESB failed. If this happens things are pretty bad and you have probably misspelled the name of the Service you are trying to reach. This type of exception can be thrown from both the `EsbNotifier` as well as the `EsbActionHandler`. In the jBPM node one can add an [ExceptionHandler](#) [TB-JBPM-USER] to handle this exception.
- The second type of exception is when the Service received the request, but something goes wrong during processing. Only if the call was made from the `EsbActionHandler` it makes sense to report back the exception to jBPM. If the call was made from the `EsbNotifier` jBPM processing has already moved on, and it is of little value to notify the process instance of the exception. This is why the exception-transition can only be specified for `EsbActionHandler`.

To illustrate the type of error handling that is now possible using standard jBPM features we will discuss some scenarios illustrated in Figure 5.

Scenario 1. Time-out

When using the `EsbActionHandler` action and the node is waiting for a callback, it maybe that you want to limit how long you want to wait for. For this scenario you can add a timer to the node. This is how `Service1` is setup in Figure 5. The timer can be set to a certain due date. In this case it is set to 10 seconds. The process definition configuration would look like

```
<node name="Service1">
  <action class=
    "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>MockCategory</esbCategoryName>
    <esbServiceName>MockService</esbServiceName>
  </action>
  <timer name='timeout' duedate='10 seconds'
    transition='time-out-transition'/>
  <transition name="ok" to="Service2"></transition>
  <transition name="time-out-transition" to="ExceptionHandler"/>
</node>
```

Node “Service1” has 2 outgoing transitions. The first one is called “ok” while the second one is called “time-out-transition”. Under normal processing the call back would signal the default transition, which is the “ok” transition since it is defined first. However if the execution of the service takes more then 10 seconds the timer will fire. The transition attribute of the timer is set to “time-out-transition”,so this transition will be taken on time-out. In Figure 5 this means that the processing ends up in the “ExceptionHandler” node in which one can perform compensating work.

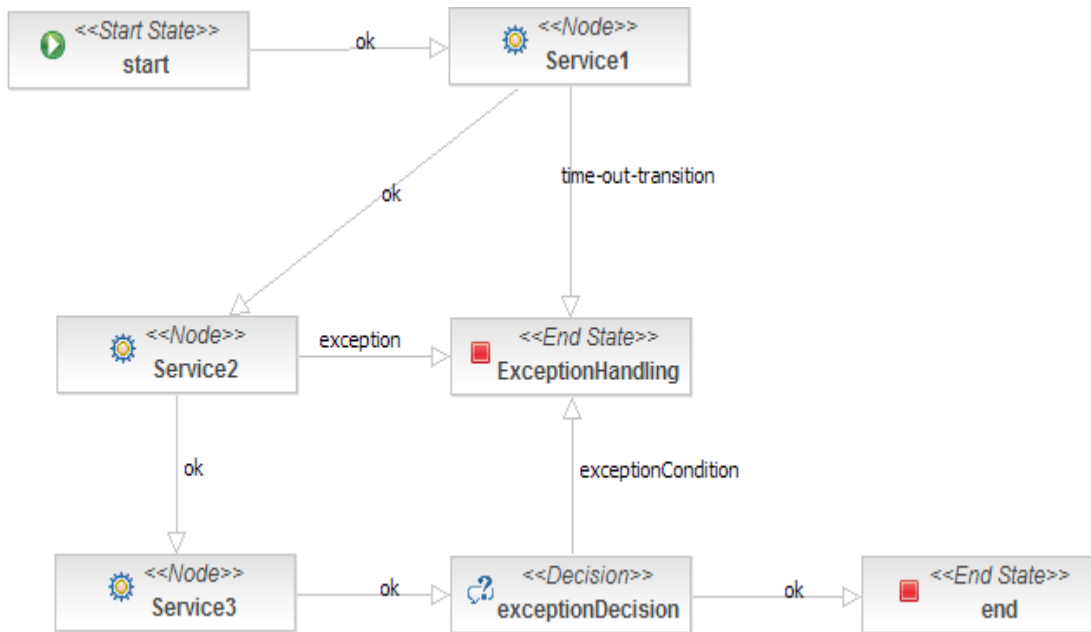


Figure 5. Three exception handling scenarios: time-out, exception-transition and exception-decision.

