JBoss jBPM 3.2.2

A Guide to Process Modeling

for System Analysts & Developers

Version 1.0

Soumyajit Paul. Programmer Analyst, Cognizant
Mandrita Sinha Choudhuri. Associate, Cognizant

Contents

1.0 Introduction

JBoss jBPM - Java-based business process management (BPM) system - enables Enterprise Java and SOA programmers to create business process and workflow applications, business process orchestration and web application page flows from a single, flexible and scalable process engine.

1.1. Intended Audience

This document is intended for several levels of IT professionals – analysts, developers, project managers and architects.

• Analyst	Analysts may read through the following chapters for a basic understanding of how to design a process within the environment of jBPM - • Introduction • Readying the environment • Building Blocks for Modeling.
• Developer	Developers can focus on how to write services to integrate existing applications and expose them properly to assemble user interfaces. To have a good level of confidence they may read through the following chapters - • Introduction • Readying the Environment • Building Blocks for Modeling
• Project Manager	A project manager would be interested in getting an overall feel of the effort estimation in order to plan for required resources. Following chapters may be of help for this purpose – • Introduction. • Left column of the Workflow detail tables to get a feel of the activities involved. • Skim through the document.
• Architect	An Architect may wish to get an overview at the nuts-and-bolts know-how level, understand any JBoss jBPM product constraints and map to implications to the design, architecture and suggested solution. For this purpose, it will be helpful to – • Read the introduction. • Skim through the left column of the Workflow detail tables to get an understanding of the steps involved as part of the development approach • Read the right most column of the Workflow detail tables to get an insight on the technology implications of individual steps • Read/try-out the detailed steps in the Workflow detail tables for a good hands on understanding of the steps and what's involved

1.2. Assumptions/ Prerequisites

- Readers are expected to have a general understanding of Business Process Modeling and its relevance in solving business problems. This document will take the user step by step from problem understanding to solution realization as it covers the following topic:
    1. The different JBoss jbpm constructs and how they can be used to model various business problem scenarios.

1.3. Overview

JBoss jBPM Suite is a product suite for modeling, executing and optimizing business processes. JBoss jBPM provides both:

- Business service interaction on top of a service infrastructure backbone
- Freestanding, complete BPMS (Business Process Management System) based on SOA.

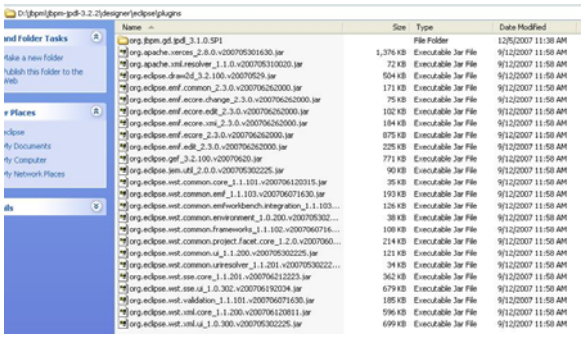
The various products and their interplay are summarized in the following table:

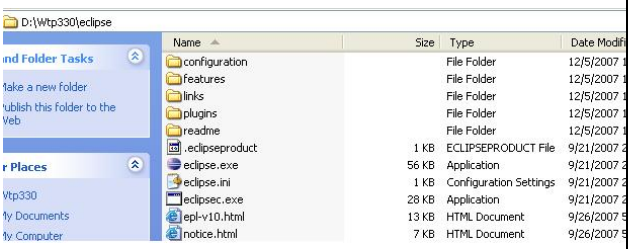
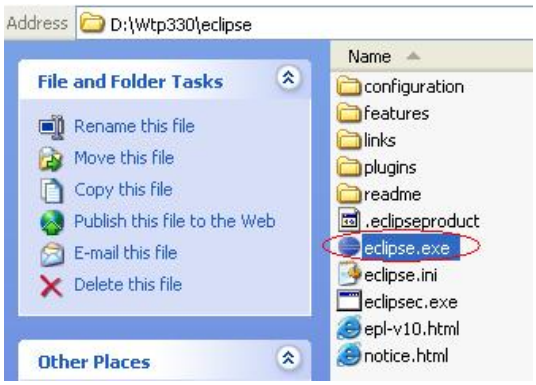
jBPM Product	Description	Comments
1. JBoss jBPM Product Suite	Process Design environment for the analyst	• JBoss jBPM enables automation of business processes that coordinate between people, applications and services • Designed for the mass market and support enterprise scale applications • JBoss jBPM bring process automation to a much wider set of business problems ranging from embedded workflow to enterprise business process orchestration and BPM. • JBoss jBPM delivers workflow, business process management (BPM) and service orchestration in a multi-process language platform.

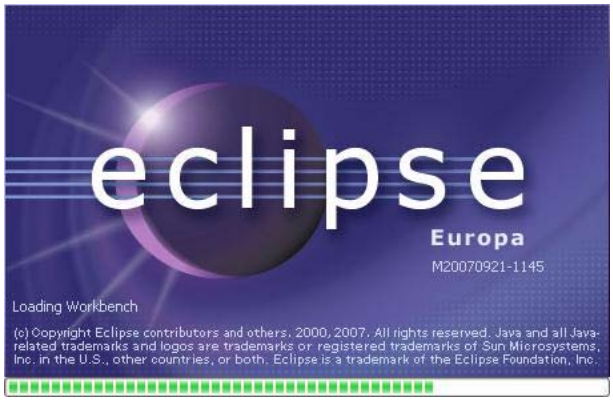
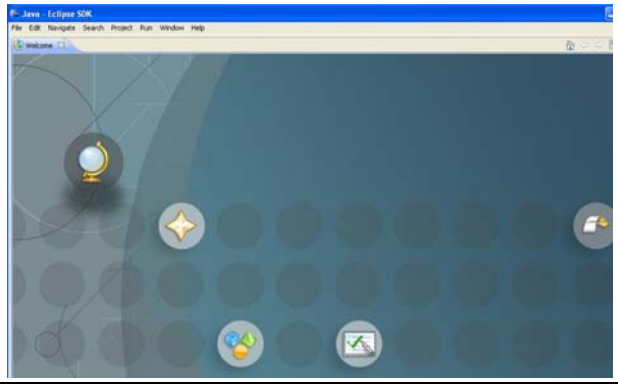
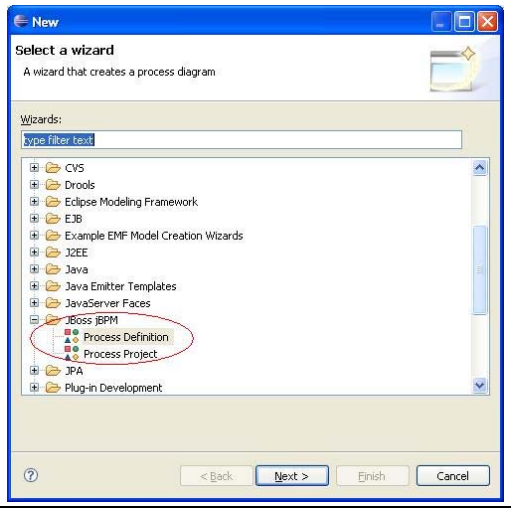
2.0 Readying the Environment

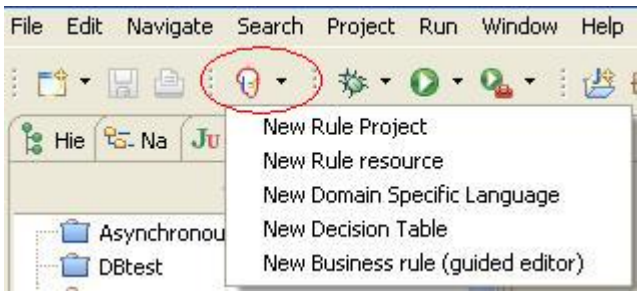
The following table details the steps to be followed for installation of jBPM Suite v3.2.2

For a list of compatible hardware/software, refer to *jBPM Release Note v3.2.2*

Steps	Description	Comments
Prerequisites	1. Obtain JDK 5 from Sun's Official site. 2. Set Environmental variable **JAVA_HOME** to point to the jdk installation directory. 3. Obtain a copy of jBPM Suite v 3.2.2 (jBPM-jpdl-3.2.2.zip) from JBoss official site.	The suite can be downloaded **for evaluation** from *http://www.JBoss.com/products/jBPM/downloads* (If the above site is not functional use the following) *http://sourceforge.net/project/showfiles.php?group_id=70542&package_id=145174&release_id=539054* (Download the jbpm-jdpl-suite.zip).
	4. Obtain Eclipse Web Tools Platform All-In-One Packages v3.3 (wtp-all-in-one-sdk-R-2.0.1-20070926042742-win32.zip) from eclipse. This ide will be required for modeling. jBPM designer comes as a plugin for Eclipse IDE.	JBPM 3.2.2 designer is compatible with eclipse 3.3.0.The Eclipse based IDE can be downloaded from *http://download.eclipse.org/webtools/downloads/drops/R2.0/R-2.0.1-20070926042742/* (If the above site is not functional alternatively the following site can be used) http://www.eclipse.org/
Installing jBPM Studio	1. Extract the .zip file (jBPM-jpdl-3.2.2.zip) in a suitable location.	
	2. Extract the .zip file (wtp-all-in-one-sdk-R-2.0.1-20070926042742-win32.zip) in a separate suitable location	
	3. Go to jBPM_JPDL_HOME>\designer\eclipse. 	
	4. Double click on the plugins directory 	jBPM Suite 3.2.2 provides plugins for its compatible eclipse based IDE .

Steps	Description	Comments
Installing jBPM Studio (Contd...)	5. Go to <*WTP_Eclipse_IDE_Home*>*eclipse**plugins*  	
	6. Paste <*jBPM_JPDL_HOME*>*designer**eclipse**plugins* contents to the <*WTP_Eclipse_IDE_Home*>*eclipse**plugins*.	
	7. Similarly copy the contents <*jBPM_JPDL_HOME*>*designer**eclipse**features* to the <*WTP_Eclipse_IDE_Home*>*eclipse**features*	
	8. Create a new folder called links inside <*jBPM_JPDL_HOME*>*designer**eclipse*\	
	9. Finally Copy the contents of <jBPM_JPDL_HOME>\designer\eclipse\links to the <WTP_Eclipse_IDE_Home>\eclipse\links	
Adding Drools Plugin	1. Download latest drool(JBoss Rules-4.0.x) plugin for eclipse Europa 3.3 workbench	The plugin can be downloaded from *http://labs.JBoss.com/drools/downloads.html*
	2. Extract the downloaded zip file.	
	3. Zip contains one jar file org.drools.eclipse_4.0.x.jar and a folder org.drools.eclipse.feature_4.0.x which contains an xml file name feature.xml. Copy the folder into <*WTP_Eclipse_IDE_Home*>*eclipse**features* and the jar into <*WTP_Eclipse_IDE_Home*>*eclipse**plugins*	
	4. Restart the Studio	
Verification of Insallation of jBPM Process Designer	1. Open the eclipse editor by double clicking the <WTP_Eclipse_IDE_Home>\eclipse\eclipse.exe with the following icon.  	
	2. This will open up the following.	

Steps	Description	Comments
		
	3. The next wizard shows a welcome screen like this. 	
Verification of Insallation of jBPM drools engine.	4. Close the welcome screen and Press Control+N. This will launch a wizard like this. 	The screen shows that the JBoss jBPM has been installed successfully within eclipse workbench.
	5. Drools workbench will be visible to your editor means a successful drools installation is done.	

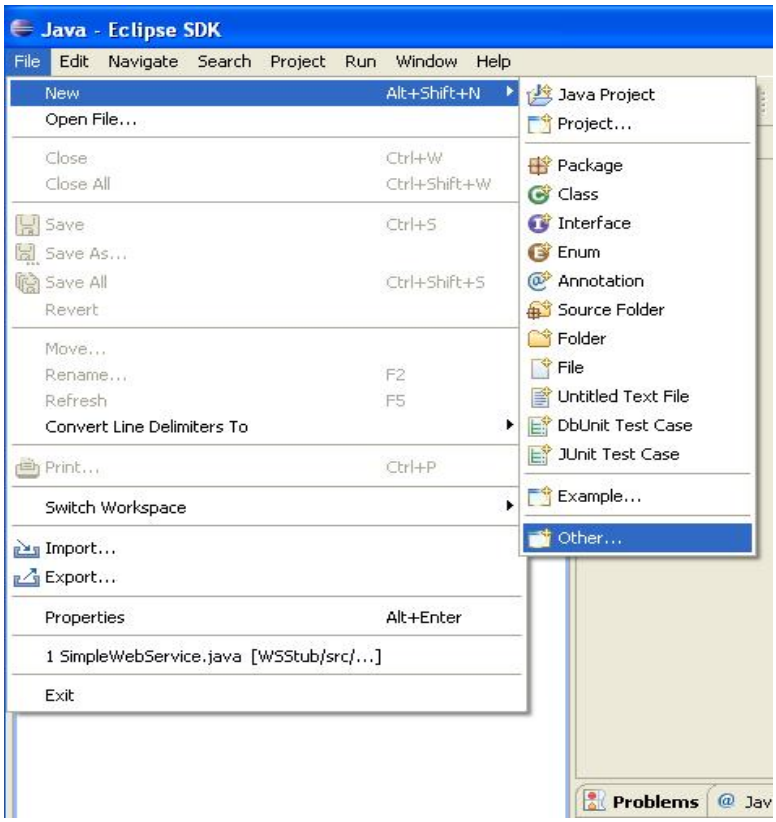
Steps	Description	Comments
		

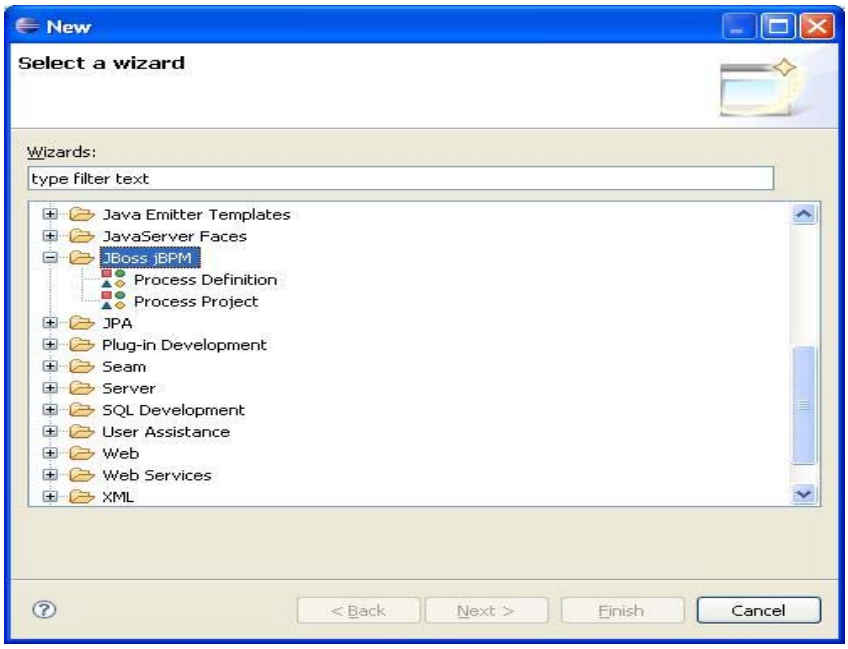
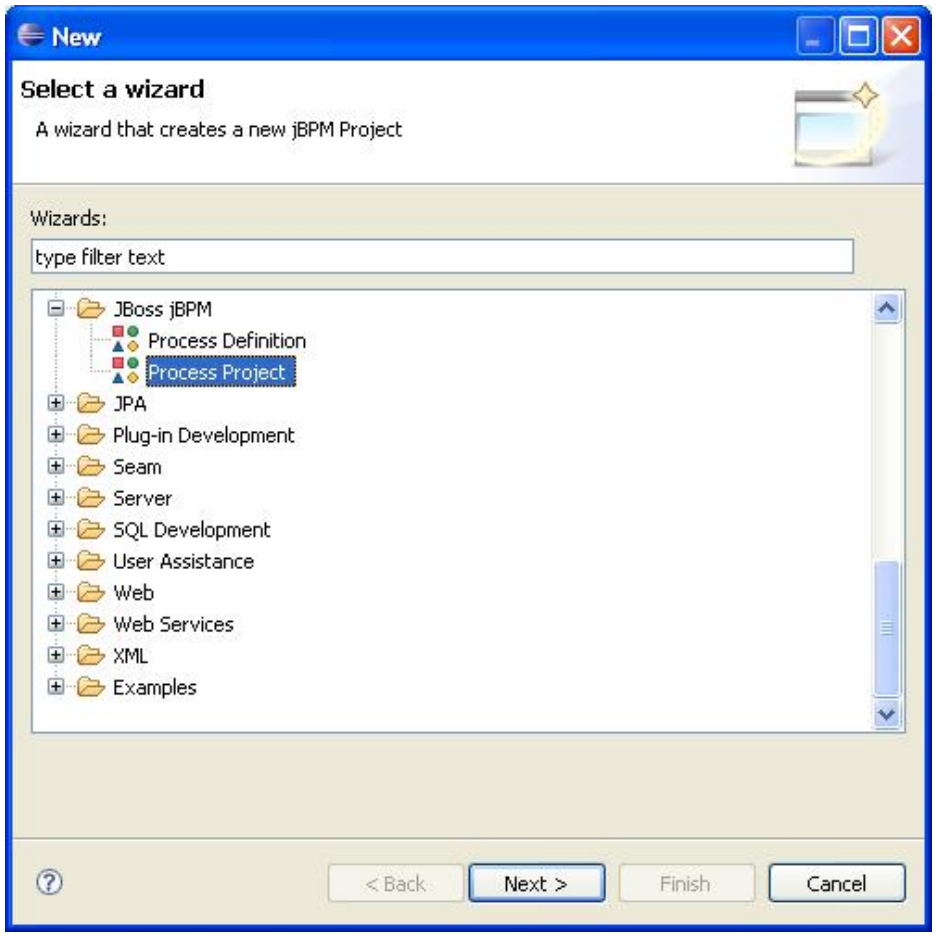
3.0 Building Blocks for Modeling

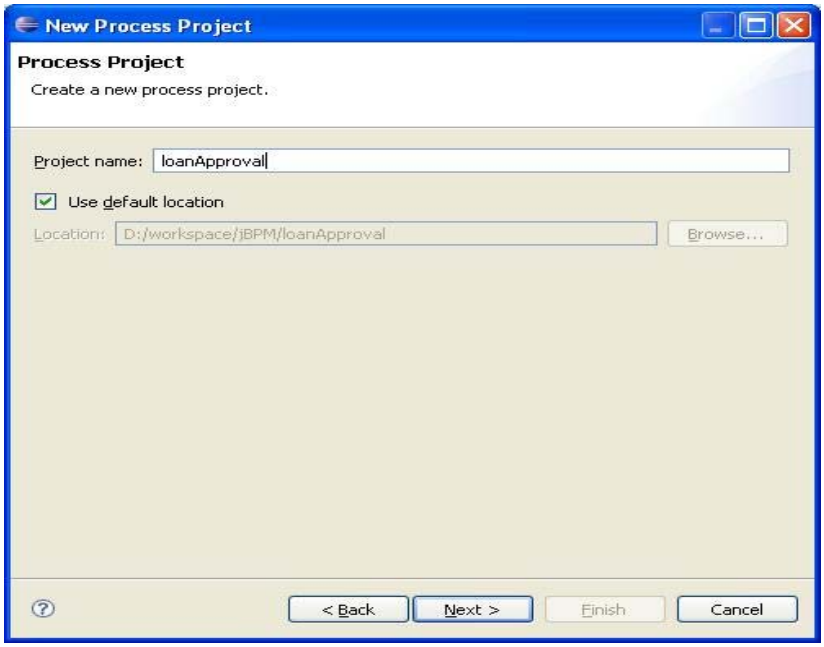
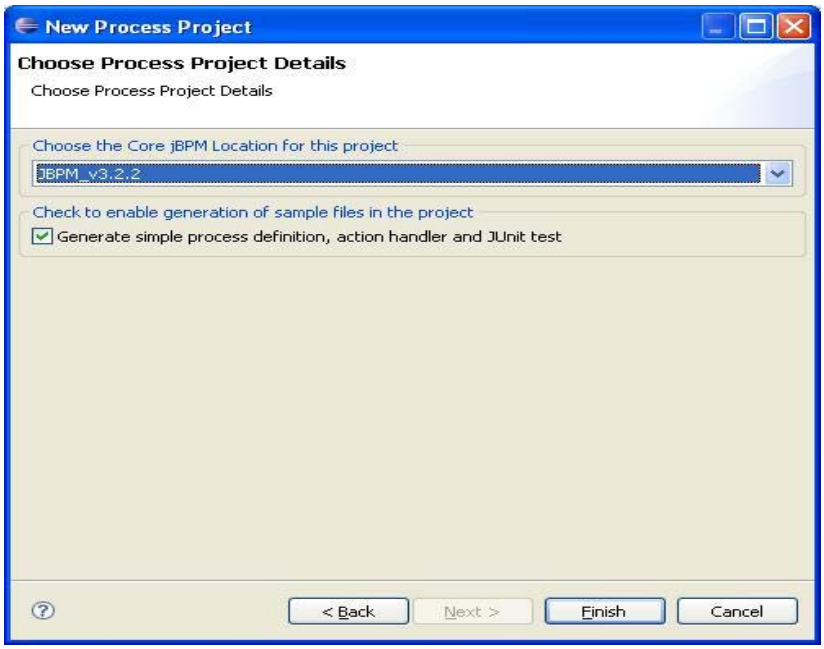
The following section details the building blocks for process modeling in jBPM studio which is built on top of eclipse.

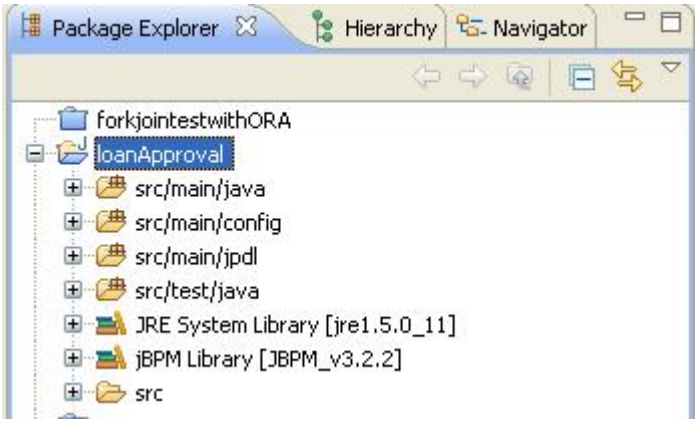
3.1. Creating a Process Project.