Scenario 2. Exception Transition

To handle exception that may occur during processing of the Service, one can define an exceptionTransition. When doing so the faultTo EPR is set on the message such that the ESB will make a callback to this node, signaling it with the exceptionTransition. Service2 has two outgoing transitions. Transition “ok” will be taken under normal processing, while the “exception” transition will be taken when the Service processing throws an exception. The definition of Service2 looks like

```

<node name="Service2">
  <action class=
    "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>MockCategory</esbCategoryName>
    <esbServiceName>MockService</esbServiceName>
    <exceptionTransition>exception</exceptionTransition>
  </action>
  <transition name="ok" to="Service3"></transition>
  <transition name="exception" to="ExceptionHandling"/>
</node>

```

where in the action, the exceptionTransition is set to “exception”. In this scenario the process also ends in the “ExceptionHandling” node.

Scenario 3. Exception Decision

Scenario 3 is illustrated in the configuration of Service3 and the “exceptionDecision” node that follows it. The idea is that processing of Service3 completes normally and the default transition out of node Service3 is taken. However, somewhere during the

Service execution an `errorCode` was set, and the “exceptionDecision” node checks if a variable called “errorCode” was set. The configuration would look like

```
<node name="Service3">
  <action class=
    "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>MockCategory</esbCategoryName>
    <esbServiceName>MockService</esbServiceName>
    <esbToBpmVars>
      <mapping esb="SomeExceptionCode" bpm="errorCode"/>
    </esbToBpmVars>
  </action>
  <transition name="ok" to="exceptionDecision"></transition>
</node>

<decision name="exceptionDecision">
  <transition name="ok" to="end"></transition>
  <transition name="exceptionCondition" to="ExceptionHandling">
    <condition>#{ errorCode!=void }</condition>
  </transition>
</decision>
```

where the `esbToBpmVars` mapping element extracts the `errorCode` called “SomeExceptionCode” from the `EsbMessage` body and sets in the jBPM context, if this “SomeExceptionCode” is set that is. In the next node “exceptionDecision” the “ok” transition is taken under normal processing, but if a variable called “errorCode” is found in the jBPM context, the “exceptionCondition” transition is taken. This is using the decision node feature of jBPM where transition can nest a condition. Here we check for the existence of the “errorCode” variable using the condition

```
<condition>#{ errorCode!=void }</condition>
```

For more details on conditional transitions please see the jBPM documentation [TB-JBPM-USER].

Service Orchestration

Introduction

Service Orchestration is the arrangement of business processes. Traditionally BPEL is used to execute SOAP based WebServices. In the Guide 'Service Orchestration' you can obtain more details on how to use ActiveBPEL with JBossESB [TF-BPEL]. If you want to orchestrate JBossESB regardless of their end point type, then it makes more sense to use jBPM. This chapter explains how to use the integration discussed in Chapter 1 to do Service Orchestration using jBPM.

Orchestration Diagram

A key component of Service Orchestration is to use a flow-chart like design tool to design and deploy processes. The jBPM IDE can be used for just this. Figure 6 shows an example of such a flow-chart, which represents a simplified order process. This example is taken from the bpm_orchestration4 quick start [JBESB-QS] which ships with JBossESB.

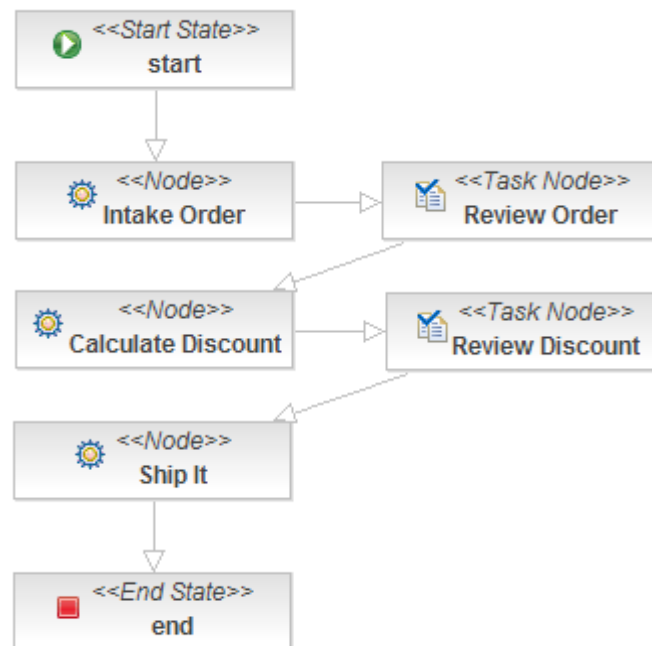


Figure 6. “Order Process” Service Orchestration using jBPM

In the “Order Process” Diagram three of the nodes are JBossESB Services, the “Intake Order”, “Calculate Discount” and the “Ship It” nodes. For these nodes the regular “Node” type was used, which is why these are labeled with “<<Node>>”. Each of these nodes have the EsbActionHandler attached to the node itself. This

means that the jBPM node will send a request to the Service and then it will remain in a wait state, waiting for the ESB to call back into the node with the response of the Service. The response of the service can then be used within jBPM context. For example when the Service of the “Intake Order” responds, the response is then used to populate the “Review Order” form. The “Review Order” node is a “Task Node”. Task Nodes are designed for human interaction. In this case someone is required to review the order before the Order Process can process.

To create the diagram in Figure 6, select File > New > Other, and from the Selection wizard select “JBoss jBPM “Process Definition” as shown in Figure 7. The wizard will direct you to save the process definition. From an organizational point of view it is recommended use one directory per process definition, as you will typically end up with multiple files per process design.

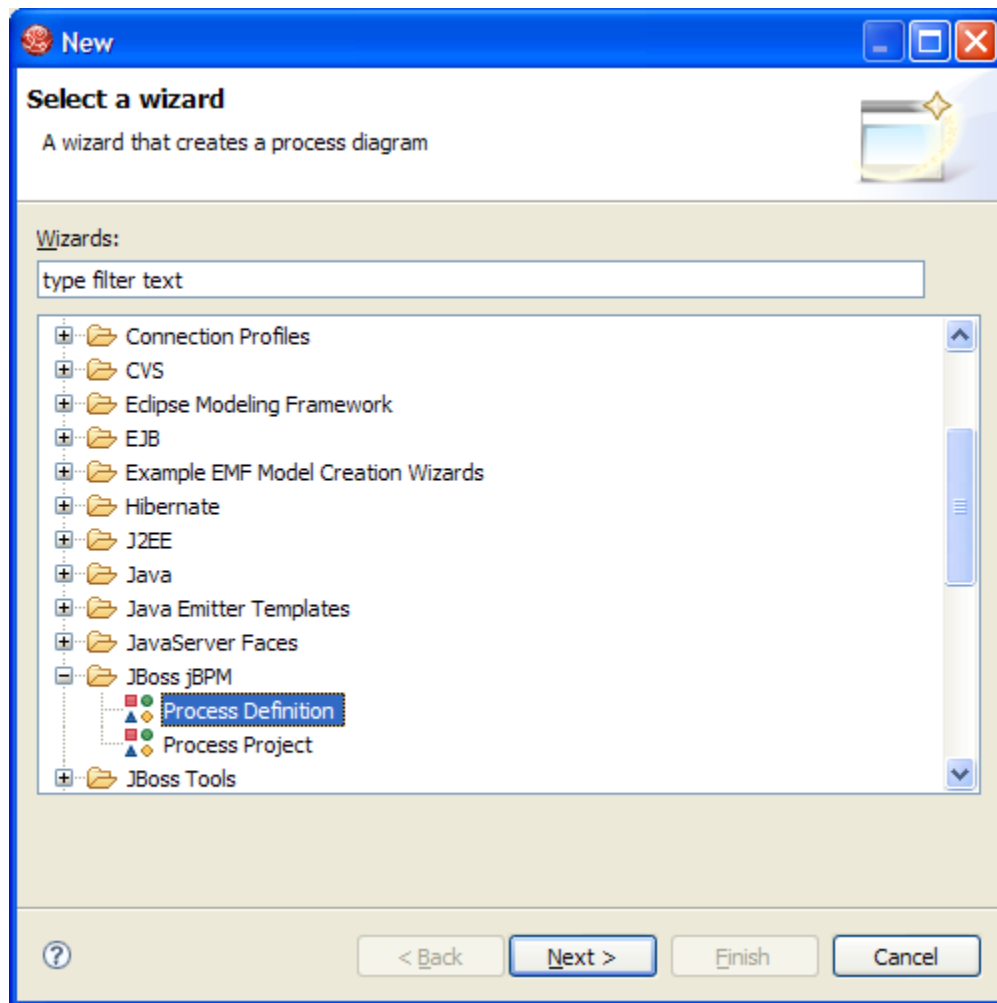


Figure 7. Select new JBoss jBPM Process Definition

After creating a new process definition. You can drag and drop any item from menu, shown in Figure 8, into the process design view. You can switch between the design and source modes if needed to check the XML elements that are being added, or to add XML fragments that are needed for the integration. Recently a new type of node was created by Koen Aers called “ESB Service” [KA-BLOG]. Currently this works