A jBPM process project is the collection of resources (models, external resources, codes) and represents a deployable unit. The following table lists out the steps to create a process Project in jBPM

Steps	Description	Comments
Create a new project	1. Start your WTP eclipse editor.	
	2. Click File Menu->New->Other 	
	3. Double Click on JBoss jBPM	

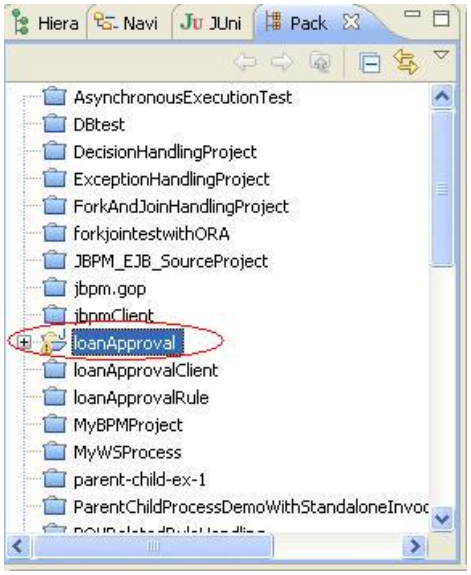
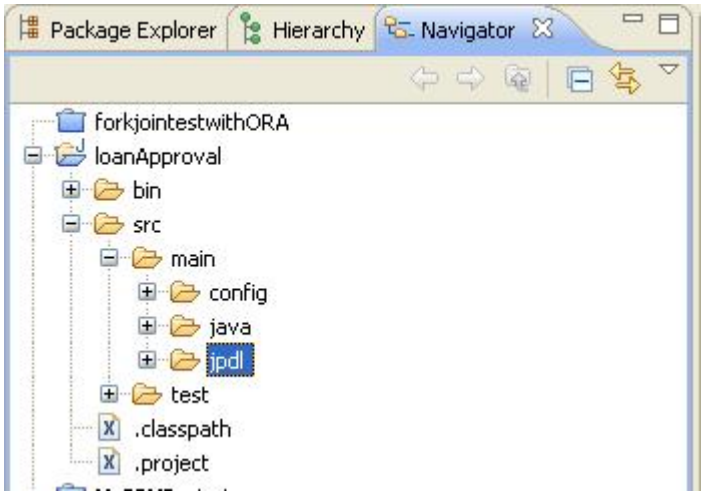
Steps	Description	Comments
		
Create a new project (Contd...)	4. Select Process Project. Click Next 	

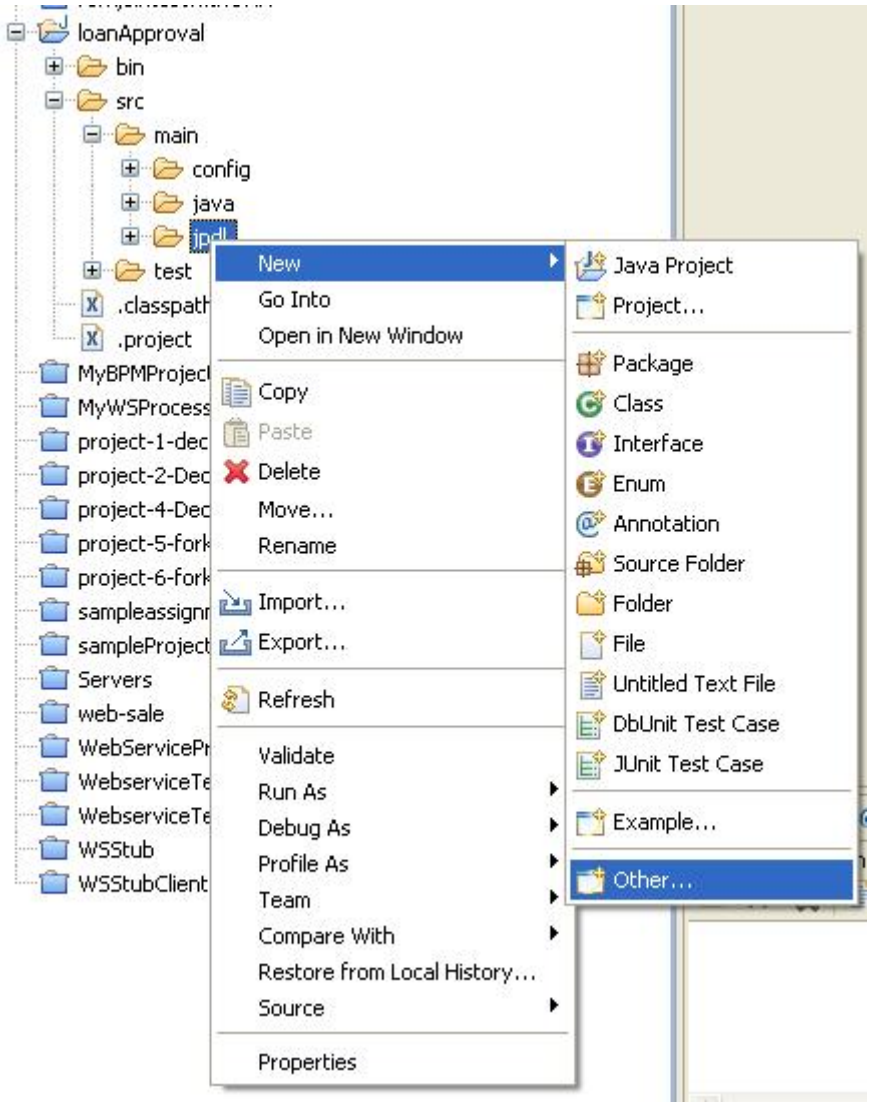
Steps	Description	Comments
	5. Give The Name Of your Process Project. Click Next. 	
	6. Choose the jBPM Location from the Drop Down. Click Finish when Done. 	1.If the JBPM runtime is not populated in the dropdown it needs to be set previously by pointing to the jbpm<version> installation directory. 2.If developer doesn't want to generate sample process definition,action handler and Junit Test case then the checkbox can be unchecked.
Create a new project (Contd...)	7. A Process Project can be viewed in the Project Explorer pane on the left side of the editor.	On creating a *Process Project* the following folders get generated.

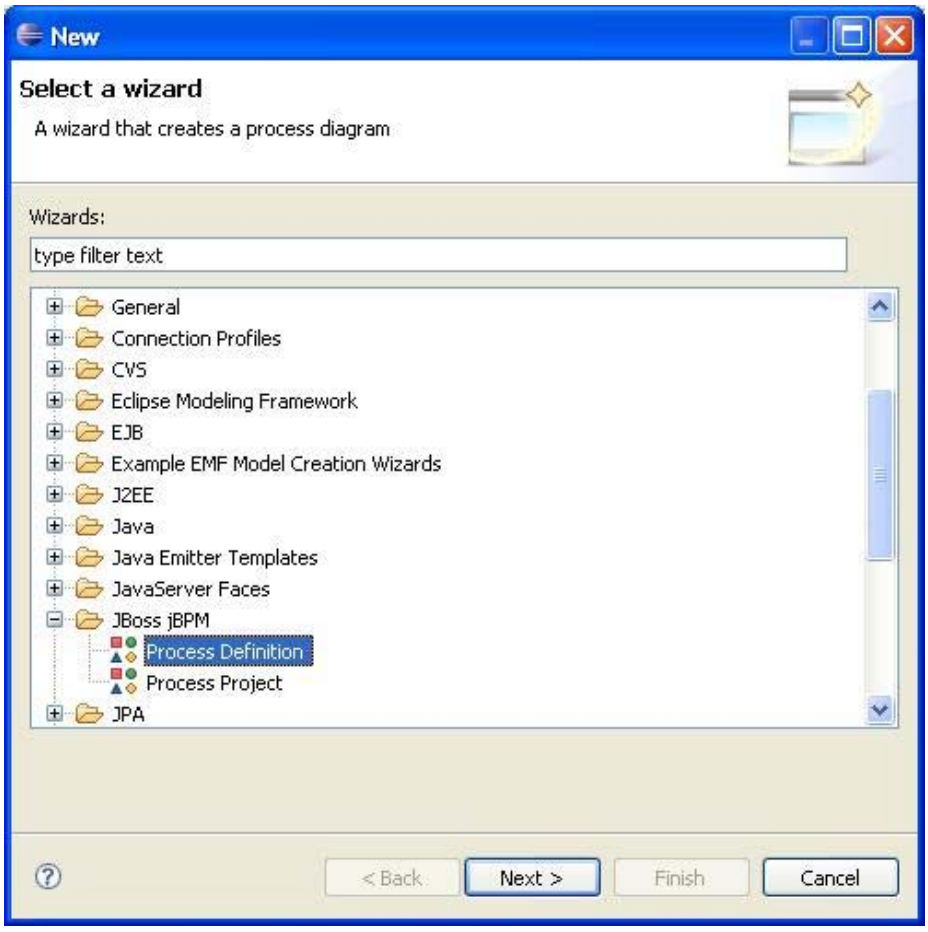
Steps	Description	Comments
		

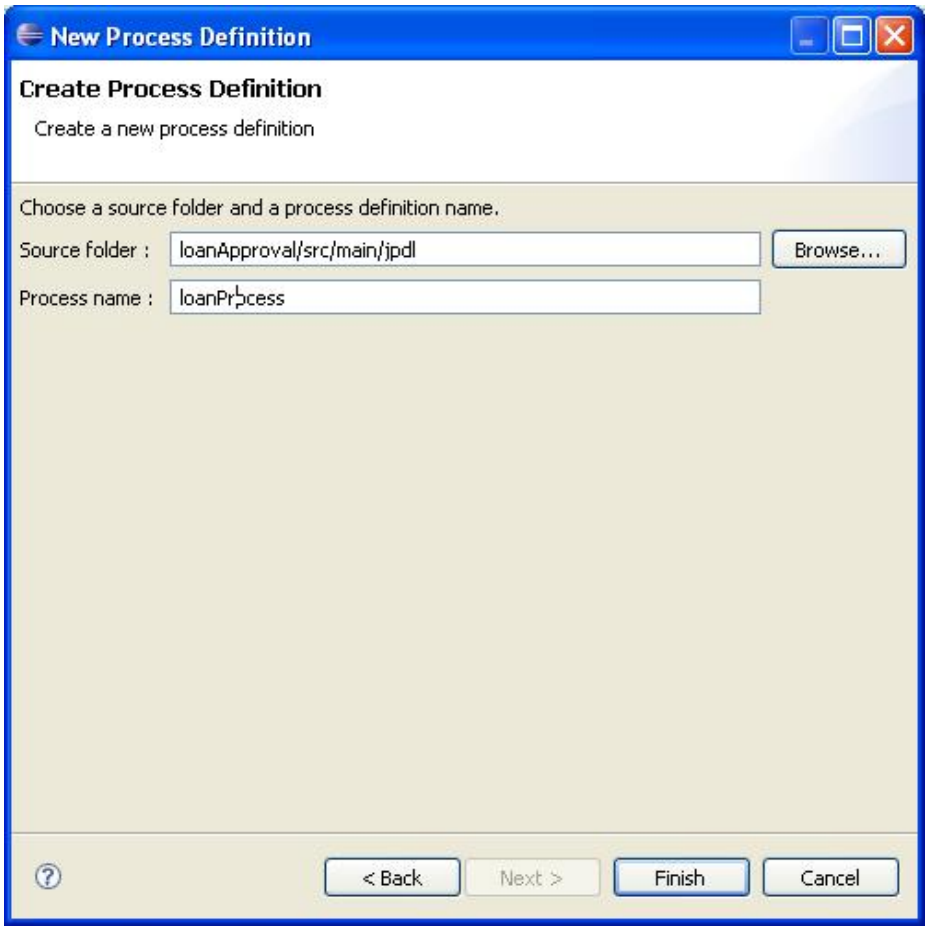
3.2. Creating a Process

A Process is where the business process scenario is modeled. A process can have multiple activities.

Steps	Description	Comments
Creating a new Process	1. In left of the editor pane (package explorer window) click on the ⊞ Sign associated with the created process project name. 	
Creating a new Process (Contd...)	1. In the left of the editor pane (navigator pane) select src\main\jpdl. 	
	2. Right click on it. Select new->Other	

Steps	Description	Comments
		
Creating a new Process (Contd ...)	3. Double click on jBPM. Select Process Definition. Click Next when done	

Steps	Description	Comments
		
	4. Give the name of the process.	

Steps	Description	Comments
		

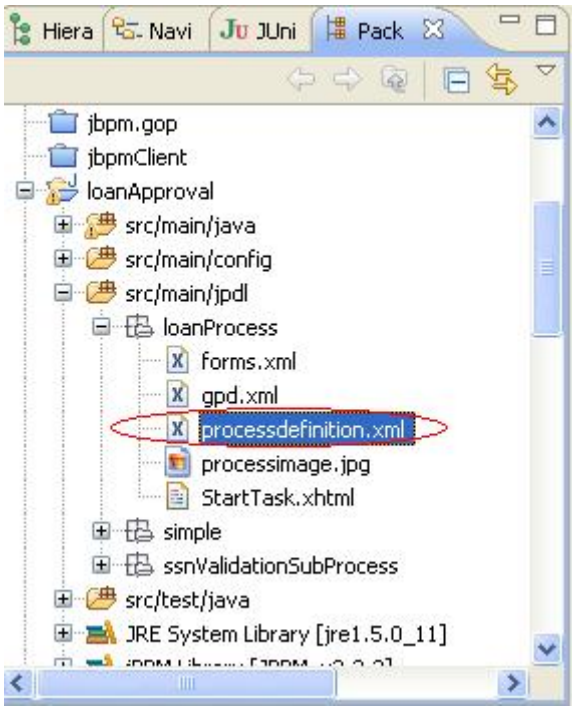
3.3. Creating Swimlane

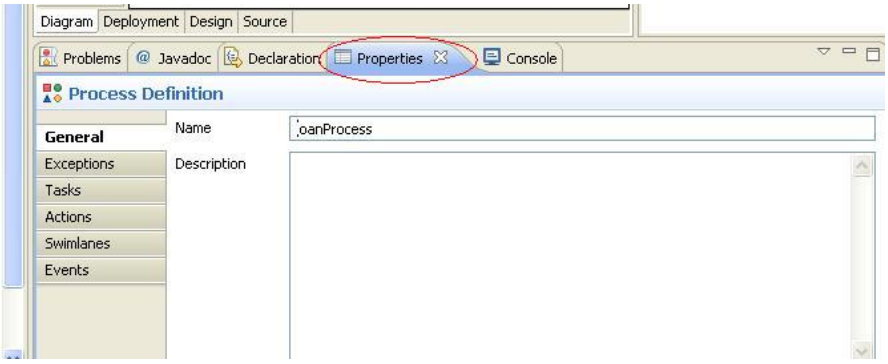
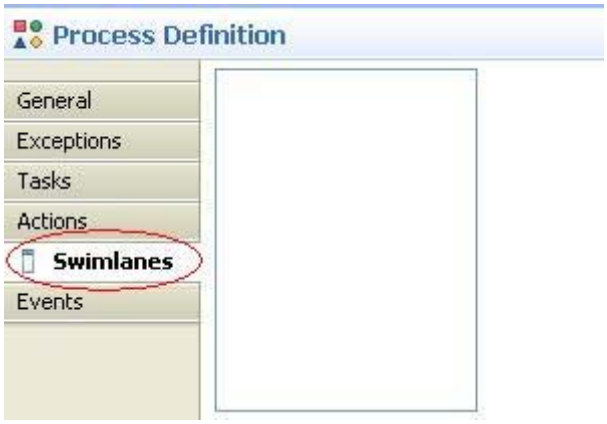
A swimlane defines a role function for a speicifc work being done in a process. Swimlane acts as role handlers for different types of activities.

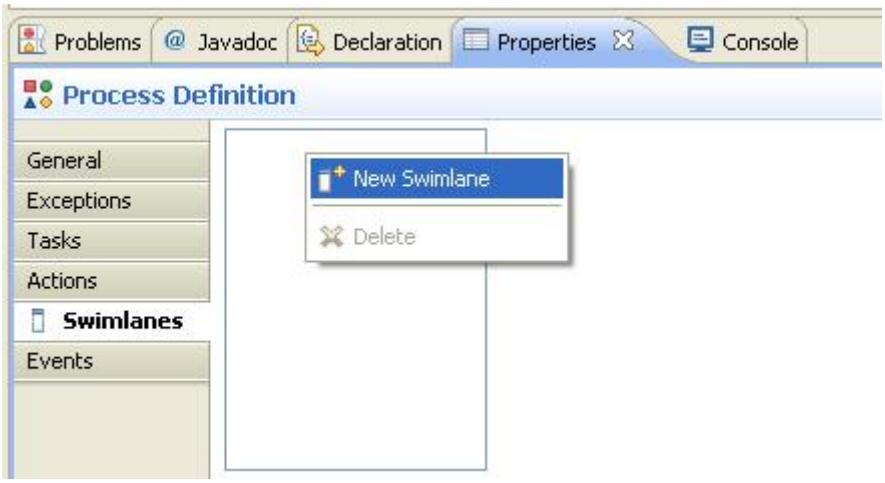
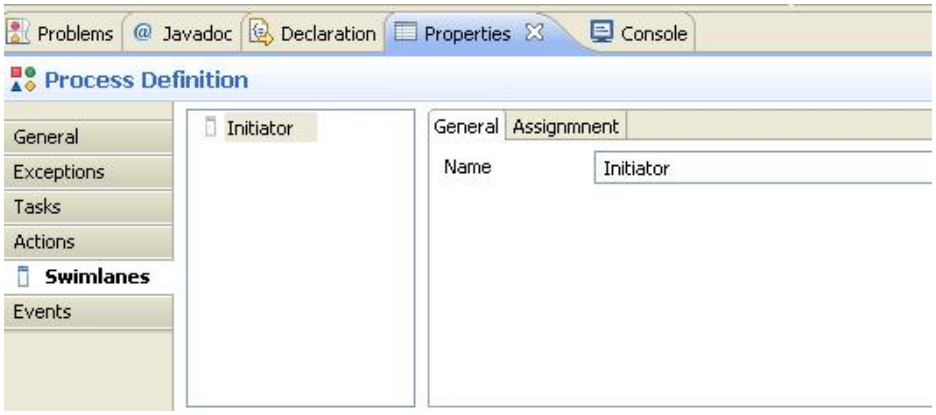
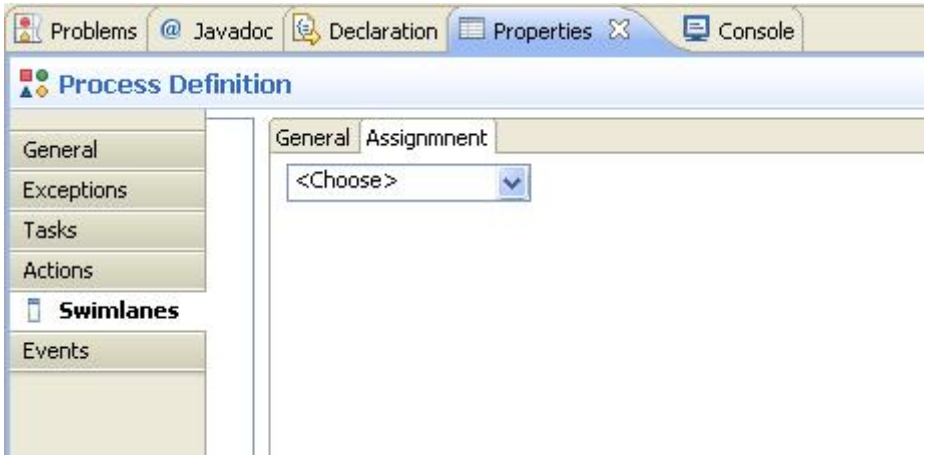
3.3.1. Swimlane with Actor-Assignment

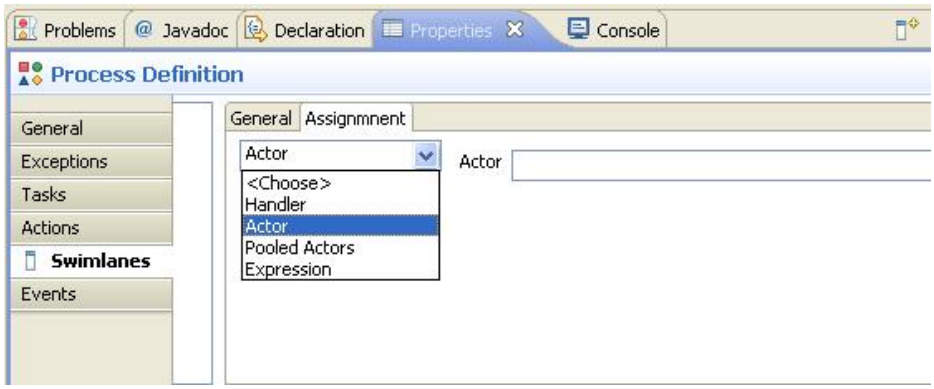
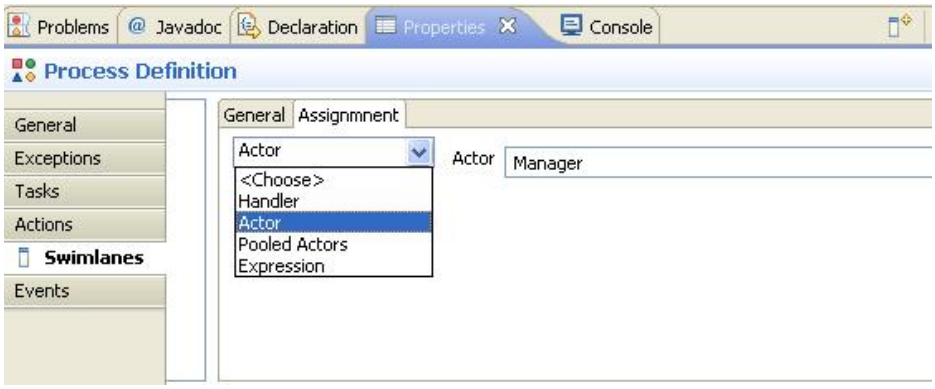
A swimlane corresponds to a specific user or a group of users. Each user belongs to a specific role and thus become categorized under different swimlane. A swimlane can be created specifying a single user or multiple users. It should be noted that the user should have a valid existence which may be found in jBPM_ID_USER table. (Refer to section 3.5.)

The following table describes the procedure to create Swimlane in jBPM using an Actor Assignment.

Steps	Description	Comments
	1. Select the processdefinition.xml of the selected process by clicking on the ⊞ sign and navigate to process-project name/ (src/main/jpdl)/process name in the project explorer window pane and double click to open it in the editor. 	

Steps	Description	Comments
Creating a new Swimlane	2. Click Properties tab. 	
	3. Click on the tab Swimlanes. 	
Creating a new Swimlane (Contd ...)	4. Right Click on the vertical white space area. 	

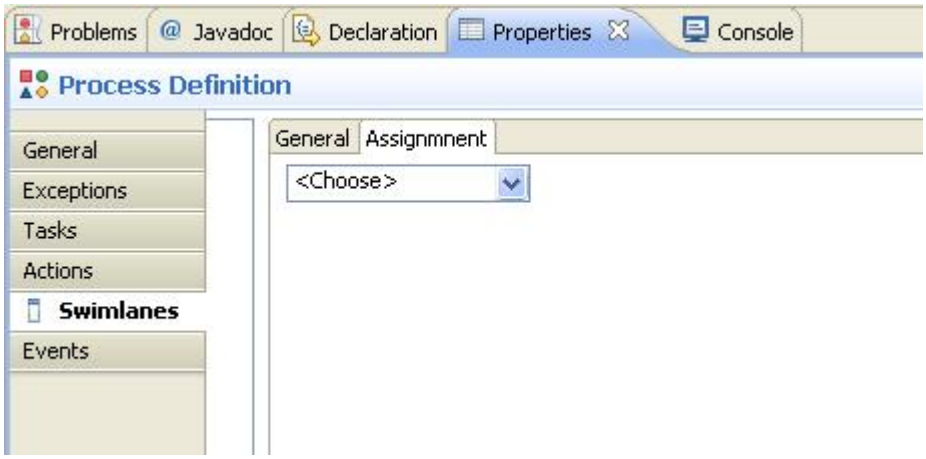
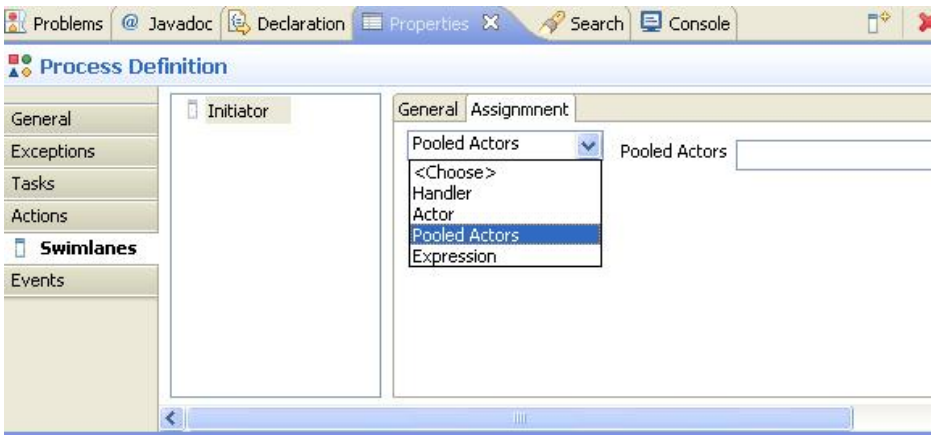
Steps	Description	Comments
	5. Choose New Swimlane. 	
	6. Give a suitable name to the Swimlane. 	
Assignment	7. Click on the Assignment tab. 	

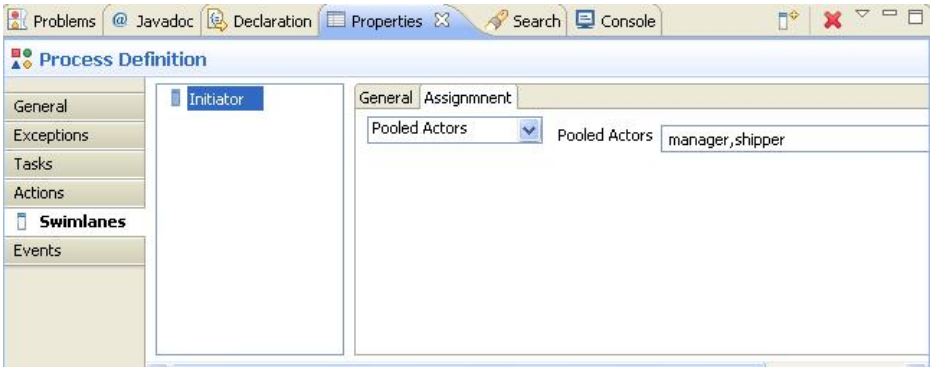
Steps	Description	Comments
	8. Choose an Assignment type as “Actor” from the drop down. 	
	9. Give a valid/existing actor or user name in Actor textbox. Here we give it as manger. 	jBPM by default offers 4 users ‘manager’,’admin’, ’shipper’,’user’ with the role ‘manager/admin/user’, ‘admin/user’,’user’ ,’user’ accordingly. User information can be obtained from jBPM_ID_USER table. (Refer to section 3.5.)