with the old jBPM integration, from before this document was written. Some small updates will be needed to make it work with the current implementation. So please check Koen's blog for any updates on this.

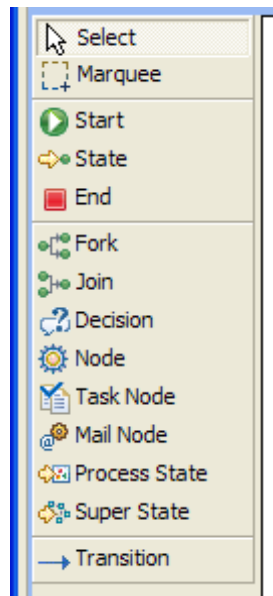


Figure 8. jBPM IDE menu palette.

Before building the “Order Process” diagram of Figure 6, we'd need to create and test the three Services. These services are 'ordinary' ESB services and are defined in the jboss-esb.xml. Check the jboss-esb.xml of the bpm_orchestration4 quick start [JBESB-QS] if you want details on them, but they only thing of importance to the Service Orchestration are the Services names and categories as shown in the following jboss-esb.xml fragment:

```
<services>

  <service category="BPM_orchestration4_Starter_Service"
    name="Starter_Service"
    description="BPM Orchestration Sample 4: Use this service to
    start a process instance">
    ....
  </service>
  <service category="BPM_Orchestration4" name="IntakeService"
    description="IntakeService: transforms, massages, calculates
    priority">
    ....
  </service>
  <service category="BPM_Orchestration4" name="DiscountService"
    description="DiscountService">
  </service>
  <service category="BPM_Orchestration4" name="ShippingService"
    description="ShippingService">
    ....
  </service>

</services>
```

These Service can be referenced using the EsbActionHandler or EsbNotifier Action Handlers as discussed in Chapter 1. The EsbActionHandler is used when jBPM

expects a response, while the EsbNotifier can be used if no response back to jBPM is needed.

Now that the ESB services are known we drag the “Start” state node into the design view. A new process instance will start a process at this node. Next we drag in a “Node” (or “ESB Service “if available). Name this Node “Intake Order”. We can connect the Start and the Intake Order Node by selecting “Transition” from the menu and by subsequently clicking on the Start and Intake Order Node. You should now see an arrow connecting these two nodes, pointing to the Intake Order Node.

Next we need to add the Service and Category names to the Intake Node. Select the “Source” view. The “Intake Order Node should look like

```
<node name="Intake Order">
  <transition name="" to="Review Order"></transition>
</node>
```

and we add the EsbHandlerAction class reference and the subelement configuration for the Service Category and Name, BPM_Orchestration4 and “IntakeService” respectively

```
<node name="Intake Order">
  <action name="esbAction" class="
    org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>BPM_Orchestration4</esbCategoryName>
    <esbServiceName>IntakeService</esbServiceName>
    <!-- async call of IntakeService -->
  </action>

  <transition name="" to="Review Order"></transition>
</node>
```

Next we want to send the some jBPM context variables along with the Service call. In this example we have a variable named “entireOrderAsXML” which we want to set in the default position on the EsbMessage body. For this to happen we add

```
<bpmToEsbVars>
  <mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
</bpmToEsbVars>
```

which will cause the XML content of the variable “entireOrderAsXML” to end up in the body of the EsbMessage, so the IntakeService will have access to it, and the Service can work on it, by letting it flow through each action in the Action Pipeline. When the last action is reached it the replyTo is checked and the EsbMessage is send to the JBpmCallBack Service, which will make a call back into jBPM signaling the “Intake Order” node to transition to the next node (“Review Order”). This time we will want to send some variables from the EsbMessage to jBPM. Note that you can send entire objects as long both contexts can load the object's class. For the mapping back to jBPM we add an “esbToEsbVars” element. Putting it all together we end up with:

```
<node name="Intake Order">

  <action name="esbAction" class=
    "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>BPM_Orchestration4</esbCategoryName>
    <esbServiceName>IntakeService</esbServiceName>
    <bpmToEsbVars>
      <mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
    </bpmToEsbVars>
    <esbToBpmVars>
```



```

    <mapping esb="body.entireOrderAsXML" bpm="entireOrderAsXML" />
    <mapping esb="body.orderHeader" bpm="entireOrderAsObject" />
    <mapping esb="body.customer" bpm="entireCustomerAsObject" />
    <mapping esb="body.order_orderId" bpm="order_orderid" />
    <mapping esb="body.order_totalAmount" bpm="order_totalamount" />
    <mapping esb="body.order_orderPriority" bpm="order_priority" />
    <mapping esb="body.customer_firstName" bpm="customer_firstName" />
    <mapping esb="body.customer_lastName" bpm="customer_lastName" />
    <mapping esb="body.customer_status" bpm="customer_status" />
  </esbToBpmVars>
</action>
<transition name="" to="Review Order"></transition>
</node>

```

So after this Service returns we have the following variables in the jBPM context for this process: entireOrderAsXML, entireOrderAsObject, entireCustomerAsObject, and for demo purposes we also added some flattened variables: order_orderid, order_totalAmount, order_priority, customer_firstName, customer_lastName and customer_status.

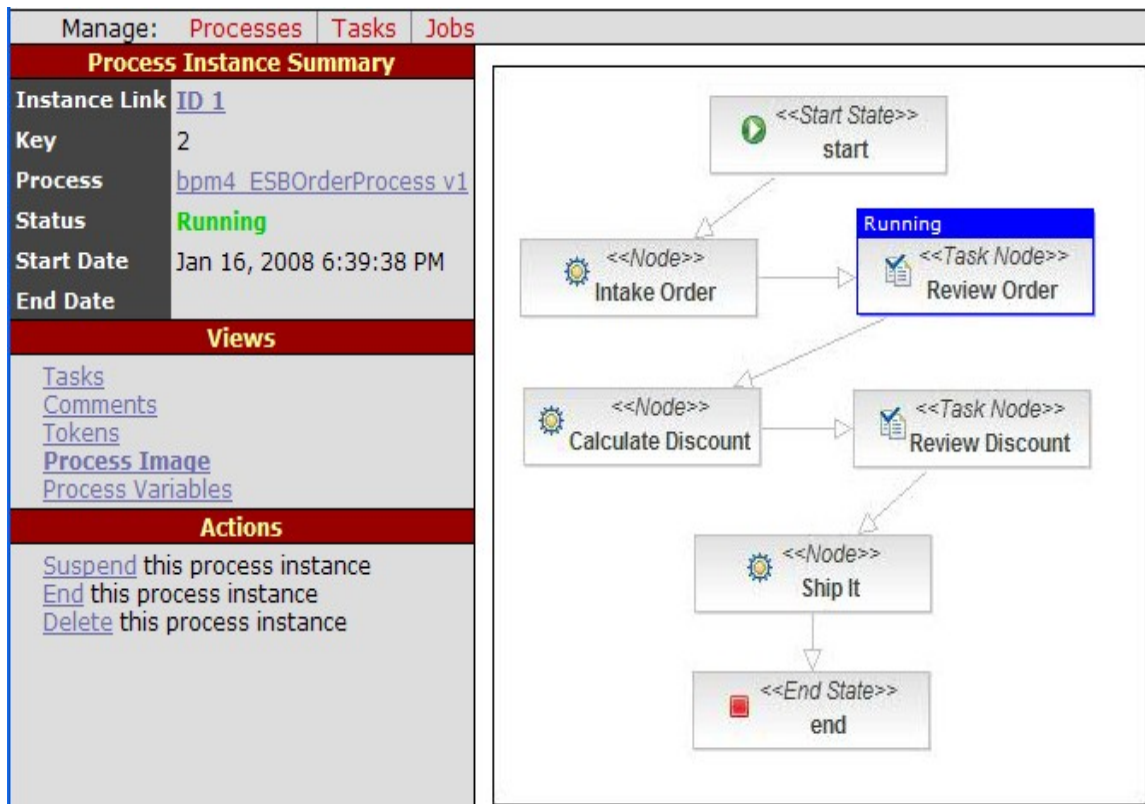


Figure 9. The Order process reached the “Review Order” node

In our Order process we require a human to review the order. We therefore add a “Task Node” and add the task “Order Review”, which needs to be performed by someone with actor_id “user”. The XML-fragment looks like

```

<task-node name="Review Order">
  <task name="Order Review">
    <assignment actor-id="user"></assignment>
  </task>
</task-node>