3.3.2. Swimlane using pooled actor

Pooled actor stands for a group of users. When a swimlane is assigned to a group of users - each user belonging to that group will be a part of the same swimlane.

The following table describes the procedure to create Swimlane in jBPM using pooled Actor.

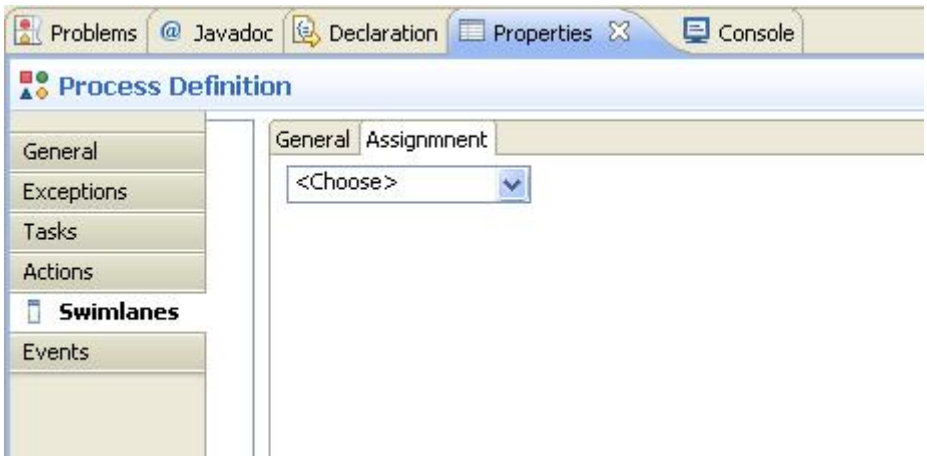
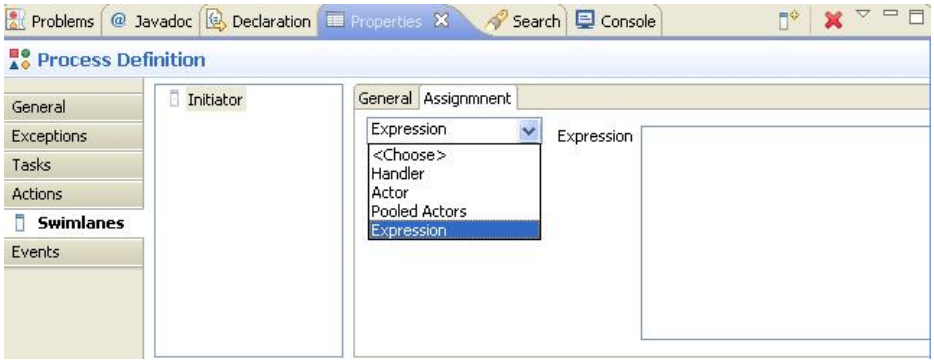
Steps	Description	Comments
Creating a new Swimlane	1. Create a new Swimlane.	Refer to *swimlane with actor assignment section (Sec-3.3.1)*
Assignment	2. Click on the Assignment tab. 	
	3. Choose an Assignment type as “Pooled Actors” from the drop down. 	

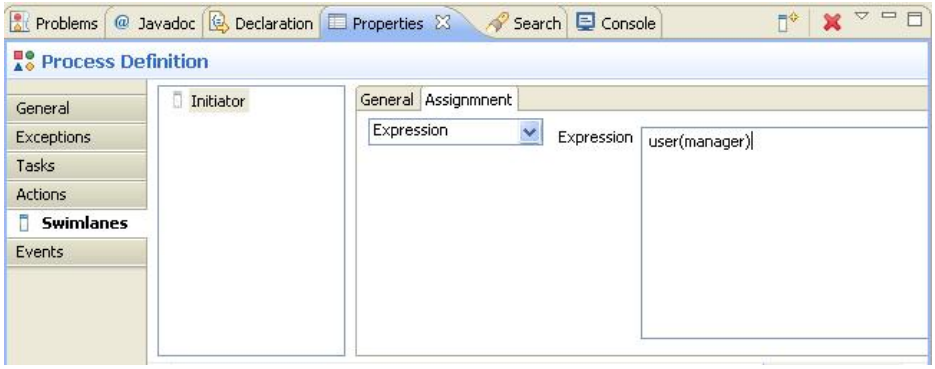
Steps	Description	Comments
	4. Give a valid/existing actor or user name in Pooled Actors textbox. Here we give it as manager. Optionally the actor name can be given as shipper,user and admin. 	jBPM by default offers 4 users 'manager','admin', 'shipper','user' with the role 'manager/admin/user', 'admin/user','user' ,'user' accordingly. User information can be obtained from jBPM_ID_USER table. (Refer to section 3.5.)

3.3.3. Swimlane using Expression

It is an assignment expression for the jBPM identity component. Management of users, groups and permissions is commonly known as identity management. The actors will be resolved from the expression.

The following depicts that how a swimlane can be associated with a user using expression.

Steps	Description	Comments
Creating a new Swimlane	1. Create a new Swimlane.	Refer to *swimlane with actor assignment section (Sec-3.3.1)*
Assignment	2. Click on the Assignment tab. 	
	3. Choose an Assignment type as “Expression” from the drop down. 	

Steps	Description	Comments
	4. Give a valid/existing actor or user name in the textbox using expression. Here we give it as Manager. Optionally the actor name can be given as Shipper, User and Admin (which is default user provided). 	jBPM by default offers 4 users 'manager','admin','shipper','user' with the role 'manager/admin/user', 'admin/user','user','user' accordingly. User information can be obtained from jBPM_ID_USER table. (Refer to section 3.5.)

3.3.4. Swimlane using Handler

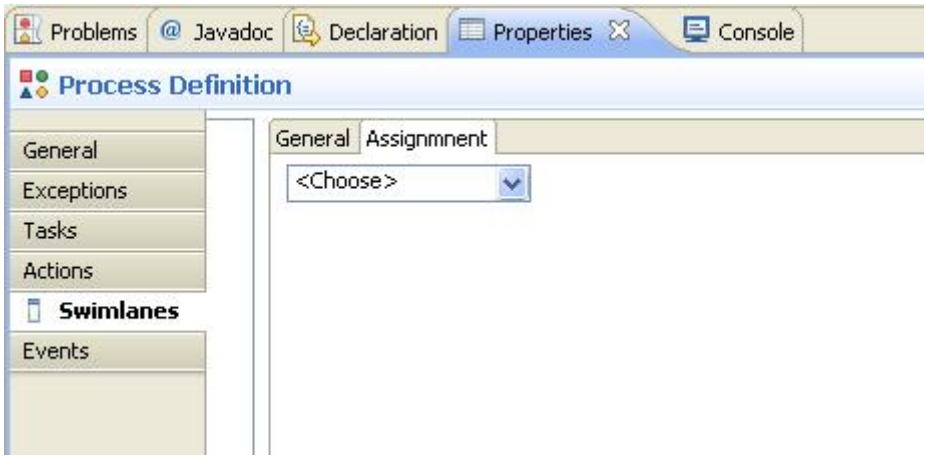
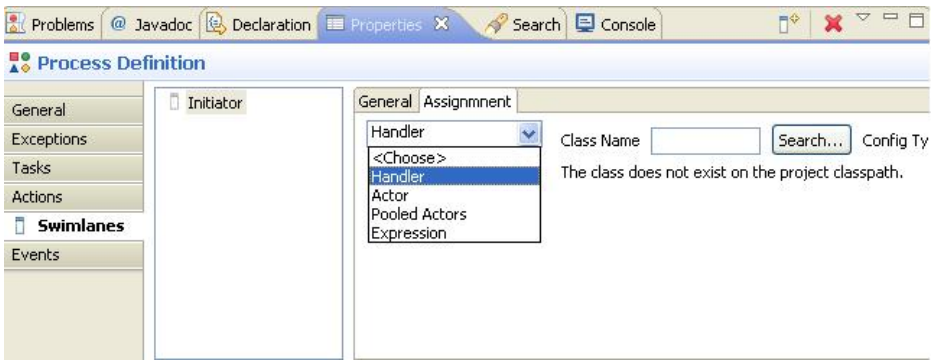
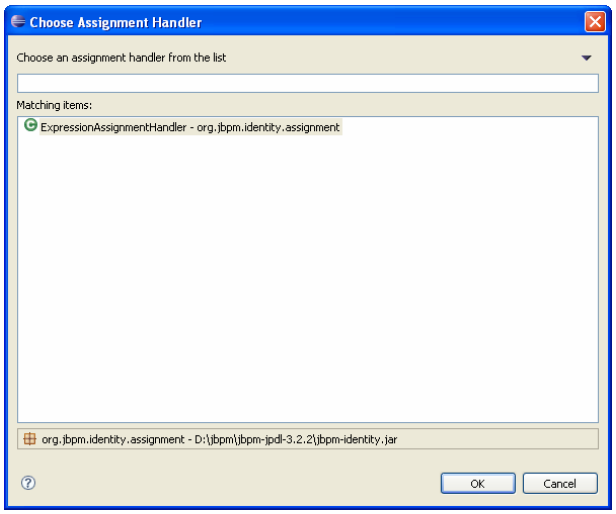
jBPM provides **org.jBPM.taskmgmt.def.AssignmentHandler** interface using which an user can be assigned to a swimlane. The mentioned interface consists of a method with the following signature that is responsible to assign a user to a swimlane.This approach is more relevant when an user is to associated with a swimlane at runtime.

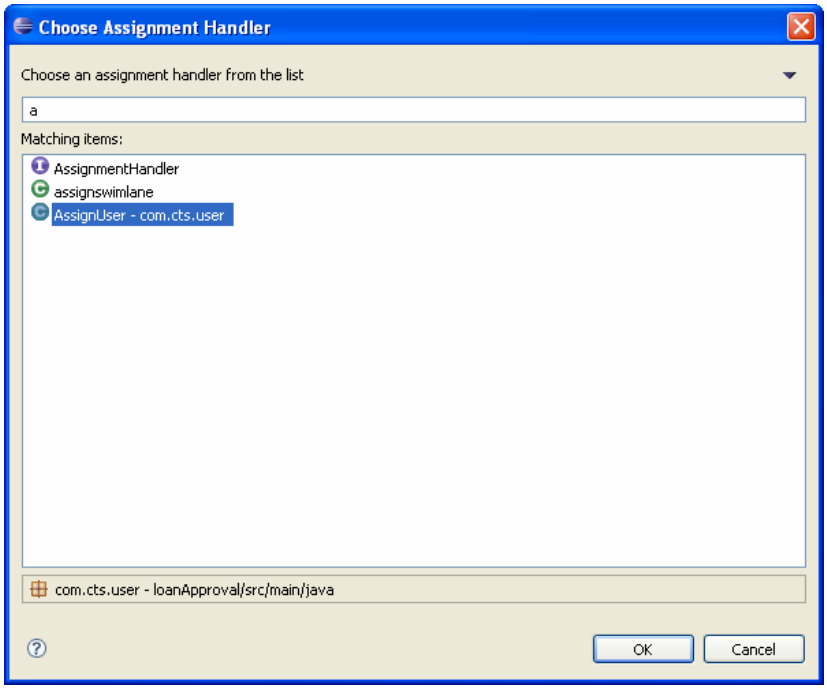
void *assign (Assignable assignable, ExecutionContext executionContext)* ***throws*** *Exception;*

The following table shows the procedure to create a swimlane using a handler

Steps	Description	Comments
Creating a new Swimlane and enter the following code snippet.	1. Create a new Swimlane.	Refer to *swimlane with actor assignment section (Sec-3.3.1)*

Create a class named *AssignUser*	Package com.cts.user;
	
	import org.jbpm.graph.exe.*;
	import org.jbpm.taskmgmt.def.*;
	import org.jbpm.taskmgmt.exe.Assignable;
	
	public class AssignUser implements AssignmentHandler {
	
	private static final long serialVersionUID = 1L;
	
	public void assign(Assignable assignable, ExecutionContext executionContext) {
	assignable.setActorId("manager");
	}
		This class is responsible to assign an actor 'manager' to a swimlane with which this class will be associated. The class has to implement the **AssignmentHandler** interface and implement the *assign* method.

Steps	Description	Comments
Assignment	2. Click on the Assignment tab. 	
	3. Choose an Assignment type as “Handler” from the drop down. 	
	4. Click on the search button to find the designated class. 	Swimlane handler implements *AssignmentHandler*.Therefore the screen will search only those classes which implements *Assignment Handler* interface. Here it is newly created *AssignUser* class.

Steps	Description	Comments
Assignment(Contd …)	5. Type the few letters of the designated class in the textbox. This will show the all possible class that one can associate with the swimlane. 	
	6. Click OK to finish.	

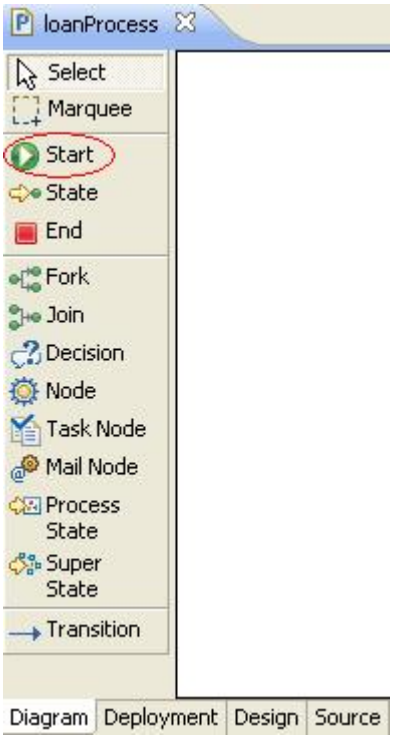
3.4. Creating Process Definition Entities

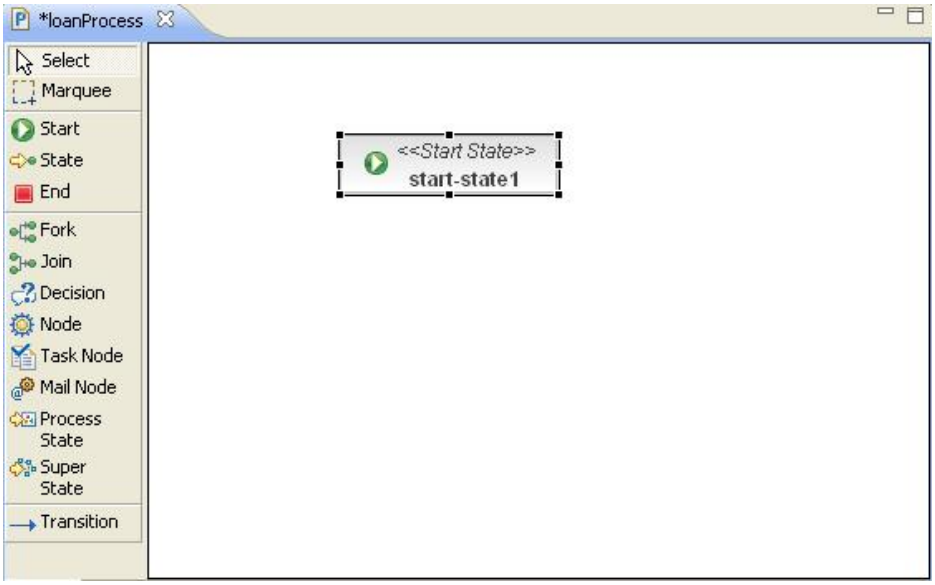
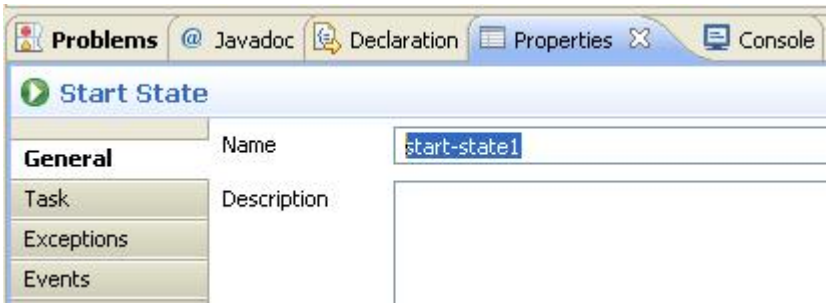
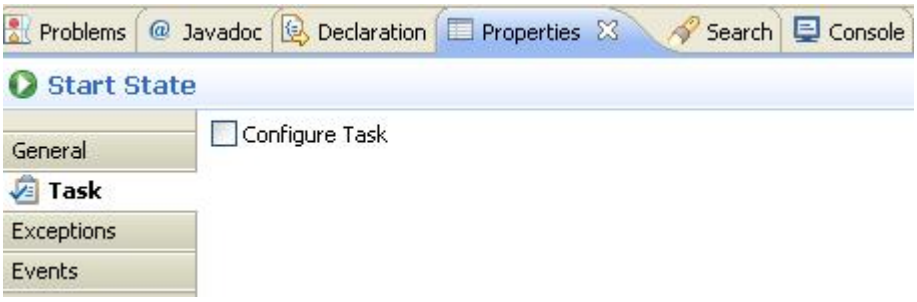
These entities define a manual or automated task that corresponds to a step within a process design. Adding a new entity allows one to create a new step and assign it to a Swimlane (optionally) within a process.

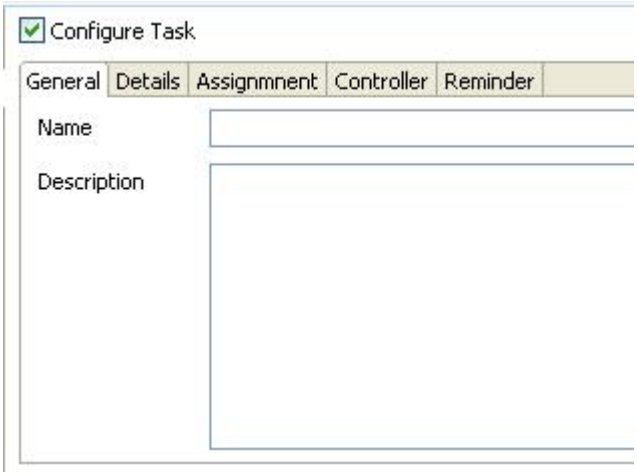
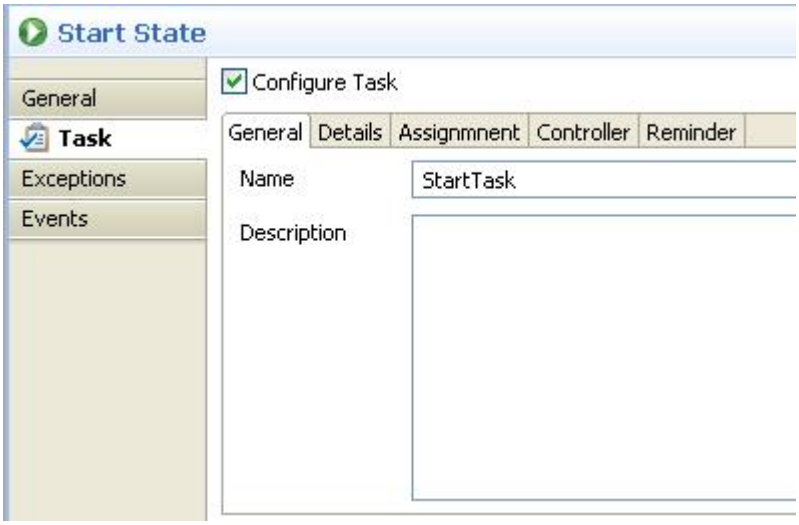
3.4.1. Creating a Start Node

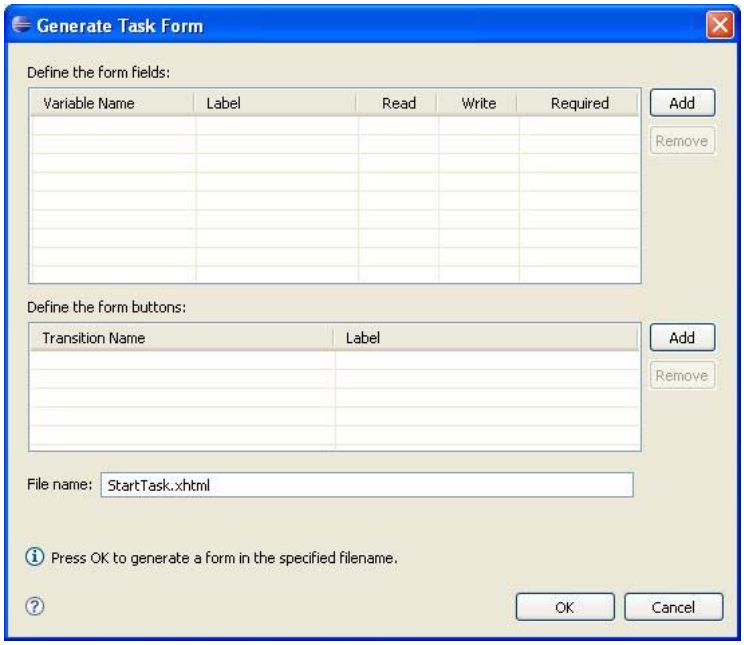
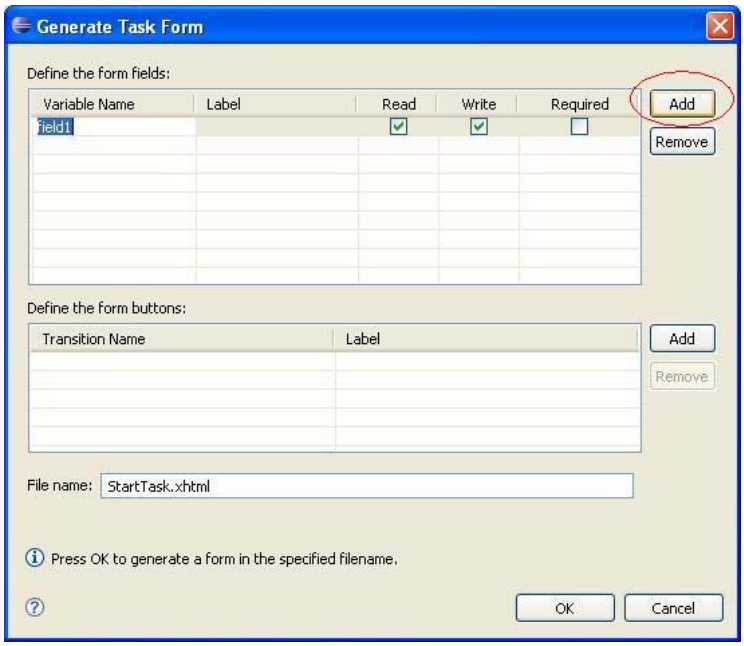
A start node is used to start a process. Without using a start node subsequent activities inside a process can't be performed. It is the entry point to a jBPM process.