```

```

        <variable name="customer_firstName"
            access="read,write,required"></variable>
        <variable name="customer_lastName"
            access="read,write,required">
        <variable name="customer_status" access="read"></variable>
        <variable name="order_totalamount" access="read"></variable>
        <variable name="order_priority" access="read"></variable>
        <variable name="order_orderid" access="read"></variable>
        <variable name="order_discount" access="read"></variable>
        <variable name="entireOrderAsXML" access="read"></variable>
    </controller>
</task>
<transition name="" to="Calculate Discount"></transition>
</task-node>

```

In order to display these variables in a form in the jbp-console we need to create an xhtml dataform (see the Review_Order.xhtml file in the bpm_orchestration4 quick start [JBESB-QS] and tie this for this TaskNode using the forms.xml file:

```

<forms>
  <form task="Order Review" form="Review_Order.xhtml"/>
  <form task="Discount Review" form="Review_Order.xhtml"/>
</forms>

```

Note that in this case the same form is used in two task nodes. The variables are referenced in the Review Order form like

```

<jbpm:datacell>
  <f:facet name="header">
    <h:outputText value="customer_firstName"/>
  </f:facet>
  <h:inputText value="#{var['customer_firstName']}" />
</jbpm:datacell>

```

which references the variables set in the jBPM context.

When the process reaches the “Review Node”, as shown in Figure 9. When the 'user' user logs into the jbp-console the user can click on 'Tasks' to see a list of tasks, as shown in Figure 10. The user can 'examine' the task by clicking on it and the user will be presented with a form as shown in Figure 11. The user can update some of the values and click “Save and Close” to let the process move to the next Node.

Manage: Processes Tasks							
Tasks				First Prev - Page 1 of 1 - Next Last			
ID	Name	Pooled Actors	Assigned To	Status	Start Date	End Date	Actions
	<input type="text"/>		<input type="text"/>	<input checked="" type="checkbox"/> N <input checked="" type="checkbox"/> R <input checked="" type="checkbox"/> S <input type="checkbox"/> E			Apply Filter Clear Filter
1	Order Review		user	Not Started			Examine Suspend Start

Figure 10. The task list for user 'user'

Manage: Processes Tasks	
Task Summary	
Task Link	ID 1
Name	Order Review
Status	Not Started
Assigned To	user
Token	ID 1
Process Instance	ID 1
Process	bpm4_ESBOrderProcess v1
Created Date	Jan 16, 2008 6:39:48 PM
Views	
Task Form Comments Variables Transitions	
Actions	
Suspend this task Start this task Reassign this task to: <input type="text"/> <input type="button" value="Save"/>	
Order Review	
customer_firstName	<input type="text" value="Rex"/>
customer_lastName	<input type="text" value="Myers"/>
customer_status	<input type="text" value="60"/>
order_totalamount	<input type="text" value="64.92"/>
order_priority	<input type="text" value="3"/>
order_orderid	<input type="text" value="2"/>
order_discount	<input type="text"/>
entireOrder	<input type="text" value='<Order netAmount="59.97'/>
Actions	<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Save and Close"/>

Figure 11. The “Order Review” form.

The next node is the “Calculate Discount” node. This is an ESB Service node again and the configuration looks like

```
<node name="Calculate Discount">
  <action name="esbAction" class="
    org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>BPM_Orchestration4</esbCategoryName>
    <esbServiceName>DiscountService</esbServiceName>
    <bpmToEsbVars>
      <mapping bpm="entireCustomerAsObject" esb="customer" />
      <mapping bpm="entireOrderAsObject" esb="orderHeader" />
      <mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
    </bpmToEsbVars>
    <esbToBpmVars>
      <mapping esb="order"
        bpm="entireOrderAsObject" />
      <mapping esb="body.order_orderDiscount"
        bpm="order_discount" />
    </esbToBpmVars>
    </action>
    <transition name="" to="Review Discount"></transition>
  </node>
```

The Service receives the customer and orderHeader objects as well as the entireOrderAsXML, and computes a discount. The response maps the body.order_orderDiscount value onto a jBPM context variable called “order_discount”, and the process is signaled to move to the “Review Discount” task node.

Discount Review	
customer_firstName	<input type="text" value="Rex"/>
customer_lastName	<input type="text" value="Myers"/>
customer_status	<input type="text" value="60"/>
order_totalamount	<input type="text" value="64.92"/>
order_priority	<input type="text" value="3"/>
order_orderid	<input type="text" value="2"/>
order_discount	<input type="text" value="8.5"/>
entireOrder	<input type="text" value='<Order netAmount="59.97'/>
Actions	<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Save and Close"/>

Figure 12. The Discount Review form

The user is asked to review the discount, which is set to 8.5. On “Save and Close” the process moves to the “Ship It” node, which again is an ESB Service. If you don't want the Order process to wait for the Ship It Service to be finished you can use the EsbNotifier action handler and attach it to the outgoing transition:

```
<node name="ShipIt">
  <transition name="ProcessingComplete" to="end">
    <action name="ShipItAction" class=
      "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbNotifier">
      <esbCategoryName>BPM_Orchestration4</esbCategoryName>
      <esbServiceName>ShippingService</esbServiceName>
      <bpmToEsbVars>
        <mapping bpm="entireCustomerAsObject" esb="customer" />
        <mapping bpm="entireOrderAsObject" esb="orderHeader" />
        <mapping bpm="entireOrderAsXML" esb="entireOrderAsXML" />
      </bpmToEsbVars>
    </action>
  </transition>
</node>
```

After notifying the ShippingService the Order process moves to the 'end' state and terminates. The ShippingService itself may still be finishing up. In bpm_orchestration4 [JBESB-QS] it uses drools to determine whether this order should be shipped 'normal' or 'express'.

Process Deployment and Instantiation

In the previous paragraph we create the process definition and we quietly assumed we had an instance of it to explain the process flow. But now that we have created the processdefinition.xml, we can deploy it to jBPM using the IDE, ant or the jbpm-console (as explained in Chapter 1). In this example we use the IDE and deployed the files: Review_Order.xhtml, forms.xml, gpd.xml, processdefinition.xml and the processimage.jpg. On deployment the IDE creates a par archive and deploys this to the jBPM database. We do not recommend deploying Java code in par archives as it

may cause class loading issues. Instead we recommend deploying classes in jar or esb archives.

The screenshot displays the JBESB console interface for deploying a process. It is divided into four main sections:

- Files and Folders:** A list of files to be included in the process archive. The files are:
 - .gpd.test.xml (checked)
 - Review_Order.xhtml (checked)
 - forms.xml (checked)
 - gpd.xml (checked)
 - processdefinition.xml (checked)
 - processimage.jpg (checked)
- Java Classes and Resources:** A section for selecting Java classes and resources. It shows a folder icon and the text 'src'.
- Local Save Settings:** A section for choosing where to save the process archive locally. It includes a checkbox for 'Save Process Archive Locally' (unchecked), a 'Location:' text box, and a 'Search...' button. Below these is a 'Save Without Deploying...' button.
- Deployment Server Settings:** A section for specifying the settings of the server to deploy to. It includes:
 - Server Name: localhost
 - Server Port: 8080
 - Server Deployer: /jbpm-console/upload
 - A 'Test Connection...' button.

At the bottom right of the console, there is a 'Deploy Process Archive...' button.

Figure 13. Deployment of the “Order Process”

When the process definition is deployed a new process instance can be created. It is interesting to note that we can use the 'StartProcessInstanceCommand' which allows us to create a process instance with some initial values already set. Take a look at

```
<service category="BPM_orchestration4_Starter_Service"
  name="Starter_Service"
  description="BPM Orchestration Sample 4: Use this service to
  start a process instance">
  <listeners>
    ....
  </listeners>
  <actions>
    <action name="setup_key" class=
    "org.jboss.soa.esb.actions.scripting.GroovyActionProcessor">
      <property name="script"
        value="/scripts/setup_key.groovy" />
    </action>
    <action name="start_a_new_order_process" class=
    "org.jboss.soa.esb.services.jbpm.actions.BpmProcessor">
      <property name="command"
        value="StartProcessInstanceCommand" />
      <property name="process-definition-name"
        value="bpm4_ESBOrderProcess" />
      <property name="key" value="body.businessKey" />
      <property name="esbToBpmVars">
```

```

        <mapping esb="BODY_CONTENT" bpm="entireOrderAsXML" />
    </property>
</action>
</actions>
</service>