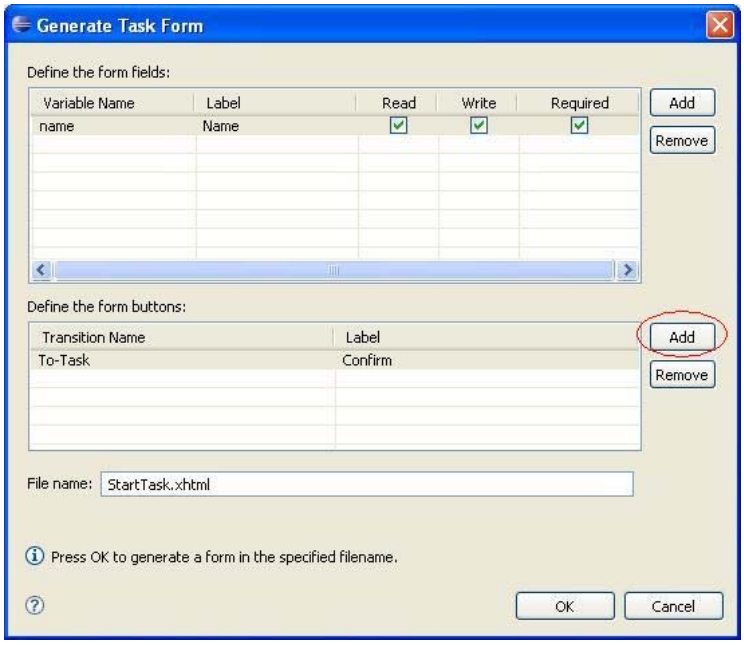
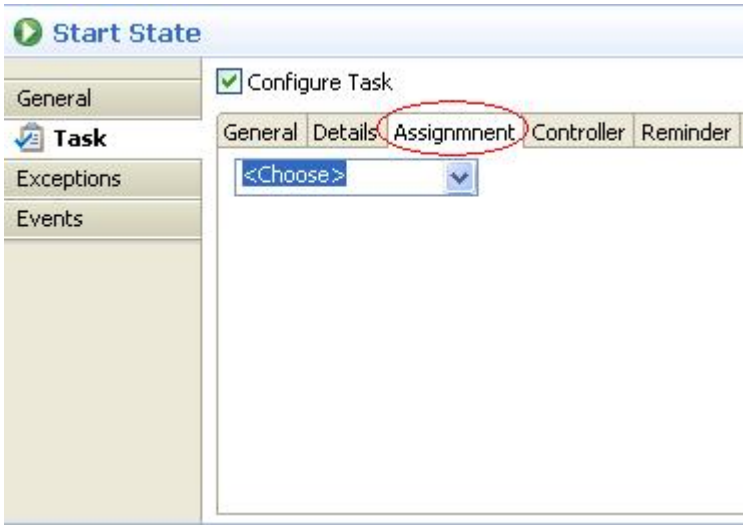
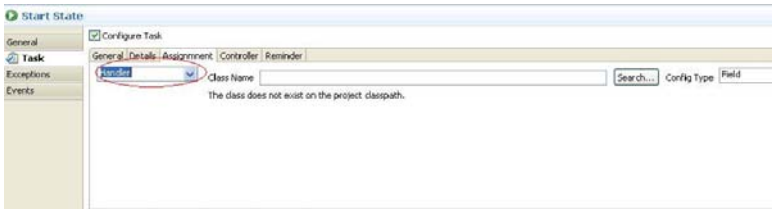
The following table describes how to include a start node in a workflow

Steps	Description	Comments
Creating a start node	1. Click on the Start node from the left window toolbar pane of the editor. 	
Creating a start node (Contd.)	2. Drop it on the design editor.	

Steps	Description	Comments
		
	3. Click on the properties tab below the design editor. 	Alternatively one can change the name of the Node and enter a brief description on that node from the 'General' tab.
Attribute-Task Configuration	4. Click on the task tab. 	If on the start of the process some manual work is required then a task can be associated with the 'Start' node.
Attribute-Task Configuration (Contd ...)	5. Check the Configure Task checkbox.	

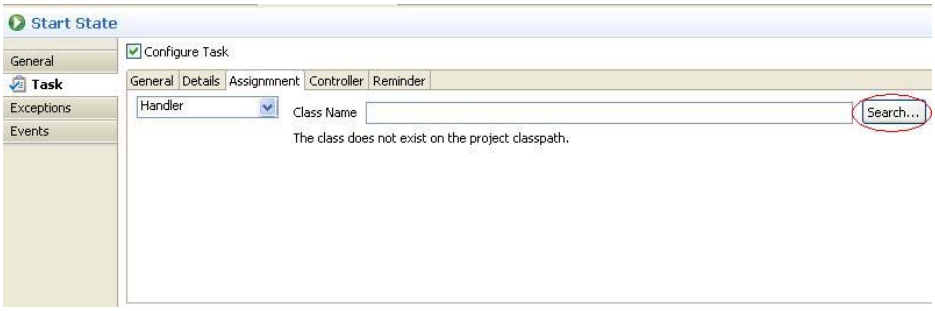
Steps	Description	Comments
		
	6. Enter the name of the task that will be associated with the Start-Node. Here it is given as StartTask as shown below. 	
Attribute-Task Configuration *Task form Generation*	7. Click on the Details tab. And Click on the Generate Form button. 	**Generate Form** option is responsible to generate a <nodename>.xhtml file which will be associated with the task during runtime and allow a user who will be working upon that task to enter certain field values.

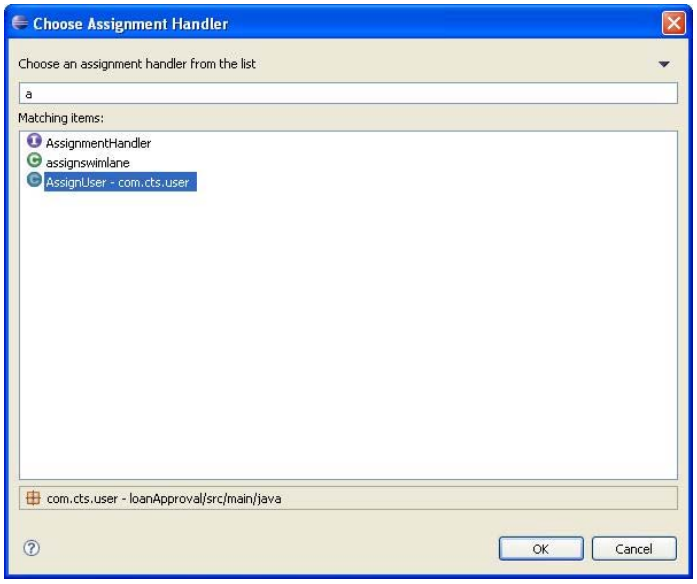
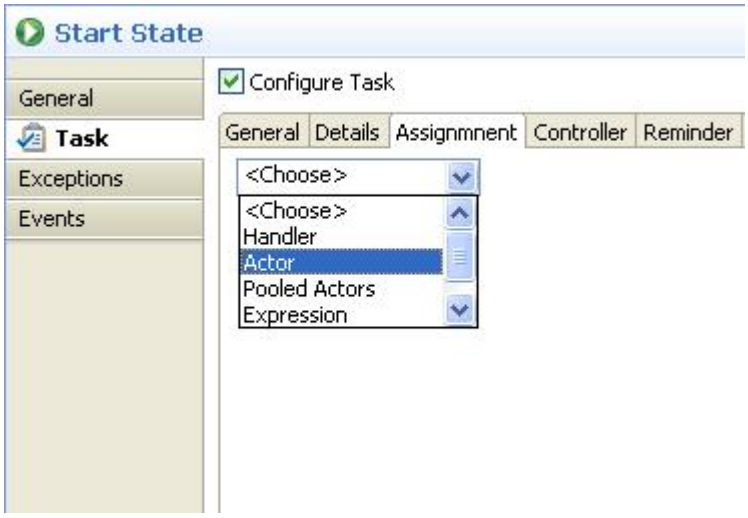
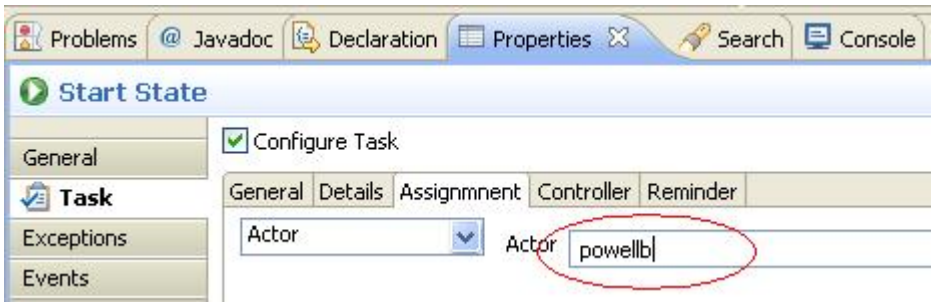
Steps	Description	Comments
Attribute-Task Configuration (Contd ...) *Task form Generation*		This wizard is responsible to create a task variable in the ContextInstance of the Process.
Attribute-Task Configuration (Contd ...) *Task form Generation*	8. Click on the add button to add a variable to the left. 	
Attribute-Task Configuration (Contd ...) *Task form Generation*	Give the name of the variable. Here it is 'name'. Press Enter. Enter the label. By default access permission of each variable entered in the entry is Read and Write. Additionally if 'Required' checkbox is selected then this variable becomes mandatory with the related task.	
Attribute-Task Configuration	9. Click on Add in Define the form buttons section. Enter the name of the transition on which the execution flow will take place. Enter the label of the button.	

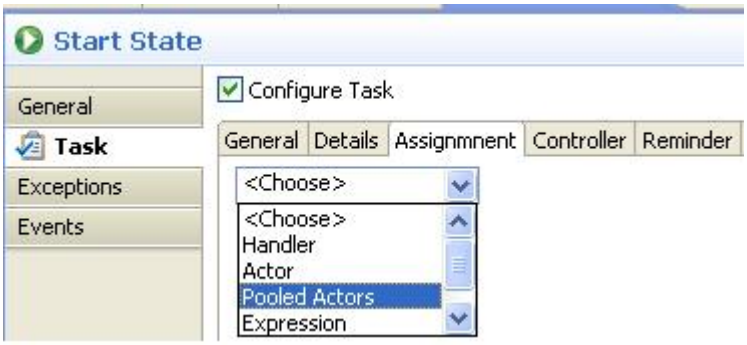
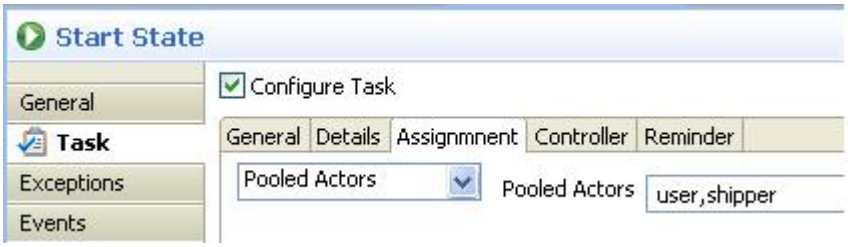
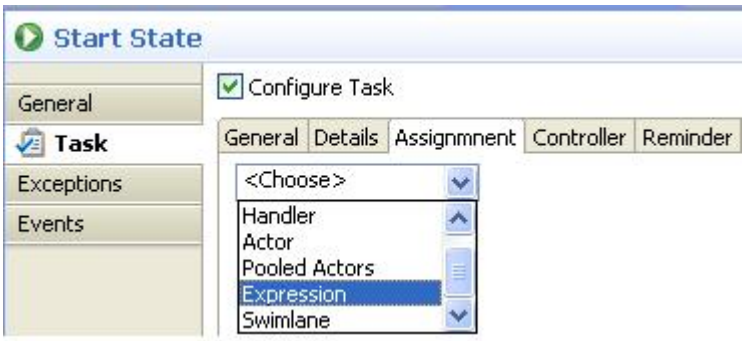
Steps	Description	Comments
(Contd ...) *Task form Generation*		
Attribute-Task Configuration (Contd ...) *Assignment Configuration*	10. Click on the Assignment tab. 	
Attribute-Task Configuration (Contd ...) *Assignment Configuration using a handler*	11. If runtime assignment is required as a business need then use 'Assignment Handler' option with the task created in previous steps. 	Appropriate assignment handler is used to define the user or a set of users who are capable of starting the task. A handler assignment expects a class which implements the **Assignment handler interface** of which assign method is responsible to set a

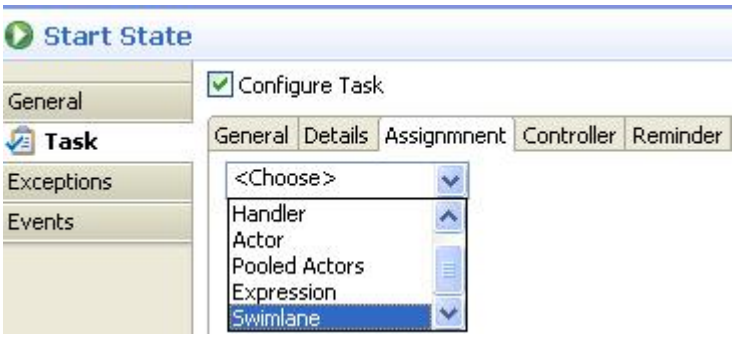
Steps	Description	Comments
		user or a group of user at runtime to work upon this task.

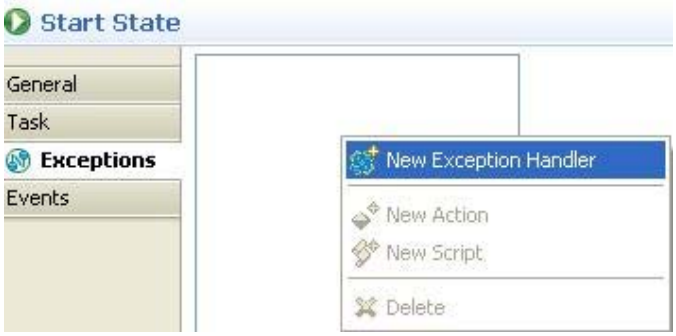
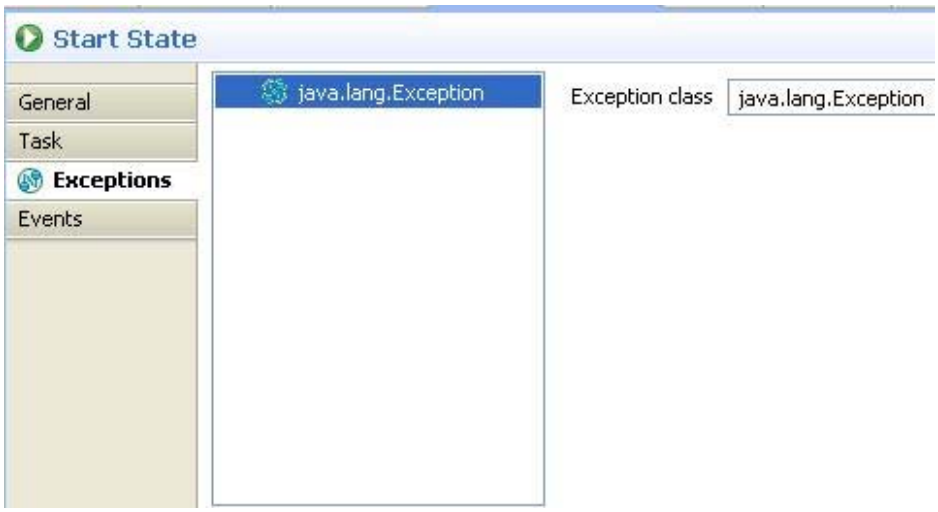
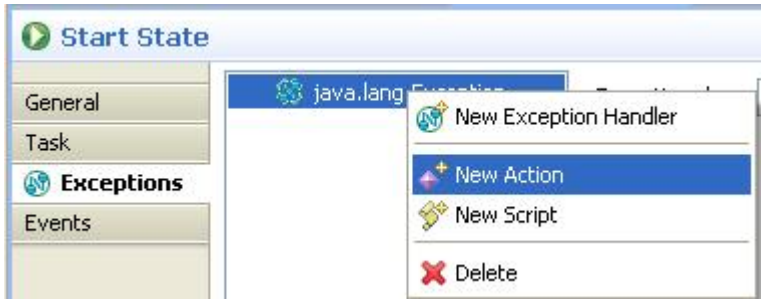
Attribute-Task Configuration (Contd ...) *Assignment Configuration using a handler*	12. Create a java class AssignUser.
	
	`import org.jbpm.graph.exe.*;`
	`import org.jbpm.taskmgmt.def.*;`
	`import org.jbpm.taskmgmt.exe.Assignable;`
	
	`public class AssignUser implements AssignmentHandler {`
	
	`  private static final long serialVersionUID = 1L;`
	
	`  public void assign(Assignable assignable, ExecutionContext executionContext) {`
	`    assignable.setActorId("powellb");`
	`  }`
	`}`	

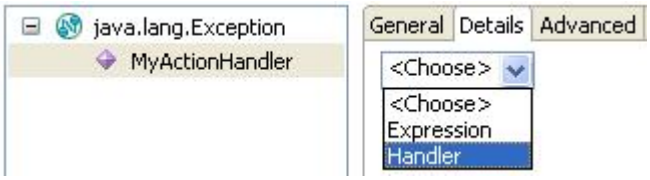
Attribute-Task Configuration (Contd ...) *Assignment Configuration using a handler*	13. Click on the Search button to find the assignment handler class. 	
Attribute-Task Configuration (Contd ...) *Assignment Configuration using a handler*	14. Choose the assignment handler class from the wizard. Click OK when done.	

Steps	Description	Comments
		
Attribute-Task Configuration (Contd ...) *Assignment Configuration using an Actor*	15. If at design time the user is to be associated with the task choose 'Actor' from the dropdown. 	Actor is a valid user. A set of valid user is available in jBPM_ID_USER table.
Attribute-Task Configuration (Contd ...) *Assignment Configuration using an Actor*	16. Enter the name of the Actor (user). 	

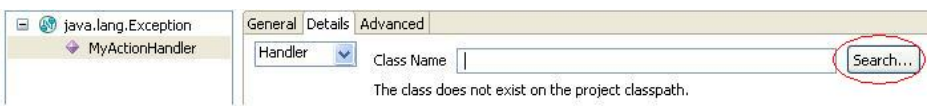
Steps	Description	Comments
Attribute-Task Configuration (Contd ...) *Assignment Configuration using an Pooled Actors*	17. If a set of actors is to work upon the task then choose 'Pooled Actors' from the dropdown under assignment tab. 	A pooled actors means a set of valid actors
Attribute-Task Configuration (Contd ...) *Assignment Configuration using an Pooled Actors*	18. Give a set of valid user name in a comma-separated expression in Pooled Actors textbox. 	
Attribute-Task Configuration (Contd ...) *Assignment Configuration using expression*	19. Alternatively one can also use choose 'Expression' to associate a single user(actor) from the dropdown. 	
Attribute-Task Configuration (Contd ...) *Assignment Configuration using expression*	20. Enter the expression text box to fill the expression like user (<valid user name>).  NB:-Expression syntax is like the following syntax : `first-term --> next-term --> next-term -->`	

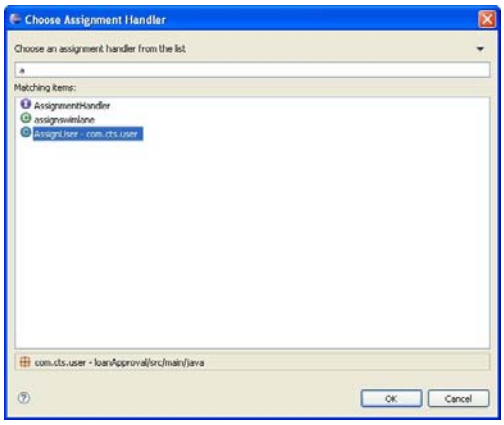
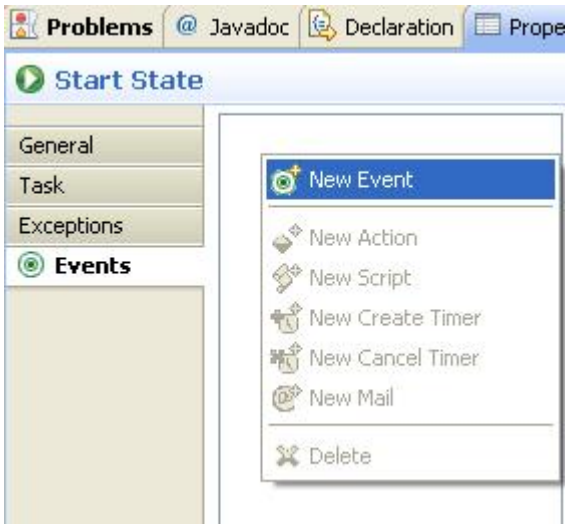
Steps	Description	Comments
	* first-term ::= previous | * swimlane(swimlane-name) | * variable(variable-name) | * user(user-name) | * group(group-name) * next-term ::= group(group-type) | * member(role-name)	
Attribute-Task Configuration (Contd ...) *Assignment Configuration using swimlane*	21. If a swimlane is previously configured then one can choose 'Swimlane' from the dropdown and allocate the task under this swimlane. 	
Attribute-Task Configuration (Contd ...) *Assignment Configuration using swimlane*	22. Enter the name of the swimlane into the textbox. Name is case-sensitive. 	*Refer to section-3.3 to know how to create a swimlane.*
Attribute-Exception handling	23. To attach some exception handling policy click on the 'Exceptions' tab under properties tab. 	

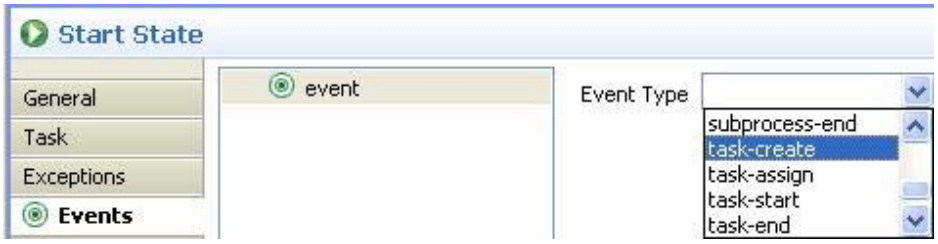
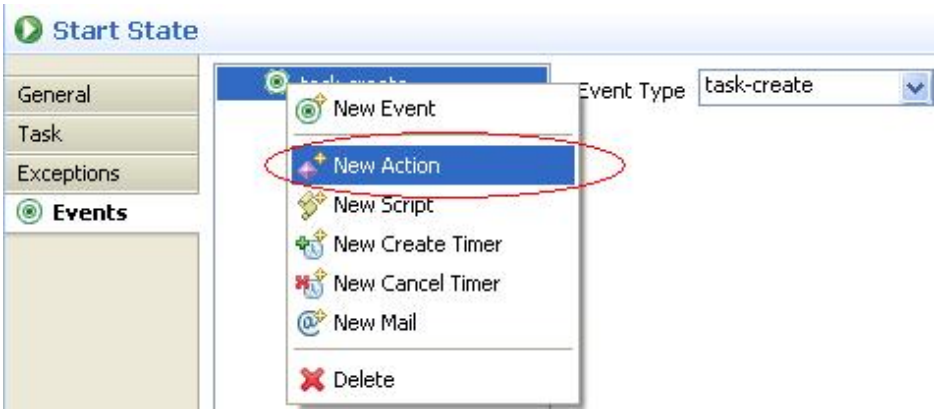
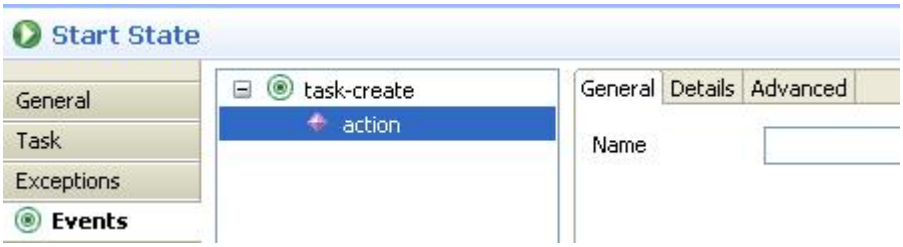
Steps	Description	Comments
Attribute-Exception handling (Contd.)	24. Right Click on the blank vertical box. Click new Exception Handler. 	
Attribute-Exception handling (Contd.)	25. Give the name of an exception class. Here it is **java.lang.Exception** class. The exception handler name is populated with the exception class name specified in the text box. 	An exception class may be any standard exception. Here for each exception one has to associate an action so that an action class will be called when specified exception occurs during the task execution and might be some corrective operations can be taken in the action class. For custom exception class to be declared one has to create an exception class extending java.lang.Exception class.
Attribute-Exception handling (Contd.) *using Action*	26. Under the newly created exception handler right click on it and click on New Action. 	
Attribute-Exception handling (Contd.) *using Action*	27. On creating a new action a separate wizard will open which will ask for a name of the action.	This action will get triggered once an exception of type java.lang.Exception is thrown from a

Steps	Description	Comments
		particular activity.
Attribute-Exception handling (Contd.) *using Action*	28. Go to details tab to associate a class with this action. Separately an expression also can be attached with this action. 	
Attribute-Exception handling (Contd.) *using Action*	29. Choose the appropriate action details. Either a handler or an expression. A handler is nothing but an action class. Here we associate using an action class. 	

Attribute-Exception handling (Contd.) *using Action*	30. Create a Java class ExceptionAction
	`import org.jbpm.JbpmConfiguration;`
	`import org.jbpm.graph.def.ActionHandler;`
	`import org.jbpm.graph.exe.ExecutionContext;`
	`public class ExceptionAction implements ActionHandler`
	`{`
	`public void execute(ExecutionContext executionContext)`
	`throws Exception {`
	`System.out.println("*******EXCEPTION IS CAUGHT*********");`
	`executionContext.getProcessInstance().getRootToken().signal("to end");`
	`} }`
	`}`
		Refer to Section *3.4.3* on *Creating Transition* at Step 5. If the associated action class in the start-state throws an exception of type java.lang.Exception then the following *ExceptionAction* will be executed. Execution flow will take place in the transition named to end. In case of a multiple transitions developer will have a choice to direct the execution flow in whichever transition the business needs drives.