```

where new process instance is invoked and using some groovy script, and the jBPM key is set to the value of 'OrderId' from an incoming order XML, and the same XML is subsequently put in jBPM context using the esbToBpmVars mapping. In the bpm_orchestration4 quickstart [JBESB-QS] the XML came from the Seam DVD Store and the "SampleOrder.xml" looks like

```

<Order orderId="2" orderDate="Wed Nov 15 13:45:28 EST 2006"
statusCode="0" netAmount="59.97" totalAmount="64.92" tax="4.95">
  <Customer userName="user1" firstName="Rex" lastName="Myers"
state="SD"/>
  <OrderLines>
    <OrderLine position="1" quantity="1">
      <Product productId="364" title="Superman Returns"
price="29.98"/>
    </OrderLine>
    <OrderLine position="2" quantity="1">
      <Product productId="299" title="Pulp Fiction" price="29.99"/>
    </OrderLine>
  </OrderLines>
</Order>

```

Note that both ESB as well as jBPM deployments are hot. An extra feature of jBPM is that process deployments are versioned. Newly created process instances will use the latest version while existing process instances will finish using the process deployment on which they where started.

Conclusion

We have demonstrated how jBPM can be used to orchestrate Services as well as do Human Task Management. Note that you are free to use any jBPM feature. For instance look at the quick start bpm_orchestration2 [JBESB-QS] how to use the jBPM fork and join concepts.

Known Issues

We'll add them when we have them.

References

[JBESB-QS], JBossESB QuickStarts,
http://anonsvn.labs.jboss.com/labs/jbossesb/branches/JBESB_4_2_1_GA_CP/product/samples/quickstarts/

[KA-BLOG] ESB Service Node, Koen Aers,
<http://koentsje.blogspot.com/2008/01/esb-service-node-in-jbpm-jpdl-gpd-312.html>

[KA-JBPM-GPD], JBoss jBPM Graphical Process Designer, Koen Aers,
<http://docs.jboss.com/jbpm/v3/gpd/>

[TB-JBPM-USER] jBPM User Documentation, Tom Baaijens
<http://docs.jboss.com/jbpm/v3/userguide/>

[TF-BPEL], Service Orchestration using ActiveBPEL, Tom Fennely,
http://anonsvn.labs.jboss.com/labs/jbossesb/branches/JBESB_4_2_1_GA_CP/product/docs/services/ServiceOrchestration.pdf

Index

ActiveBPEL	21
actor	14
BpmProcessor	13
bpmToEsbVars	16
CancelProcessInstanceCommand	13
conditional transitions	20
create a Process Definition	10
database	7
DatabaseInitializer	7
deploy a process definition	12
Deployment of a Process Definition	10
design tool	21
EsbActionHandler	15, 16
esbCategoryName	16
EsbNotifier	15
esbServiceName	16
esbToBpmVars	14, 17
Exception Decision	19
Exception Handling	15, 18
Exception Transition	19
exceptionTransition	17
flow-chart	21
globalProcessScope	16
hibernate.cfg.xml	9
Human Task Management	7
JBoss Developer Studio	10
jboss-esb.xml	13
jBPM configuration	9
jBPM console	8
jbpm-ds.xml	8
jbpm.cfg.xml	9
jbpm.esb	7
jbpm.mail.templates.xml	9
jBPMCallbackBus	9
JBpmCallbackService	9
JbpmDS datasource	8
jbpmProcessInstId	15
JTA transaction manager	9
key	14
mapping	14
NewProcessInstanceCommand	13

Orchestration Diagram	21
Process Designer Plugin	10
process-definition-id	14
process-scope	16
processdefinition	14
processdefinition.xml	12, 15
replyTo	24
security settings	8
Service Orchestration	7, 21
StartProcessInstanceCommand	13
Time-out	18
timeout	17
Timer	17
transition-name	14
WebServices	21