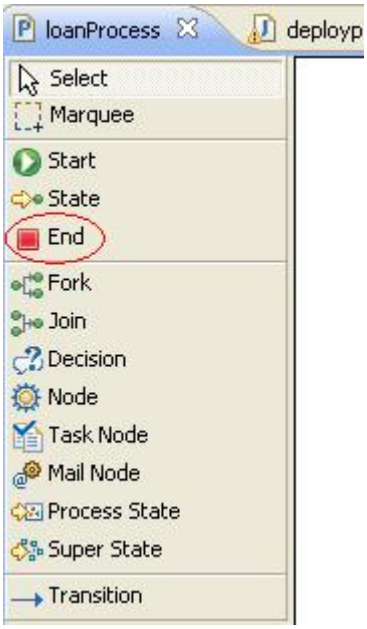
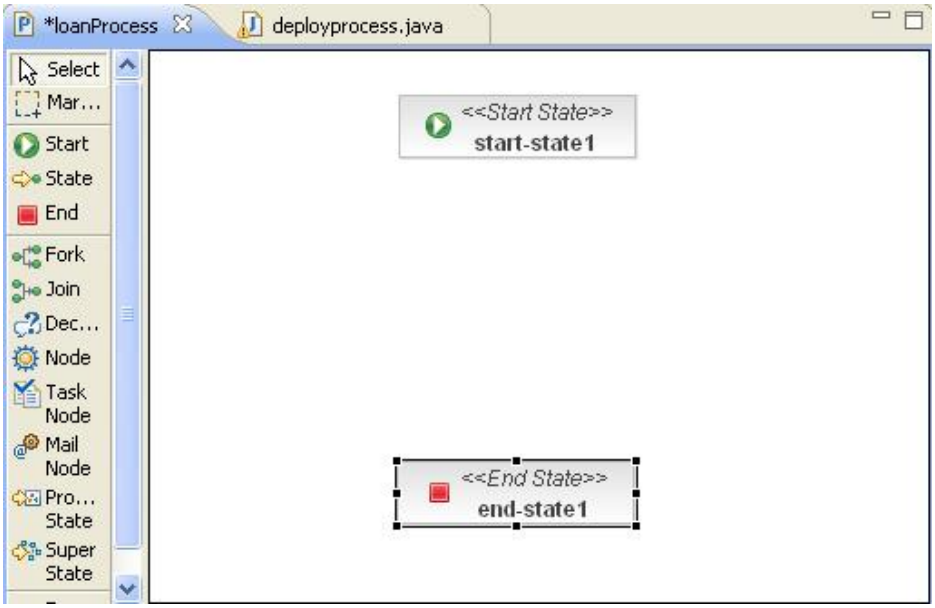
Attribute-Exception handling (Contd.) *using Action*	31. Click on the search button to get the action handler class. 	

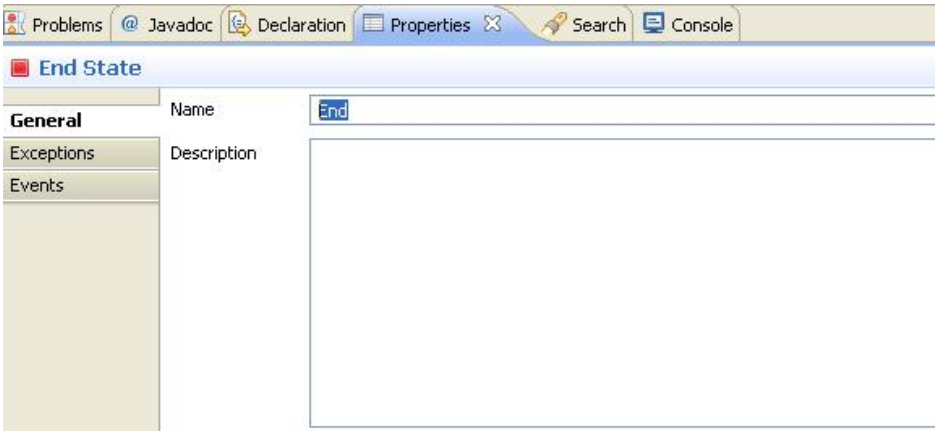
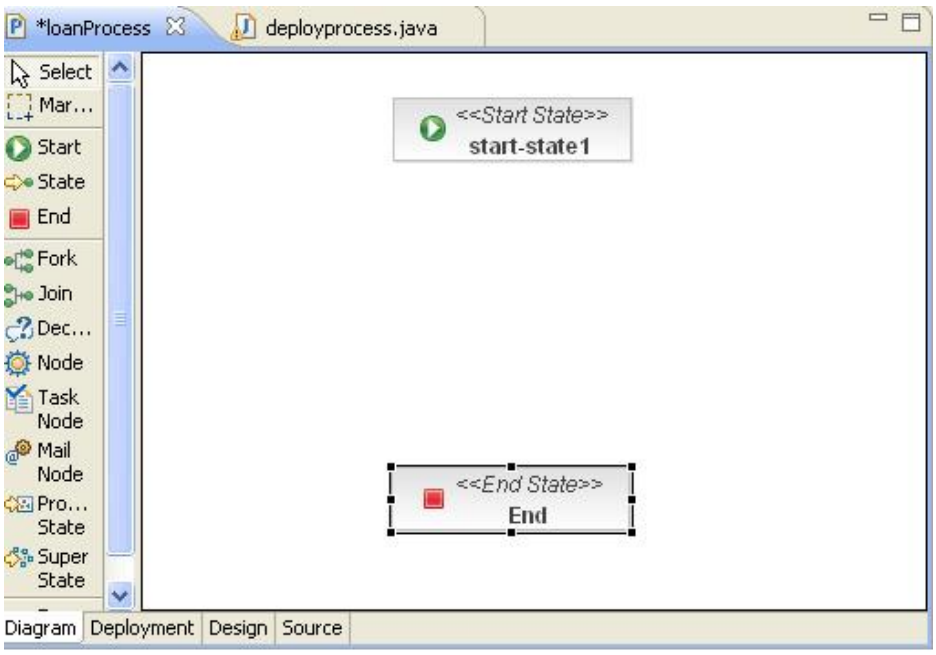
Steps	Description	Comments
Attribute-Exception handling (Contd.) *using Action*	32. Choose the action handler class from the wizard. Click OK when done. 	
Attribute-Events	33. If the task is to be based on certain event occurrences then click on the Events tab. 	
Attribute-Events(Contd..) *using Actions*	34. Right Click on the right blank area and click New Event. 	Every node is state and each node is associated with an event.
Attribute-Events(Contd..) *using*	35. Choose appropriate Event Type from the dropdown that might occur during the execution of a start node. Here task-create event type has been chosen.	Since a task has already being associated with the

Steps	Description	Comments
Actions		start-node hence this event will get triggered once the process is started.
Attribute-Events(Contd.) *using Actions*	36. An event with name of an event-type will be generated. 	
Attribute-Events(Contd.) *using Actions*	37. Right click on the generated event. Select New Action. 	
Attribute-Events(Contd.) *using Actions*	38. An action will be generated in the right pane as the figure shows. 	
Attribute-Events(Contd.) *using Actions*	39. Configure the action.	*Refer to section 3.4.3 Creating a transition with attribute Action to know how to configure action with an event.*

3.4.2. Creating a End Node

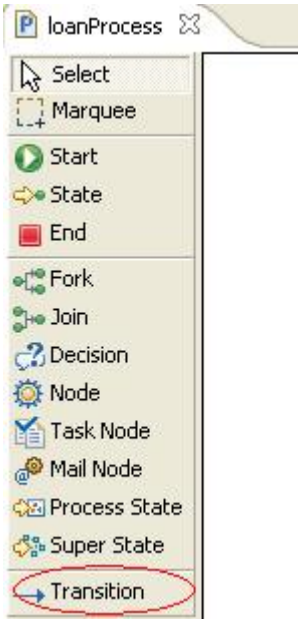
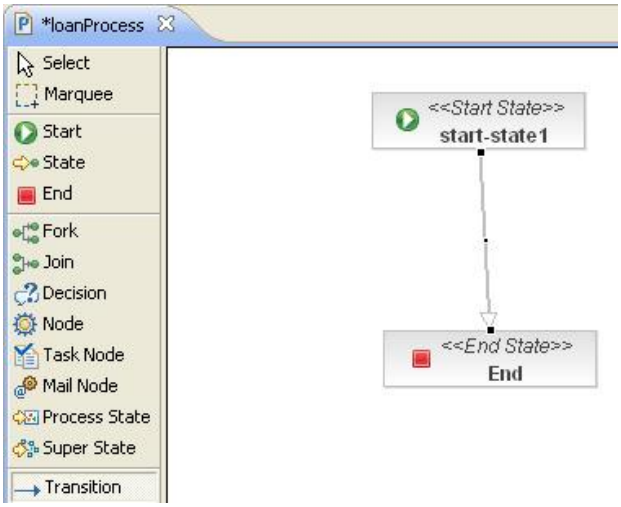
An End node is the exit point of the process.

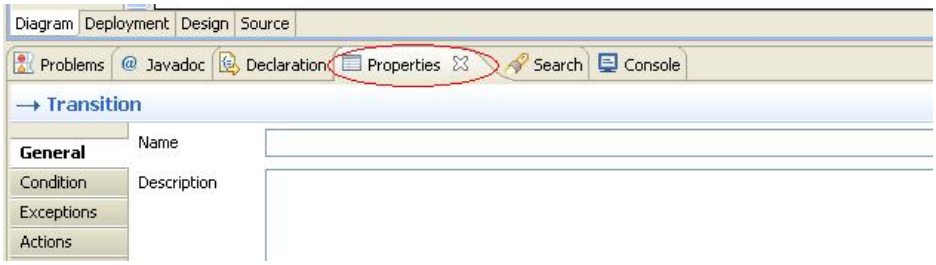
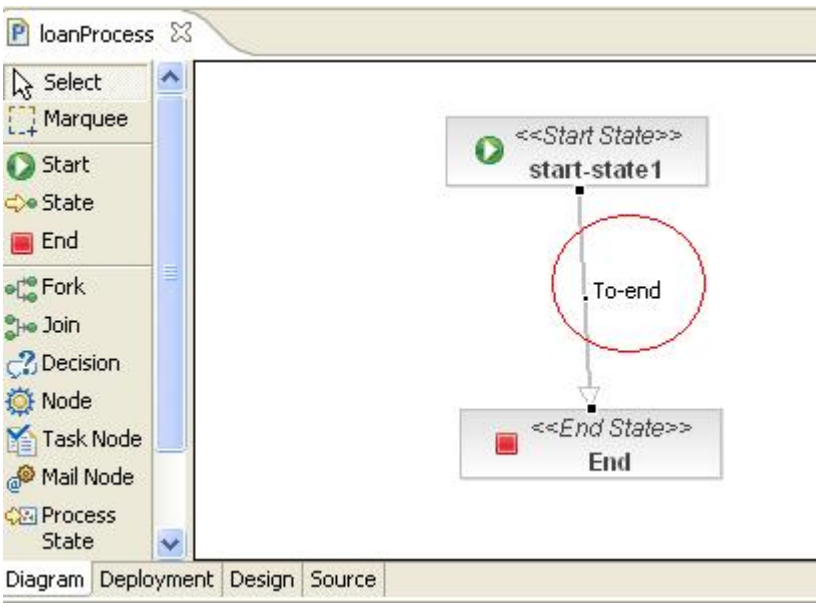
Steps	Description	Comments
Creating an End node	1. Click on the End node from the left window toolbar pane of the editor. 	
Creating a End node (Contd.)	2. Drop it on the design editor. 	
	3. Click on the properties tab below the design editor.	

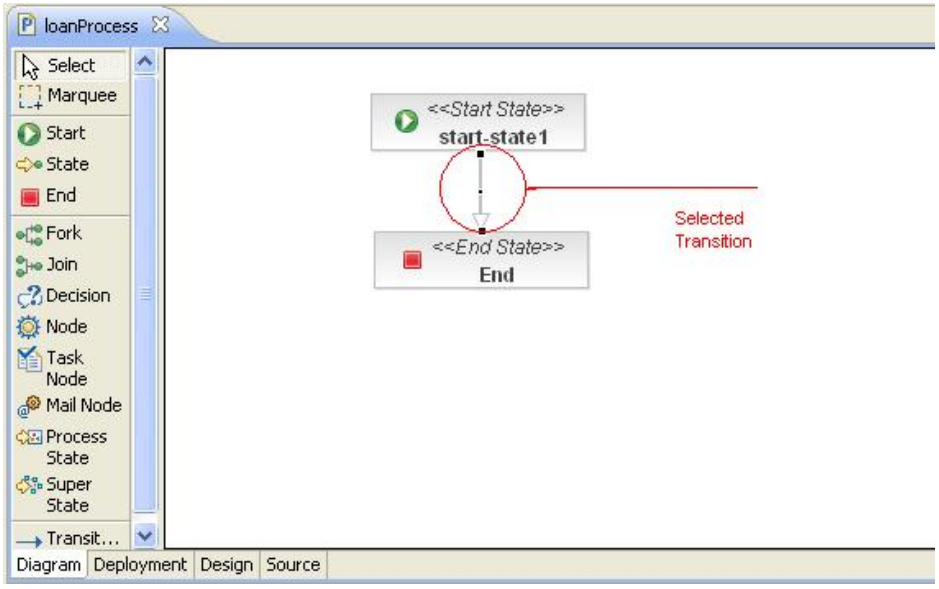
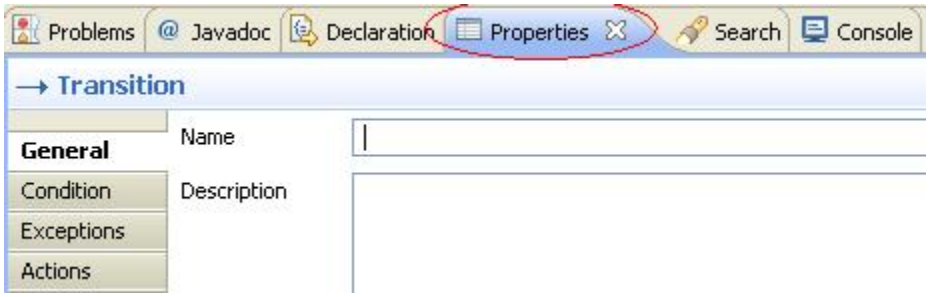
Steps	Description	Comments
		
	4. Rename your end node if required. 	
	5. The Changes will be reflected in the design editor. 	

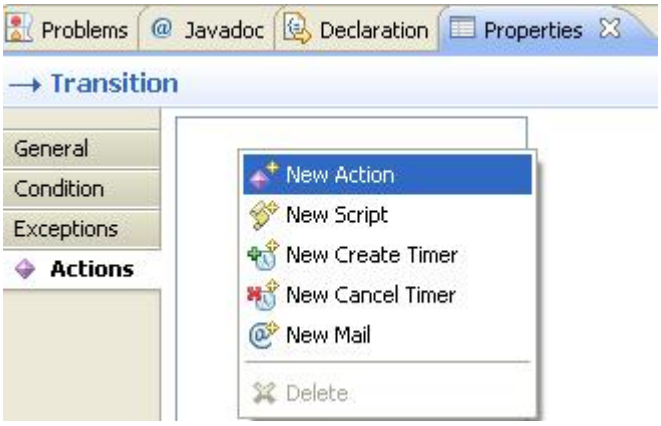
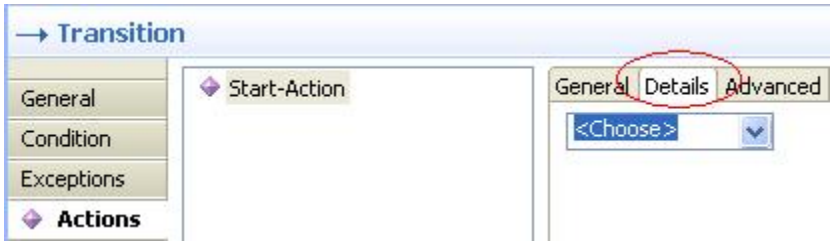
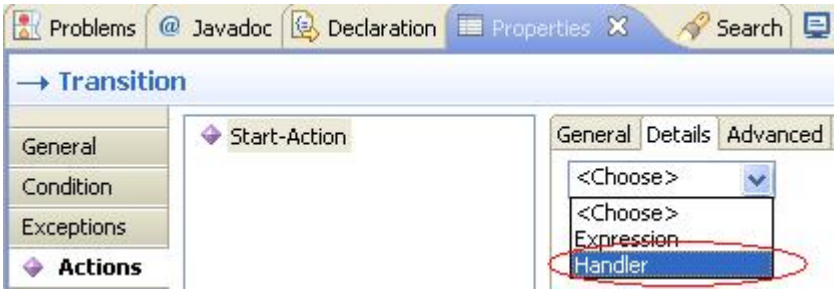
3.4.3. Creating Transition

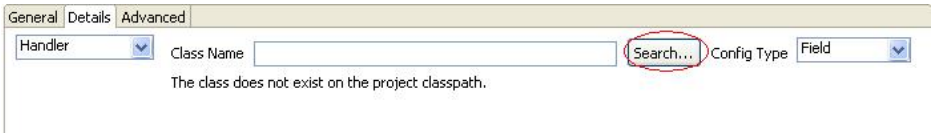
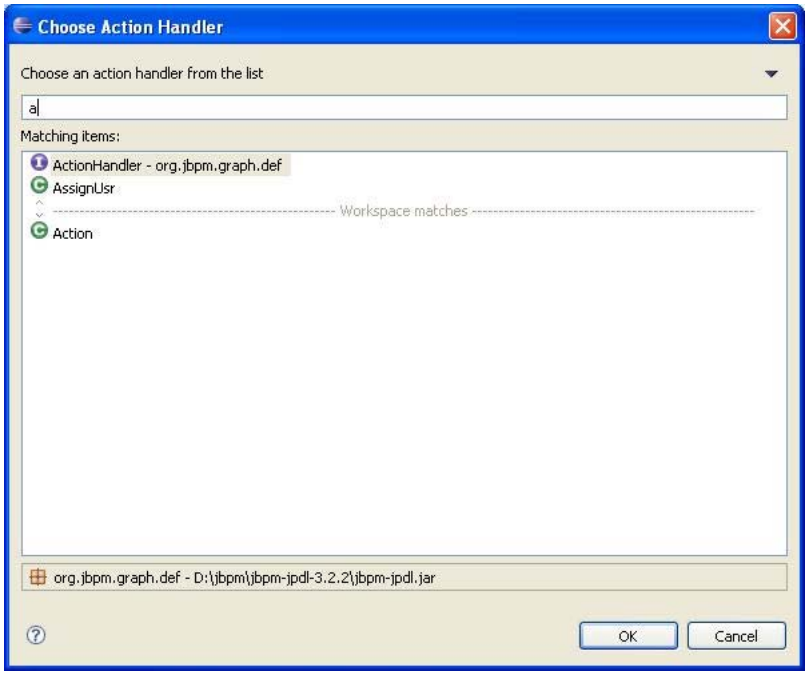
Transitions have a source node and a destination node. A transition is responsible for traversing an execution token from one node to the other during a process flow. Transitions are, therefore, very important in the context of process execution flow.

Steps	Description	Comments
Creating a transition	1. Click on the transition from the left window toolbar pane of the editor. 	
Creating a transition (Contd.)	2. Select the source node. Drag up to the destination node. 	
Creating a transition (Contd.)	3. Click on the properties tab below the design editor.	Transition properties gives additional scope to the developer

Steps	Description	Comments
		to associate action-class, exception handler, type of transition such as condition/unconditional etc.
Creating a transition (Contd.)	4. Default transition doesn't have any name. A name can be provided in the General tab under 'Name' textbox section. 	
	5. The Changes will be reflected in the design editor. 	
Attribute-Exception handling	*6. Refer to section-3.4.1 under Attribute-Exception handling steps.*	.
Attribute-Action-Handling	7. Select the transition from the Process Design Area.	

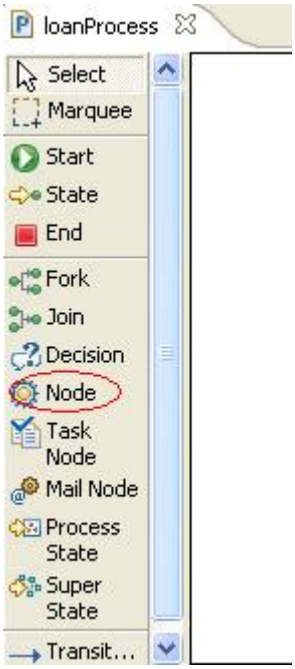
Steps	Description	Comments
		
Attribute-Action-Handling	8. Select the property tab below the Process design editor. 	
Attribute-Action-Handling (Contd ...)	9. Select the Actions tab to add action. 	
Attribute-Action-Handling (Contd ...)	10. Right click on the right blank area and click on New Action.	

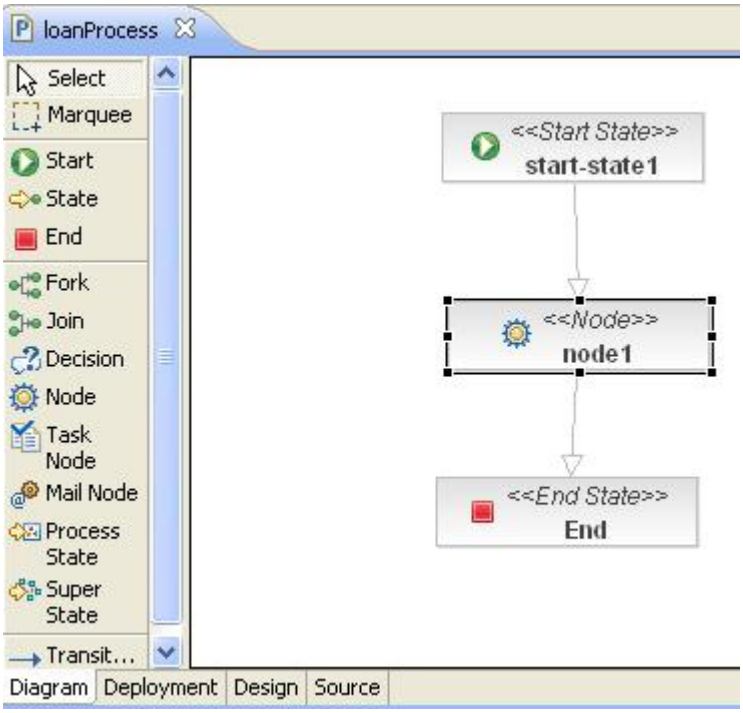
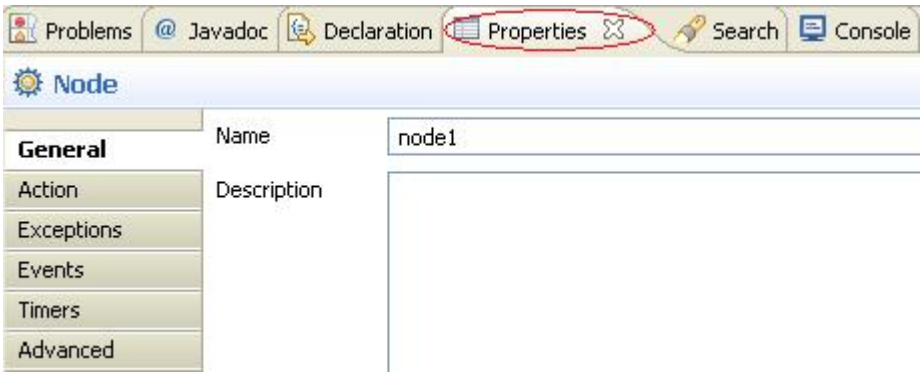
Steps	Description	Comments
		
Attribute-Action-Handling (Contd ...)	11. Give the name of the action as follows. 	
Attribute-Action-Handling (Contd ...)	12. Click on details tab as depicted. 	
Attribute-Action-Handling (Contd ...)	13. Choose the appropriate Action either as handler or Expression. Here the chosen one is a Handler. 	
Attribute-Action-Handling (Contd ...)	14. Choose the appropriate action handler class that one has to create before this step. To choose the handler (Java class) click on the Search button.	Note that an action handler will be a java class which has to implement

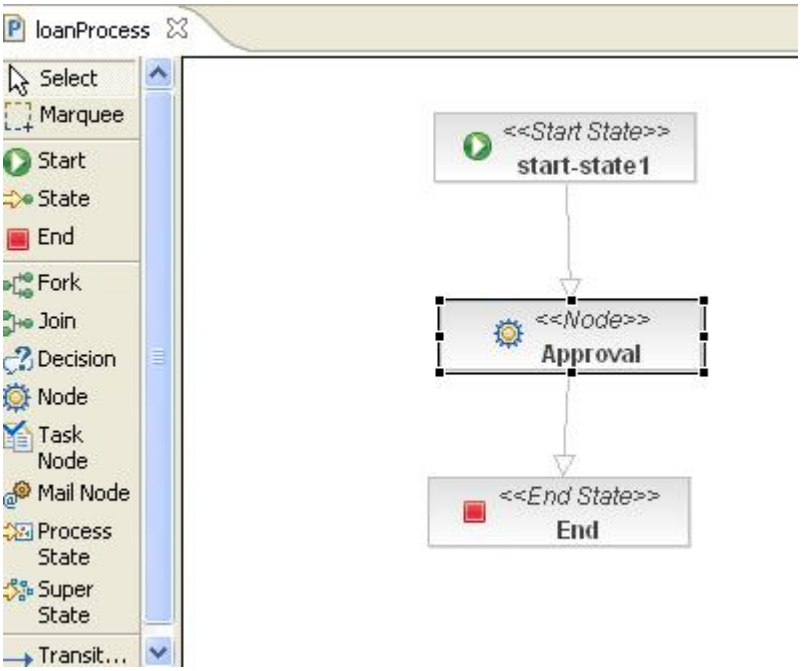
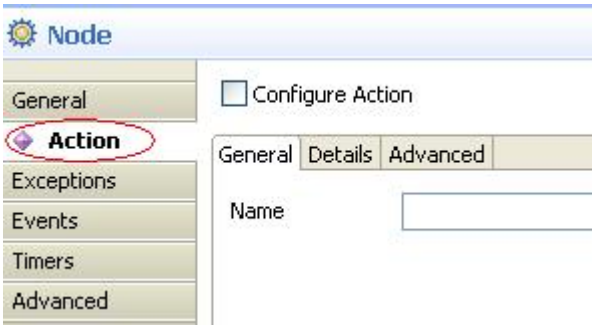
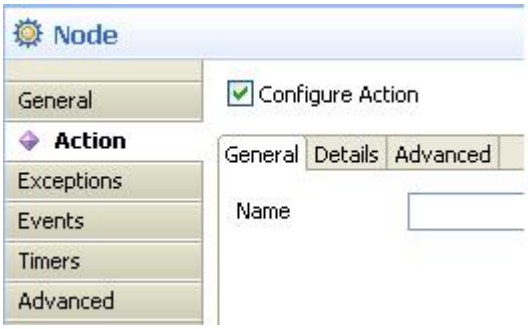
Steps	Description	Comments
		jBPM provided *ActionHandler* interface.
Attribute-Action-Handling (Contd ...)	15. Choose the appropriate action handler class from the matching items. Click OK when finished. 	
Attribute-Action-Handling (Contd ...)	16. Click on the advanced tab. 	The wizard is responsible for event propagation and asynchronous behavior. Default is true|yes. (Refer jBPM Wiki Page 139) Asynchronous implementation is a known bug issue in jBPM. *Refer to JIRA-#1114*

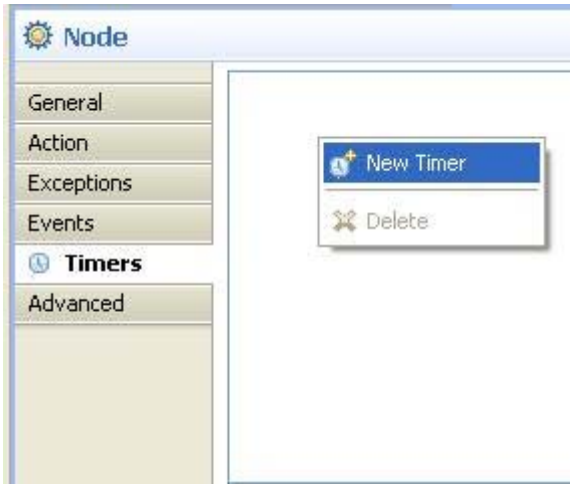
3.4.4. Creating a Node

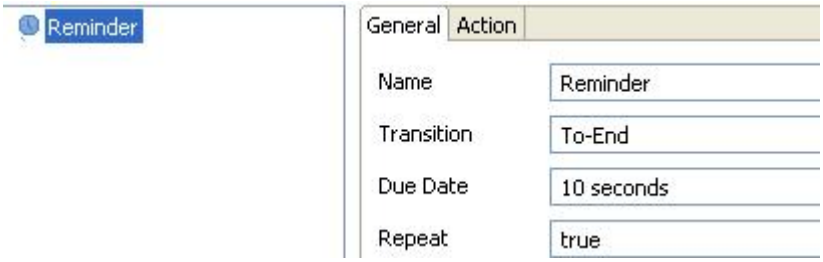
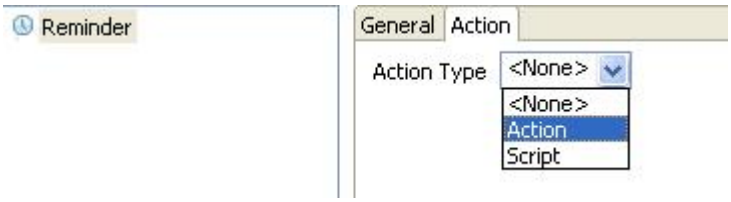
This node serves the situation where the task is automatic. The node expects one sub element action. The action is executed when the execution token arrives at the node. This node can be used if one wants to use Java to implement some functional logic that is required for the business process.

Steps	Description	Comments
Creating a Node	1. Select the Node from the design toolbar. 	
Create a Workflow	2. Create a workflow as shown.	

Steps	Description	Comments
		
Configuring Node	3. Select the Node. Click on the properties tab below. 	
Configuring Node (Contd...)	4. Change the default name of the Node from ‘node1’ to a name as shown ‘Approval’. 	

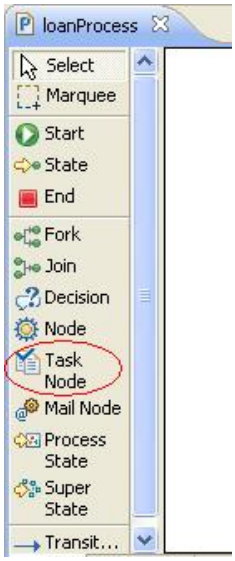
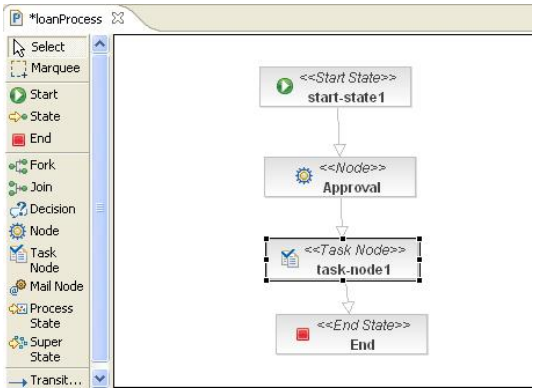
Steps	Description	Comments
Configuring Node (Contd...)	5. Changes will be reflected to the Node named approval. 	
Node Attribute-Action	6. Click on the Action tab in left window property bar. 	
Node Attribute-Action(Contd …)	7. Check the Configure Action checkbox. 	
Node Attribute-Action -Handler	8. *Refer to Section 3.4.1 under Start-state Attribute Events using actions*	

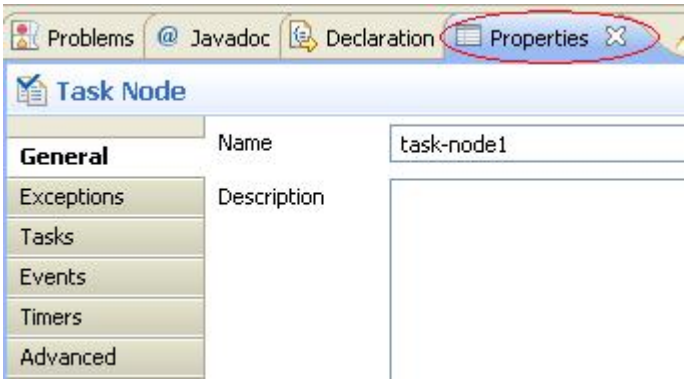
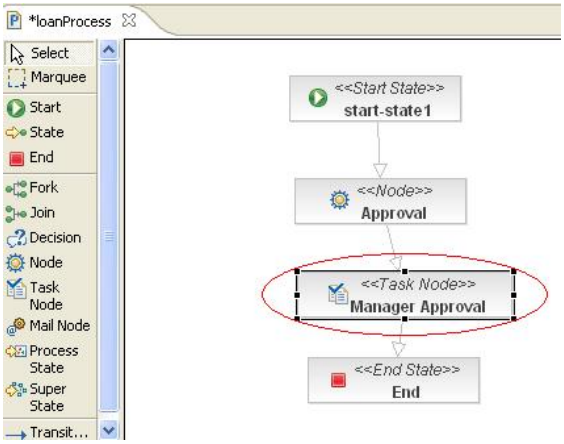
Steps	Description	Comments
Node-Attribute Exceptions	9. *Refer to Section 3.4.1 under Start-state Attribute Exceptions*	
Node-Attribute Events	10. *Refer to Section 3.4.1 under Start-state Attribute Events*	Nodes are not relevant for tasks therefore a task related event is not applicable to a node.
Node-Attribute Timers	11. Click on the tab timer in the left window property bar. 	Timers are required if we want an action associated with this node to occur on a scheduled basis.
Node-Attribute Timers (Contd ...)	12. Right click on the left blank area and click on New Timer. 	
Node-Attribute Timers (Contd ...) Configuration	13. Following screen will appear to configure the timer. 	

Steps	Description	Comments
Node-Attribute Timers (Contd ...) Configuration	14. Give name of the timer, Transition, Due Date, Repeat in the textbox provided. 	*Due-Date*-It is the Duration (optionally expressed in business hours) that specifies the time period between the creation of the timer and the execution of the timer. *Repeat*-If set to true then the action will repeat after every due date. *A transition*-Name of the next node to flow to
Node-Attribute Timers (Contd ...) Action Configuration	15. Click on the Action tab. 	Specify the action corresponding to a timer that will be executed when the timer fires.
Node-Attribute Timers (Contd ...) Action Configuration	16. Select action from the Action Type dropdown. 	
Node-Attribute Timers (Contd ...) for Action Configuration	17. *Refer to Section 3.4.1 under Start-state Attribute Events using actions*	
Node-Attribute Advanced	18. Check the checkbox to make the timer asynchronous if required.	Asynchronous implementation is a known bug in jBPM. *Refer to JIRA-#1114*

3.4.5. Creating a Task Node

A task node represents one or more tasks that are to be performed by human. So when execution arrives in a task node, task instances will be created in the task lists of the workflow participants. After that, the node will go to a wait state meaning that some manual work is required for completion of the task. So when the users perform their task, the task completion will trigger the resuming of the execution.

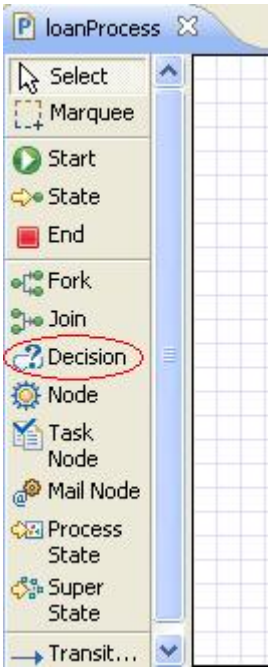
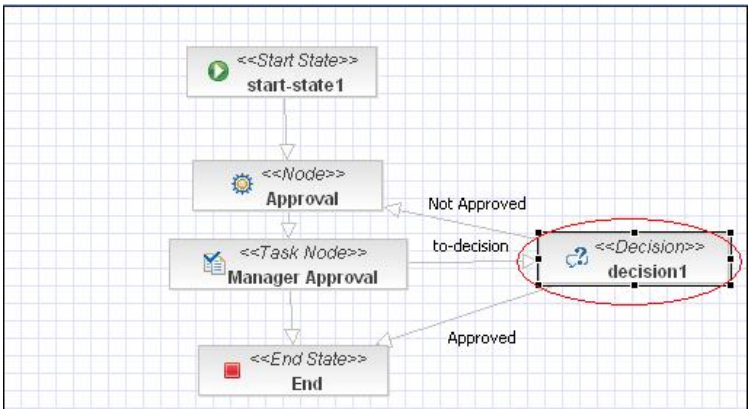
Steps	Description	Comments
Creating a Task node	1. Select the Task-Node from the design toolbar. 	
Creating a Task node (Contd...)	2. Drop it on the process design Editor. Workflow after creating a task node. 	
Creating a Task node (Contd...)	3. Click on the properties tab.	

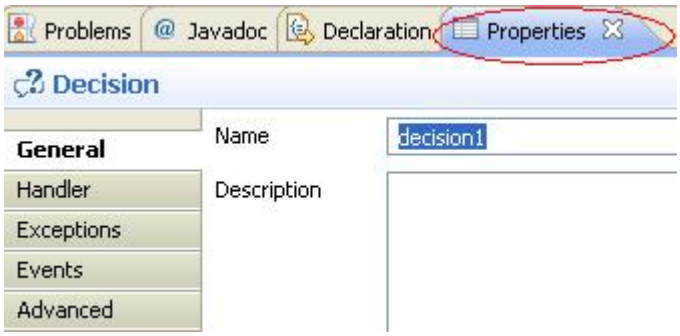
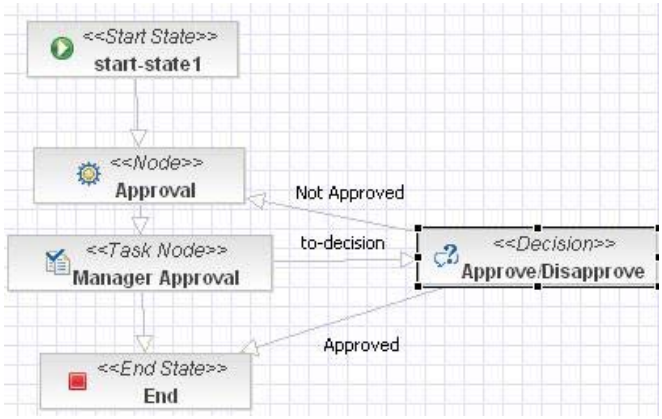
Steps	Description	Comments
		
Creating a Task node (Contd...)	4. Enter the name of the task in the 'Name' textbox. 	
Creating a Task node (Contd...)	5. Changes will be reflected in the process design editor as follows. 	
Task-Node Attribute Exceptions	6. *Refer to Section 3.4.1 under Start-state Attribute Exceptions*	
Task-Node Attribute Tasks	*7. Refer to section-3.4.1 under Start-state Attribute-Task Configuration*	
Task-Node Attribute Events	*8. Refer to section-3.4.1 under Start-state Attribute-Events*	

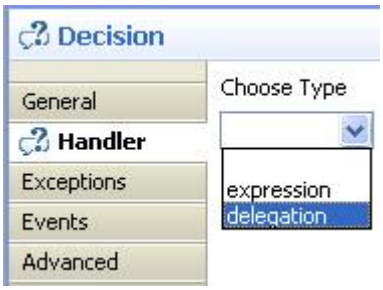
Steps	Description	Comments
Task-Node Attribute Timers	*9. Refer to section-3.4.4 under Node Attribute-Timers*	
Task-Node Attribute Advanced	*10. Refer to section-3.4.4 under Node Attribute-Advanced*	

3.4.6. Creating a Decision Node

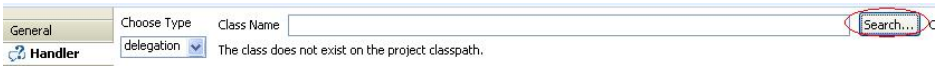
There are two ways to specify the decision criteria in a jbpm process. One is by adding condition elements on the transitions. Conditions are script expressions that returns a 'Boolean'. At runtime the decision node will loop over its leaving transitions and evaluate each condition. The first transition for which the conditions resolve to 'true' will be taken. Alternatively, an implementation of the DecisionHandler can be specified. Then the decision is generated by a Java class and the selected leaving transition is returned by the decide-method of the DecisionHandler implementation.

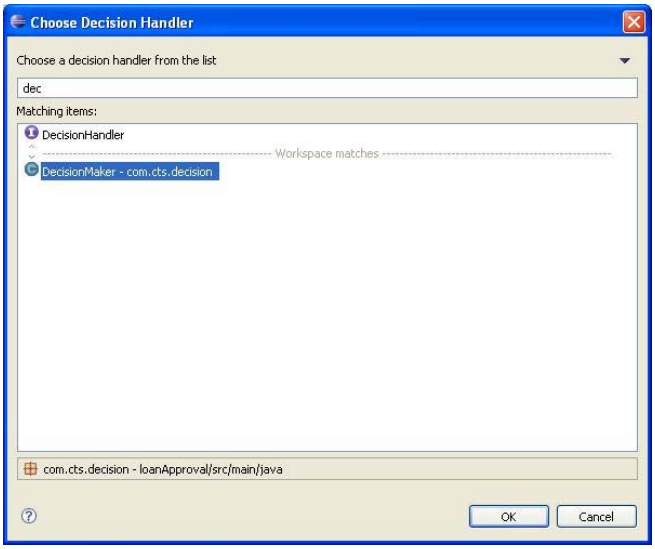
Steps	Description	Comments
Creating a decision node	1. Select the Decision Node from the toolbar. 	
Creating a decision node (Contd...)	2. Drop on the process design editor. Design your process in the process design editor. 	

Steps	Description	Comments
Creating a decision node (Contd...)	3. Click on the properties tab below. 	
Creating a decision node (Contd...)	4. Change the name to 'Approve/Disapprove' 	
Creating a decision node (Contd...)	Changes will be reflected in the process design editor. 	
Decision node-Attribute Handler	5. Click on the Handler tab. 	

Steps	Description	Comments
Decision node-Attribute Handler (Contd...)	6. Select choose type Delegation from the dropdown. 	

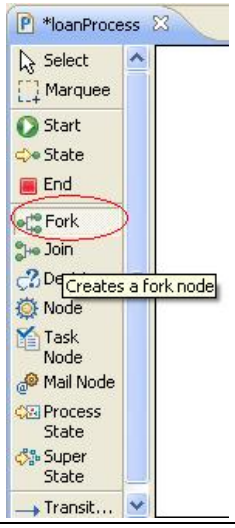
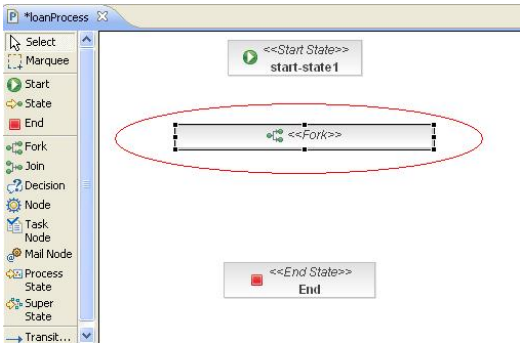
Decision node-Attribute Handler (Contd...)	7. Create a custom Decision Handling class for the decision handler.
	`import org.jbpm.graph.exe.ExecutionContext;`
	`import org.jbpm.graph.node.DecisionHandler;`
	`public class DecisionMaker implements DecisionHandler {`
	`private static final long serialVersionUID = 1L;`
	`public String decide(ExecutionContext executionContext) throws Exception {`
	`String transition="";`
	`/*For the time being assume that the flag approve has been set to true to it's prior state*/`
	`boolean approve=true;`
	`/* assume that an approval flag is generated from the previous task. Check whether the flag is true or false.If it's true delegate the token to the End state else return it back to the Approval node.*/`
	`if(approve==true)`
	`{`
	`//Set the transition name to this variable.Delegation will take place over that transition upon returning the transition name.`
	`transition="Approved";`
	`}`
	`else`
	`{`
	`transition="Not Approved";`
	`}`
	`return null;`
	`}`
	`}`
		The custom DecisionHandler class will implement jBPM provided interface *Decision Handler* and implement the decide method which returns the transition name. Token will traverse to that transition which decide () will return.

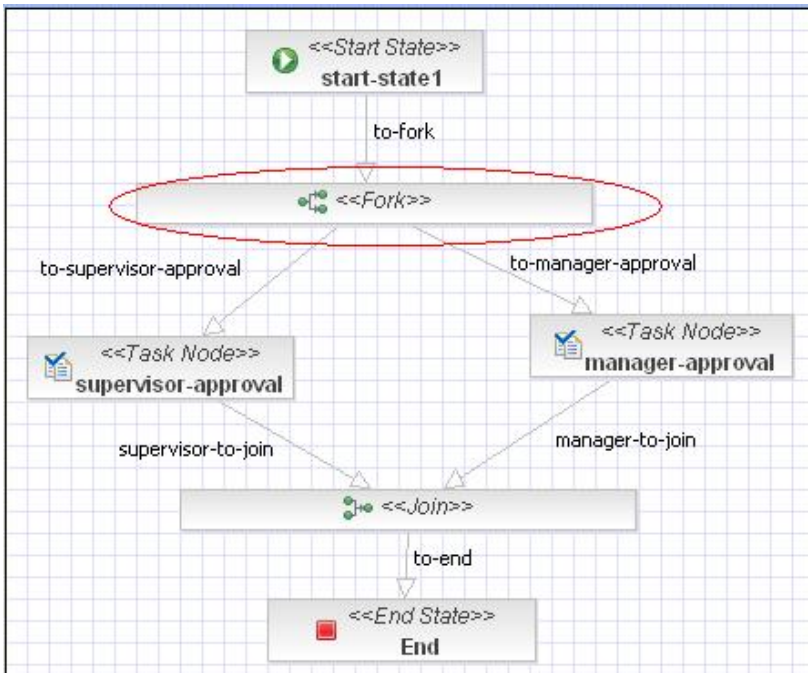
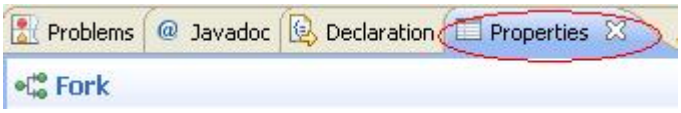
Decision node-Attribute Handler (Contd...)	8. Choose your newly created class to set the delegation. Click the search button and set the decision handler class. 	

Steps	Description	Comments
	 Click OK to finish.	
Decision node-Attribute Exception	9. *Refer to Section 3.4.1 under Start-state Attribute Exceptions*	
Decision node-Attribute Events	*10. Refer to Section 3.4.1 under Start-state Attribute Events*	
Decision node-Attribute Advanced	*11. Refer to section-3.4.4 under Node Attribute-Advanced*	

3.4.7. Creating a Fork

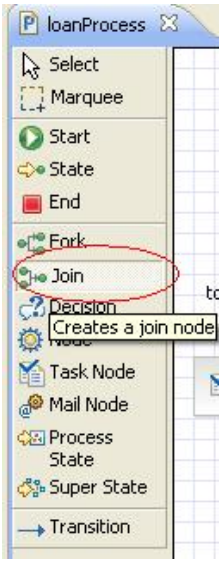
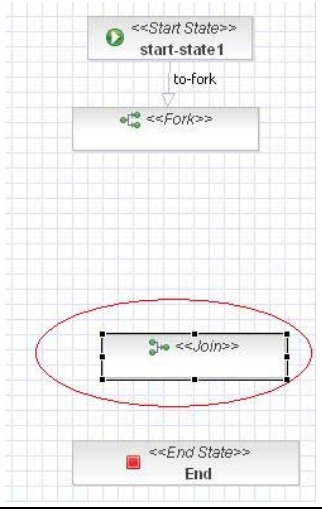
A fork splits one path of execution into multiple concurrent paths of execution. The default fork behavior is to create a child token for each transition that leaves the fork, creating a parent-child relation between the token that arrives in the fork. However, concurrent paths of execution don't have to run in separate threads in persistence mode. The important thing is to isolate each transaction to other. It should be noted that though the execution flow is not multithreaded but each transaction (A transaction is a token traversal mechanism after the token generates and reaches either to a wait state or an end state) is separate from the other.

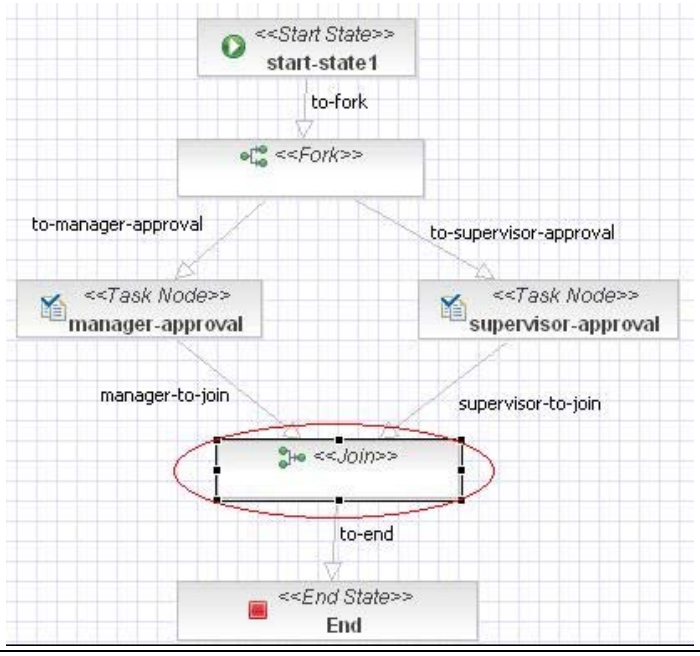
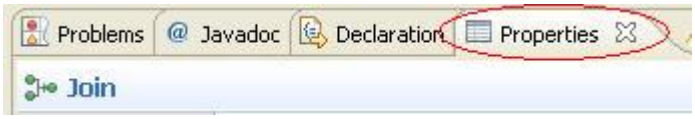
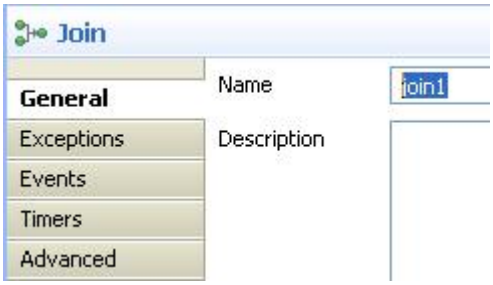
Steps	Description	Comments
Creating a fork	1. Select fork from the design toolbar. 	
Creating a fork (Contd...)	2. Drop it on the design editor. 	
Creating a fork (Contd...)	3. After creating a fork workflow may be created as follows.	Join is associated task to fork.

Steps	Description	Comments
		
Creating a fork (Contd...)	4. Click on the properties tab below the process design editor. 	
Creating a fork (Contd...)	5. Change the name of the newly created fork if required. Default name is 'fork1' for creating a first fork in the entire process. 	
Fork-Attribute Exceptions	6. *Refer to Section 3.4.1 under Start-state Attribute Exceptions*	
Fork-Attribute Events	*7. Refer to Section 3.4.1 under Start-state Attribute Events*	
Fork-Attribute Timers	*8. Refer to section-3.4.4 under Node Attribute-Timers*	
Fork-Attribute Advanced	*9. Refer to section-3.4.4 under Node Attribute-Advanced*	

3.4.8. Creating a Join

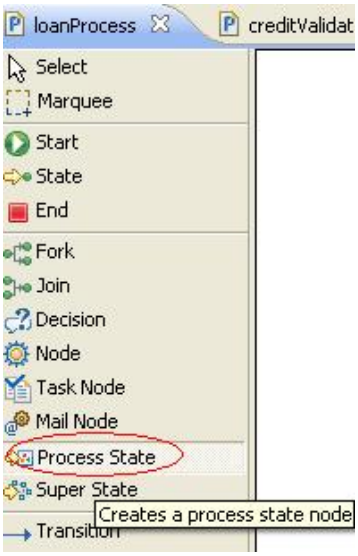
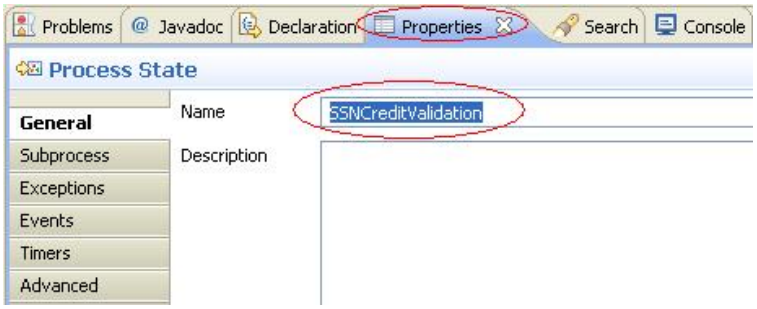
The default join assumes that all tokens that arrive in the join are children of the same parent. This situation is created when using the fork,all tokens created by a fork arrive at the same join. A join will end every token that enters the join. Then the join will examine the parent-child relation of the token that enters the join. When all sibling tokens have arrived in the join, the parent token will be propagated over the (unique) leaving transition. When there are still sibling tokens active, the join will behave as a wait state.

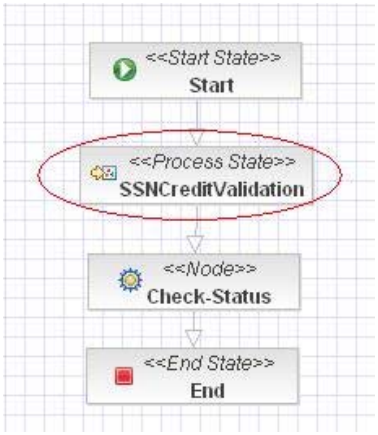
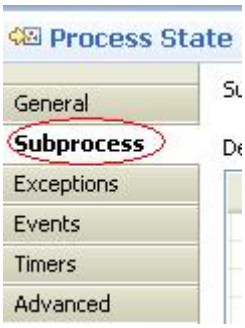
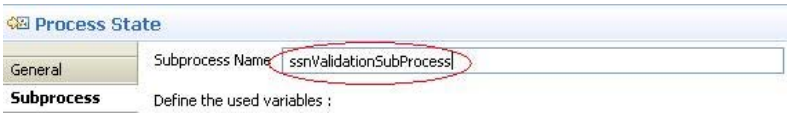
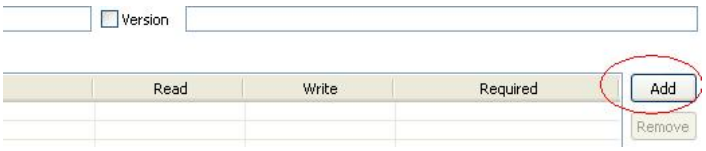
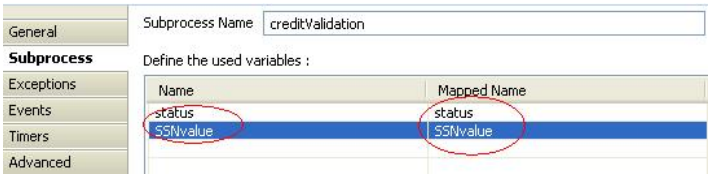
Steps	Description	Comments
Creating a Join	1. Select join from the design toolbar. 	
Creating a Join (Contd...)	2. Drop it on the process design editor. 	
Creating a Join (Contd...)	3. After creating a Join workflow may be created as follows.	

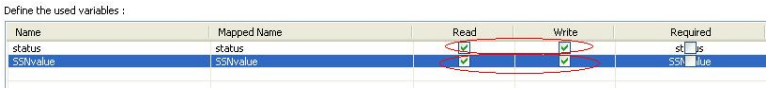
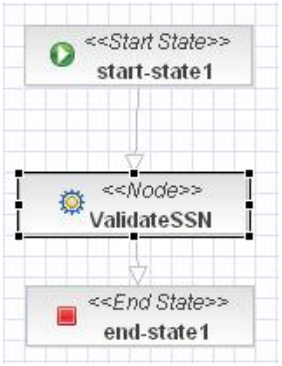
Steps	Description	Comments
		
Creating a Join (Contd...)	4. Click on the properties tab below the process design editor. 	
Creating a Join (Contd...)	5. Change the name of the newly created join if required. Default name is 'join1' for creating a first join in the entire process. 	
Join-Attribute Exceptions	6. *Refer to Section 3.4.1under Start-state Attribute Exceptions*	
Join-Attribute Events	7. *Refer to Section 3.4.1 under Start-state Attribute Events*	
Join-Attribute Timers	8. *Refer to section-3.4.4 under Node Attribute-Timers*	
Join-Attribute Advanced	9. *Refer to section-3.4.4 under Node Attribute-Advanced*	

3.4.9. Creating a Process-State

Process composition is supported in jBPM by means of the process-state. The process state is a state that is associated with another process definition. When graph execution arrives in the process state, a new process instance of the sub-process is created and it is associated with the path of execution that arrived in the process state. The path of execution of the super process will wait until the sub process instance ends for a synchronous execution. In case of asynchronous executions, process state still goes to blocking mode. As mentioned in section 3.4.4 step 18, this defect of the tool is already logged (*Refer to JIRA-#1114)*. When the sub process instance ends, the path of execution of the super process will leave the process state and continue graph execution in the super process.

Steps	Description	Comments
Creating a process State	1. Click on the Process-State tool in the Process design toolbar. 	
Creating a process State(Contd ..)	2. Drop it on the process design editor. Click on the properties tab to change the name of your process state. Default is 'process-state1'. 	
Creating a process State (Contd ...)	3. Workflow model after creating a process state.	

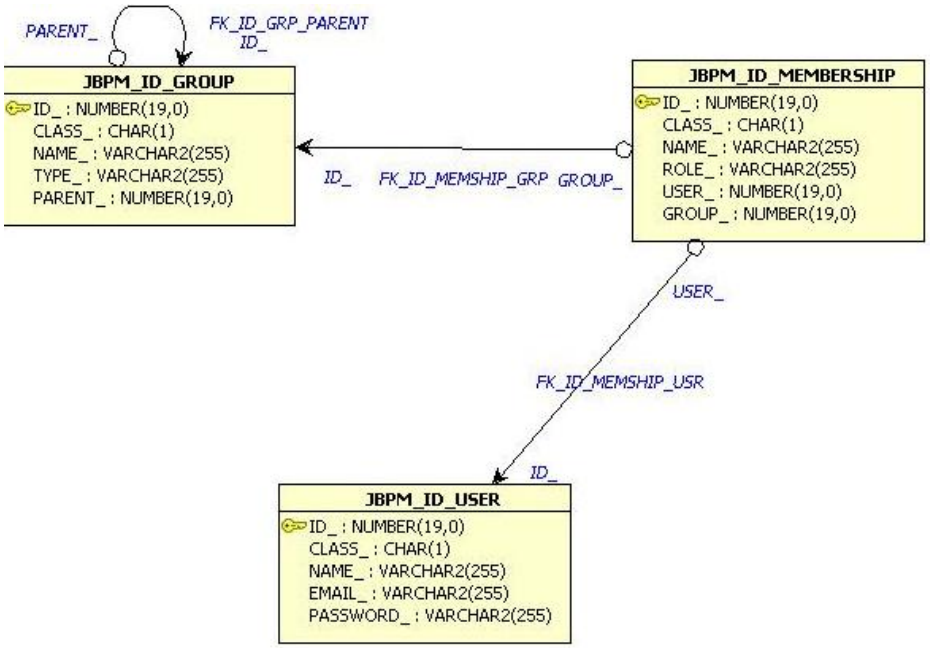
Steps	Description	Comments
		
Process-State Attribute Subprocess	4. Click on the Subprocess tab under Process-State Property. 	
Process-State Attribute Subprocess (Contd...)	5. Enter the sub process name. 	Refer to figure at step-9 below to design the subprocess .
Process-State Attribute Subprocess (Contd...)	6. Click Add to add the variables of parent process that are to be mapped with the child process. 	
Process-State Attribute Subprocess (Contd...)	7. Enter the variables of parent process that are to be mapped with the child process. 	

Steps	Description	Comments
Process-State Attribute Subprocess (Contd...)	8. Define access permission as per business need. Three permissions are allowable, **Read**, **Write** and **Required**. 	
Process-State Attribute Subprocess (Contd...)	9. Design the Child Process workflow as follows. 	Refer figure at step-5 above. The following picture shows the process flow diagram of the process *ssnValidationSubProcess* which is entered as the subprocess name of the property subprocess of Process State.
Process-State - Attribute Exceptions	10. *Refer to Section 3.4.1 under Start-state Attribute Exceptions*	
Process-State - Attribute Events	11. *Refer to Section 3.4.1 under Start-state Attribute Events*	
Process-State - Attribute Timers	12. *Refer to section-3.4.4 under Node Attribute-Timers*	
Process-State - Attribute Advanced	13. *Refer to section-3.4.4 under Node Attribute-Advanced*	

3.5. Migrating jBPM to Oracle Database

Migrating to Oracle is required due to limited functionality provided by the JBoss in-built database Hypersonic. Hypersonic does not provide more than one connection at a time to a database schema. It generates a lock file as soon as a JBoss application server is started. Consequently a standalone client application which tries to execute the process deployed on the server fails to obtain a connection instance to the database. To overcome this limitation it's advisable to migrate jBPM from Hypersonic to Oracle or to any other supported enterprise databases.

The following table describes step by step procedure how to migrate Jbpm database scripts to Oracle database.

Steps	Description	Comments
Setting up Oracle schema and user for necessary authentication	1. Create an user in Oracle database so that the JBPM schema can be created using that user credential.	
	2. Create a schema using the given Oracle Script located at <jBPM_JPDL_HOME>\db named as **jBPM.jpdl.oracle.sql**.	
	3. Create user as defined in web.xml located at jBPM-console.war\WEB-INF. A user who wants to login into the jBPM-console should have at least a '*user*' role. Additional role is defined for a user for accessing further modules of a jBPM deployed process. For example a user without having *admin* role can't delete a process using jBPM-Console. Similarly a user without having '*manager*' role can't start a process. Three tables in the schema are responsible to maintain jBPM authentication and authorization -constraints. An ER diagram to those tables is given below.  	

Steps	Description	Comments
Setting up jBPM-console.war	1. Extract jBPM-console.war from <jBPM_JPDL_HOME>\server\server\jbpm\deploy	
Setting up jBPM-console. war(Contd ...)	2. Make following changes to the \WEB_INF\classes\hibernate.cfg.xml file. <!-- hibernate dialect --> <property name="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</property> <!-- JDBC connection properties (begin) ===--> <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property> <property name="hibernate.connection.url">jdbc:oracle:thin:@10.227.32.35:1521:orcl</property> <property name="hibernate.connection.username">sa</property> <property name="hibernate.connection.password">sa</property> <!--==== JDBC connection properties (end) --> <property name="hibernate.cache.provider_class">org.hibernate.cache.HashtableCacheProvider</property>	The database connection url will be specific to the installed database server connection parameters.

Steps	Description	Comments

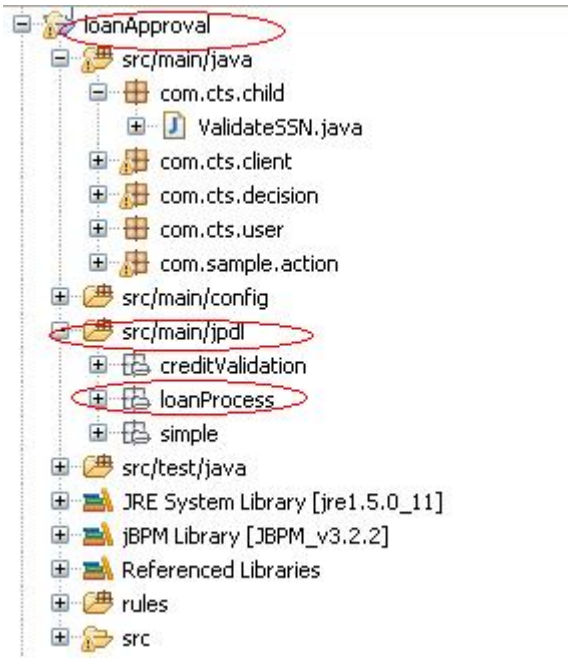
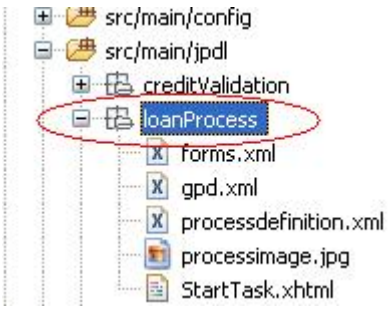
	3. Replace the existing <jBPM_JPDL_HOME>\server\server\jBPM\deploy\jBPM-ds.xml with the following entries. This is required since an Oracle datasource needs to be defined to authenticate and authorize a user from the jBPM identity tables that has been migrated to Oracle Database now. The JNDI name provided here is referred by <jBPM_JPDL_HOME>\server\server\jBPM\conf\login-config.xml in its jBPM application-policy element. The following is the content of jBPM-ds.xml.
	
	`<?xml version="1.0" encoding="UTF-8"?>`
	
	`<datasources>`
	`</local-tx-datasource>`
	`<jndi-name>JbpmDS</jndi-name>`
	`<connection`
	`url>jdbc:oracle:thin:@10.227.32.35:1521:orcl</connec`
	`tion-url>`
	`<driver-`
	`class>oracle.jdbc.driver.OracleDriver</driver-class>`
	`<user-name>sa</user-name>`
	`<password>sa</password>`
	`<min-pool-size>5</min-pool-size>`
	`<max-pool-size>20</max-pool-size>`
	`<idle-timeout-minutes>5</idle-timeout-`
	`minutes>`
	`</local-tx-datasource>`
	`</datasources>`	

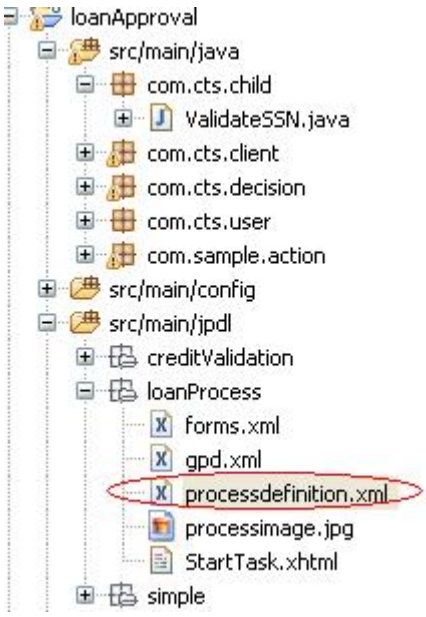
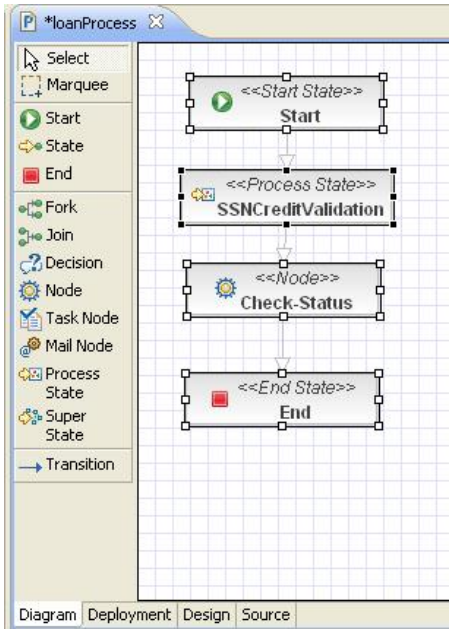
<table>
<tr><th>Steps</th><th>Description</th><th>Comments</th></tr>
<tr><td></td><td>4. Additionally configuring datasource is required since <jBPM_JPDL_HOME>\server\server\jBPM\conf\login-config.xml requires a dsJndiName (datasource JNDI name) in its jBPM application policy. Below shows that how jBPM application policy in login-config.xml uses the datasource. So before creating a datasource make sure that the datasource has been deployed on the server and the jndi has got registered to the server and is specified appropriately with the login-config.xml. Otherwise most of the time a login violation occurs at the jBPM console due to inappropriate jndi specification.

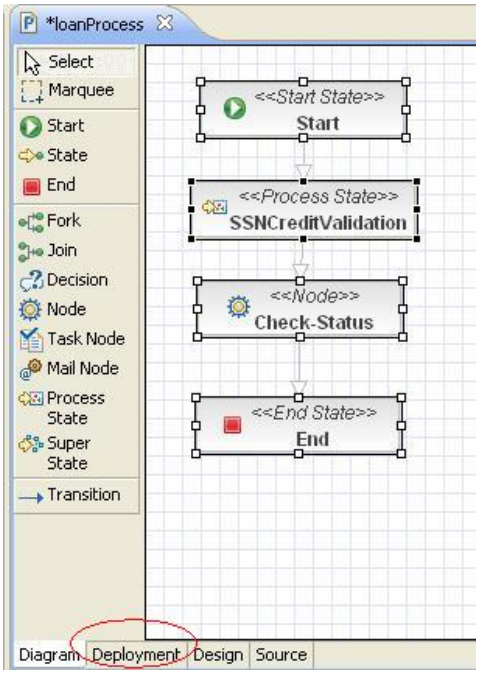
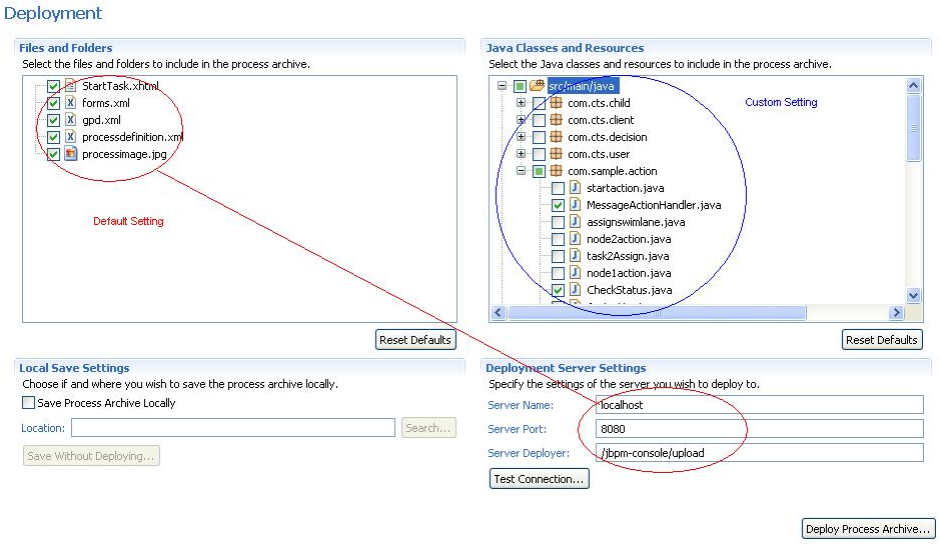
<pre>
<application-policy name = "jbpm">
  <authentication>
    <login-module
code="org.JBoss.security.auth.spi.DatabaseServerLoginModule"
              flag="required">
      <module-option name="dsJndiName">java:/JbpmDS</module-
option>
      <module-option name="principalsQuery">
        SELECT PASSWORD_ FROM jBPM_ID_USER WHERE
NAME_=?
      </module-option>
      <module-option name="rolesQuery">
        SELECT g.NAME_,'Roles'
        FROM jBPM_ID_USER u,
            jBPM_ID_MEMBERSHIP m,
            jBPM_ID_GROUP g
        WHERE g.TYPE_='security-role'
          AND m.GROUP_ = g.ID_
          AND m.USER_ = u.ID_
          AND u.NAME_=?
      </module-option>
    </login-module>
  </authentication>
</application-policy>
</pre></td><td></td></tr>
<tr><td rowspan="4">Setting up jBPM-console.
war(Contd ...)</td><td>5. Download latest Oracle driver jar ojdbc14.jar and copy it to <jBPM-Console.war extract>/WEB-INF/lib as well copy the jar file to the <server>/lib folder.</td><td rowspan="4"></td></tr>
<tr><td>6. Recreate jBPM-console.war</td></tr>
<tr><td>7. Redeploy the war file to the server.</td></tr>
<tr><td>8. Restart the server.</td></tr>
</table>

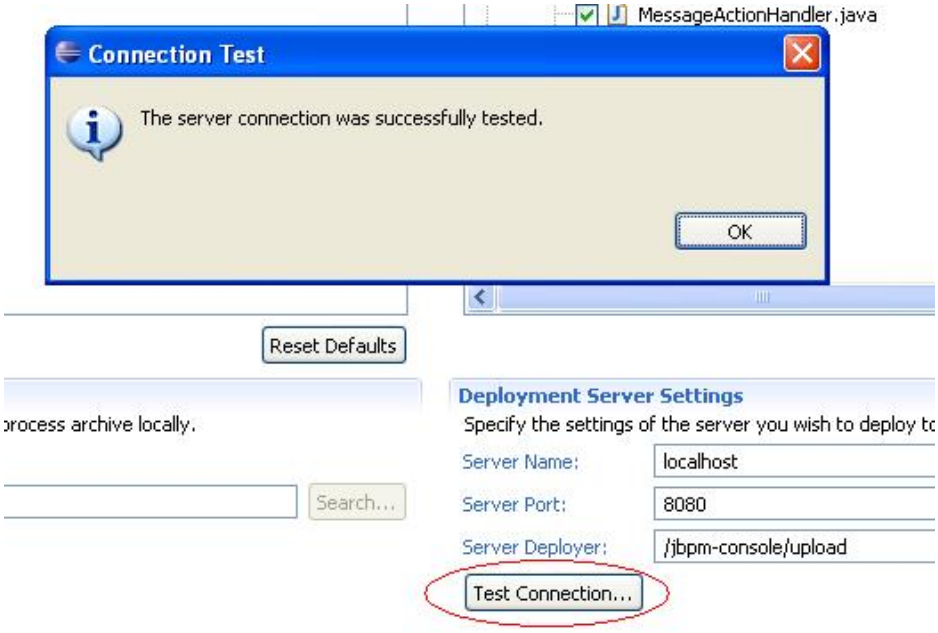
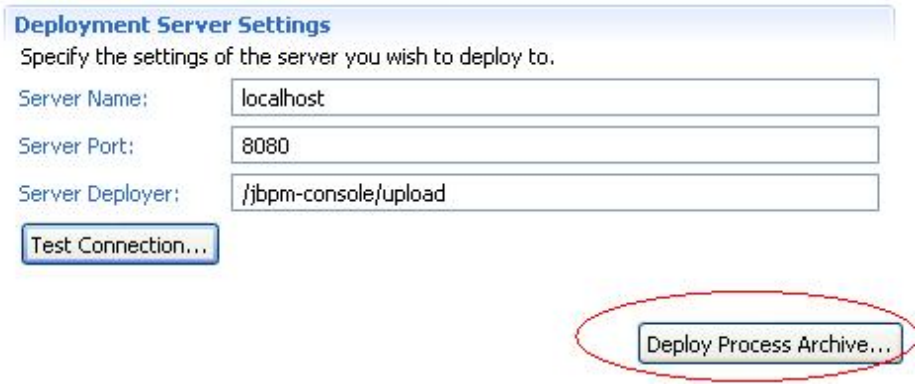
3.6. Process Deployment

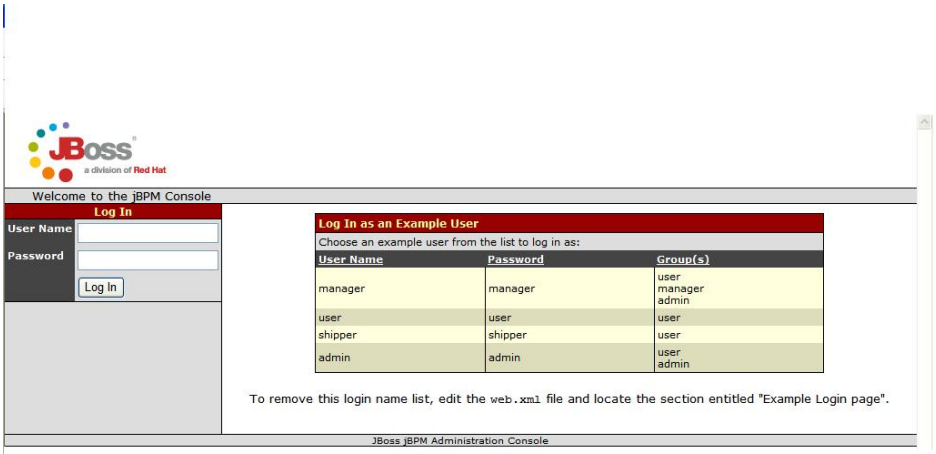
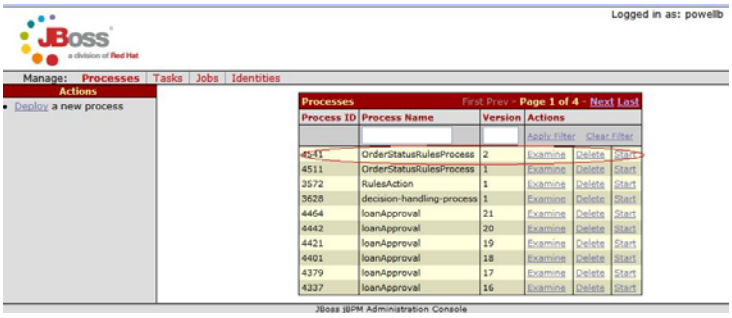
Typical deployment of jBPM process will include persistent storage of process definitions. When a process is deployed the back-end database tables are populated with different information related to a jPDL such as process-id, nodes, transition, roles, swimlane, variables etc.

Steps	Description	Comments
Deploy Process	1. Double click on the jpdl folder of a process project. 	
Deploy Process (Contd...)	2. Double click on the desired process you want to deploy. Here click on the loanProcess. 	In case of a process comprising of sub-processes, the sub-processes must be deployed before the main process because the main process at runtime refers to the sub-process Id which gets created in the database only after the process is deployed.

Steps	Description	Comments
Deploy Process (Contd...)	3. Double Click on the processdefintion.xml of the process. 	
Deploy Process (Contd...)	4. Above will open the following. 	

Steps	Description	Comments
Deploy Process (Contd...)	5. Click on the deployment tab below the process design editor. 	
Deploy Process (Contd...)	6. This window will open the deployment wizard. 	**Java classes and Resources** are for back-end beans, action classes etc that are associated with the process entities. Choose the checkbox as per the runtime deployment requirements.

Steps	Description	Comments
Deploy Process (Contd...)	7. Click on the test connection before any deployment. For this click on the Test Connection. Upon successful testing the following will appear. 	
Deploy Process (Contd...)	8. Click on the Deploy Process Archive button for deployment. 	
Deploy Process (Contd...)	9. Upon successful deployment the following message will be displayed.	

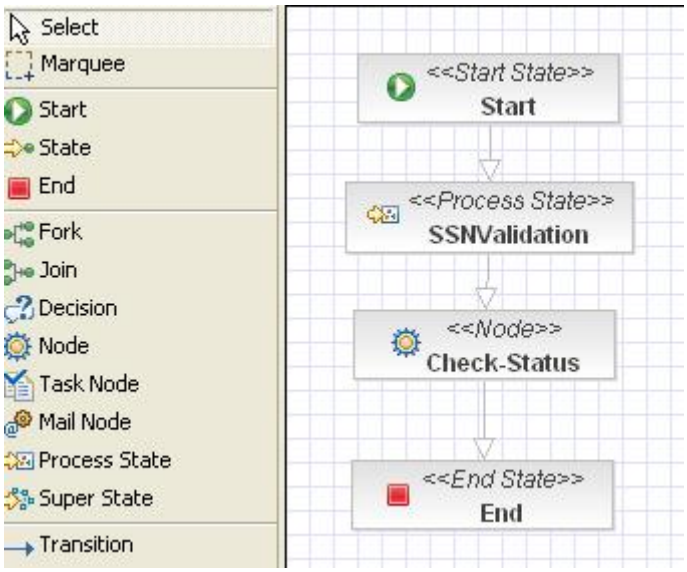
Steps	Description	Comments
		
jBPM -console	10.Log into the jBPM-console http://<host-name>:8080/jBPM-console(As per jBPM-3.2.2) specification. 	
Verification of the deployed process	11.After login the following will appear with the process name that you have deployed recently. 	

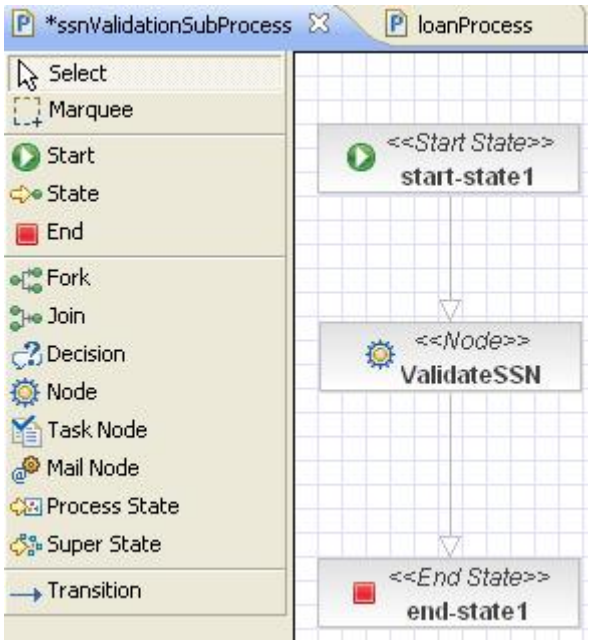
3.7. Creating a Client to invoke jBPM deployed process

Sometimes business scenario needs to invoke certain process application outside the BPM environment. jBPM offers a set of APIs to invoke a jBPM process outside its environment.

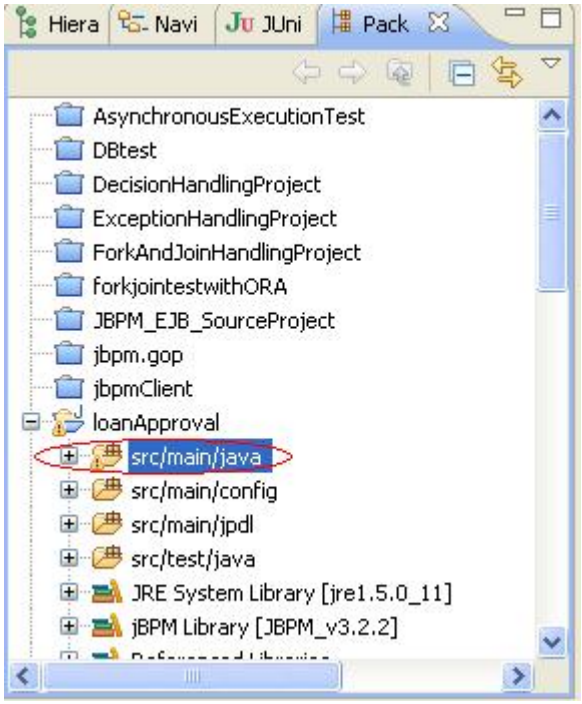
The following table shows step by step implementation for creating a standalone process client.

Process Overview: The following process validates a SSN entered by the user who wants to apply for a loan. The SSNValidation (Social Security Number) will be done in a sub process.

Steps	Description	Comments
Creating a jBPM Main Process	1. Create a process named loanProcess. Below shows the workflow of our loanProcess. 	
Creating a jBPM subprocess	2. The main process has a Process State. Corresponding to this we create a child process or a subprocess. Our subprocess workflow is as below.	

Steps	Description	Comments
		

Steps	Description	Comments
jPDL of Main Process	<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="loanProcess"> <start-state name="Start"> <task name="StartTask"> </task> <transition to="SSNCreditValidation"></transition> <event type="task-create"> <action name="startaction" class="com.sample.action.StartingAction"></action> </event> </start-state> <process-state name="SSNCreditValidation"> <sub-process name="ssnValidationSubProcess"></sub-process> <variable access="read,write" name="status" mapped-name="status"></variable> <variable access="read,write" name="SSNvalue" mapped-name="SSNvalue"></variable> <transition to="Check-Status"></transition> </process-state> <node name="Check-Status"> <action name="Check-SSNApproval" class="com.sample.action.CheckStatus"></action> <exception-handler exception-class="java.lang.Exception"> <action name="exceptionhandle" class="com.sample.action.MessageActionHandler"></action> </exception-handler> <transition to="End"></transition> </node> <end-state name="End"></end-state> </process-definition>	
jPDL of Sub Process	<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="ssnValidationSubProcess"> <start-state name="start-state1"> <transition to="ValidateSSN"></transition> </start-state> <node name="ValidateSSN"> <action name="creditAction" class="com.cts.child.ValidateSSN"></action> <exception-handler exception-class="java.lang.NullPointerException"> <action name="ValidateException" class="com.sample.action.MessageActionHandler"></action> </exception-handler> <transition to="end-state1"></transition> </node> <end-state name="end-state1"></end-state> </process-definition>	

Steps	Description	Comments
Create action class	3. Create action class in the following folder under the process project. Create suitable package under this folder. 	

com.sample.action.CheckStatus Class in main Process	The Class CheckStatus is responsible to retrieve the variable value status which is the validation result of SSN entered by the user and returned by the subprocess.
	`public class CheckStatus implements ActionHandler {`
	`  private static final long serialVersionUID = 1L;`
	
	`  public void execute(ExecutionContext executionContext) throws Exception {`
	`  System.out.println("***********Inside Parent Process*****");`
	`  String status=(String)executionContext.getContextInstance().getVariable("status");`
	`  System.out.println("Status :"+status);`
	`  }`
	
	`}`	

Steps	Description	Comments
com.sample.action.MessageActionHandler	This action class is triggered whenever an exception is generated by the Exception handler.	

```
public class MessageActionHandler implements ActionHandler {

      private static final long serialVersionUID = 1L;
      String message;
      public void execute(ExecutionContext context) throws
Exception {
            context.getContextInstance().setVariable("message",
"Status is :" +
(String)context.getContextInstance().getVariable("status"));

      System.out.println((String)context.getContextInstance().getVa
riable("message"));
      }

}
```

<table>
<tr><th>Steps</th><th>Description</th><th>Comments</th></tr>
<tr><td>com.cts.child.ValidateSSN in sub process</td><td colspan="2">The following class is used to Validate a SSNvalue entered by the user.Currently this one randomly generates a true/false value irrespective of the SSNvalue.The logic of validation is kept simple.
<pre>
public class ValidateSSN implements ActionHandler
{
      private static final long serialVersionUID = 1L;

  public void execute(ExecutionContext executionContext) throws
Exception
  {
        String ssn="";

ssn=(String)executionContext.getContextInstance().getVariable("SSN
value");
        if(!ssn.equals(null))
        {
              System.out.println("Before status set in child
process
:"+(String)executionContext.getContextInstance().getVariable("stat
us"));
              Random rnd=new Random();
              if(rnd.nextBoolean()==true)
              {

executionContext.getContextInstance().setVariable("status",
"true");

executionContext.getNode().leave(executionContext);
              }
              else
              {

executionContext.getContextInstance().setVariable("status",
"false");

executionContext.getNode().leave(executionContext);
              }
        }
  }
}
</pre></td></tr>
<tr><td rowspan="2">Deployment of the Process</td><td>4. Deploy Child Process first.</td><td>Refer to section-3.6 to know how to deploy the Process</td></tr>
<tr><td>5. Deploy Main Process.</td><td>Refer to section-3.6 t to know how to deploy the Process.</td></tr>
</table>

Steps	Description	Comments

Standalone Client that will invoke the main Process	1. Graph Oriented Programming (GOP) is highly based on hibernate persistence perspective. Hibernate persistence relates to different configuration files. **default.jBPM.cfg.xml** is the parent of the entire jBPM-hibernate configuration related files. Whatever information is related to a process is maintained internally by the back-end database using persistence which requires various queries , datasource configuration related information, isolation of transaction, establishing connection to the back-end, mapping object to a table (as per hibernate fundamentals) etc. Therefore jBPM accumulates all this information from the specified file which in turn uses internally different persistence configuration files. JbpmConfiguration class retrieves all this information by parsing the default.jBPM.cfg.xml. Code snippet as follows
	`static JbpmConfiguration jbpmConfiguration = null;`
	`jbpmConfiguration = JbpmConfiguration.parseResource("default.jBPM.cfg.xml");`	

	2. A **graphSession** is a generic session specific to a GOP (Refer to Wiki Chapter -4 for GOP). Any jBPM process requires a graphSession to obtain an instance of that process. Before getting a graphSession one has to create a jbpmContext. JbpmContext will help to create a session.
	`JbpmContext jbpmContext=jbpmConfiguration.createJbpmContext();`
	`GraphSession gpsession=jbpmContext.getGraphSession();`	

	3. Using grapSession one can retrieve the existing/deployed process's definition using **findLatestProcessDefinition()** method. In previous step created graphSession is required to fetch the deployed process's ProcessDefinition. `ProcessDefinition pdef=gpsession.findLatestProcessDefinition("loanProcess");`	
	4. Create a new Instance of the process using existing processDefinition. `ProcessInstance processInstance = pdef.createProcessInstance();`	

	5. loanProcess is the parent of the ssnValidationSubProcess process. So set the value of the parent's context variables SSNvalues (Entered by the client process) and status (default set to false).
	`processInstance.getContextInstance().setVariable("SSNvalue", "123");`
	`processInstance.getContextInstance().setVariable("status", "false");`	

	6. According to GOP a token traversal mechanism takes place when execution flow reaches one state to other. This continues till a wait state or end state is reached. In a wait state one has to manually send a signal to the traversed token to continue execution. Our loanprocess has wait state at the start of the process. Therefore manually a signal must be sent to the start node.
	`Token token = processInstance.getRootToken();`
	`token.signal();`	

Steps	Description	Comments
	To retrieve the result from a loan process the following code snippet should be written in the standalone client.The parent and child process are already mapped using the 'Process State' construct. `String ssn=(String)processInstance.getContextInstance().getVariable("status");`	
	Finally close the context. `jbpmContext.close();`	
A Typical Standalone Client that will invoke the main Process		

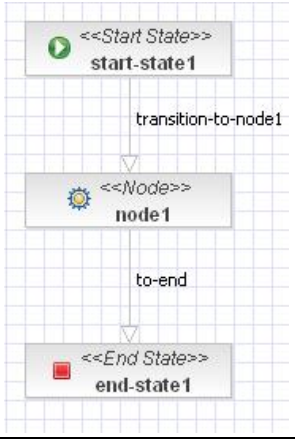
```
public class ClientApp {
      static JbpmConfiguration jbpmConfiguration = null;
      public static void main(String args[])
      {
            jbpmConfiguration =
JbpmConfiguration.parseResource("default.jbpm.cfg.xml");
            JbpmContext
jbpmContext=jbpmConfiguration.createJbpmContext();
            GraphSession
gpsession=jbpmContext.getGraphSession();

            //Create Parent Process Instance
            ProcessDefinition
pdef=gpsession.findLatestProcessDefinition("loanProcess");
                        ProcessInstance processInstance =
pdef.createProcessInstance();
                        //Set default value to the parent's
Context variable

      processInstance.getContextInstance().setVariable("SSNvalue
", "123");

      processInstance.getContextInstance().setVariable("status",
"false");

            Token token = processInstance.getRootToken();
            token.signal();
            String
ssn=(String)processInstance.getContextInstance().getVariable("SS
Nvalue");
            System.out.println("***********SSNvalue :"+ssn);
            jbpmContext.close();

      }
}
```

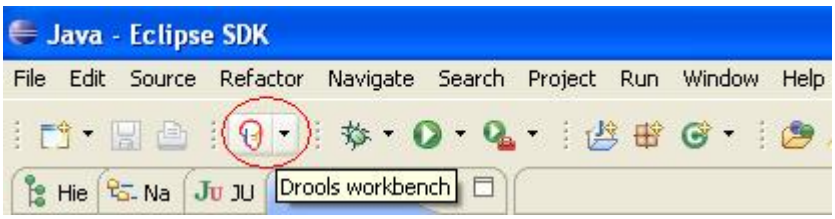
3.8. Drools Integration with jBPM

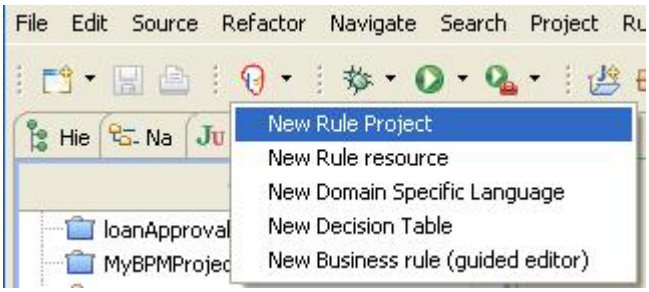
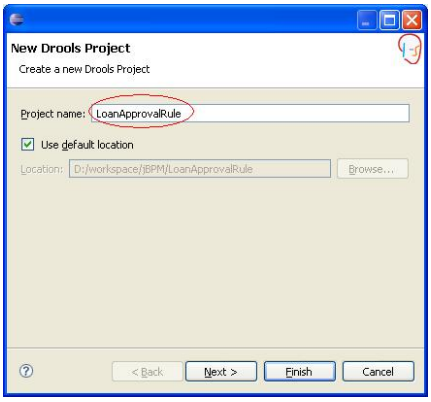
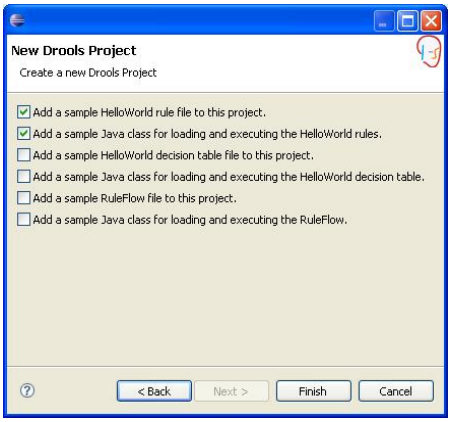
Process Engines (also capable of workflow) such as jBPM are required to graphically (or programmatically) describe steps in a process - those steps can also involve decision points which are in them a simple rule. JBoss jBPM uses expressions and delegates in its Decision nodes; which control the transitions in a Workflow. Limitation of a decision node is that much more nested if-else code style, lengthy and congested and almost difficult to comprehend. JBoss rules provide more flexibility to define complex business rule in a more human readable and understandable format. Integrating JBoss rules engine (Drools) with jBPM can therefore overcome the inherent limitation of jBPM decision nodes by leveraging Drool's flexibilities.

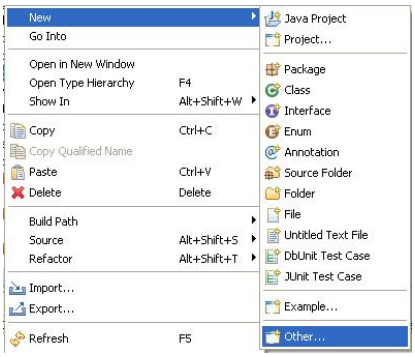
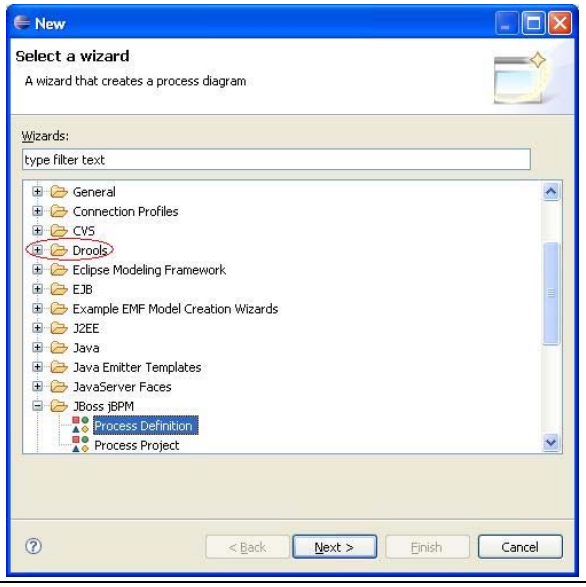
Process Overview: The following process is responsible to incorporate certain rule inside the loanApproval mechanism.

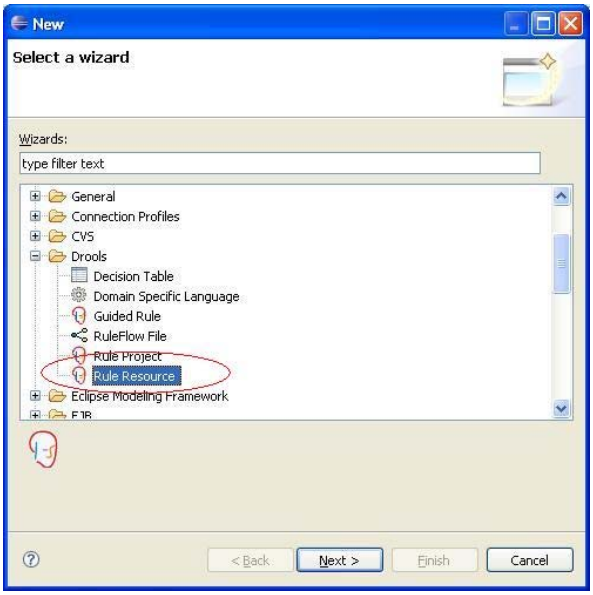
a) Below Limit: If the dollar amount of an order is under $1000, then the order is approved automatically.
b) Over Limit: For non-platinum customers, if the dollar amount of the order is greater than or equal to $1000, then the order requires manual approval.
c) Platinum Member: If the customer's status is platinum, then the order is approved automatically, regardless of the amount of the order.

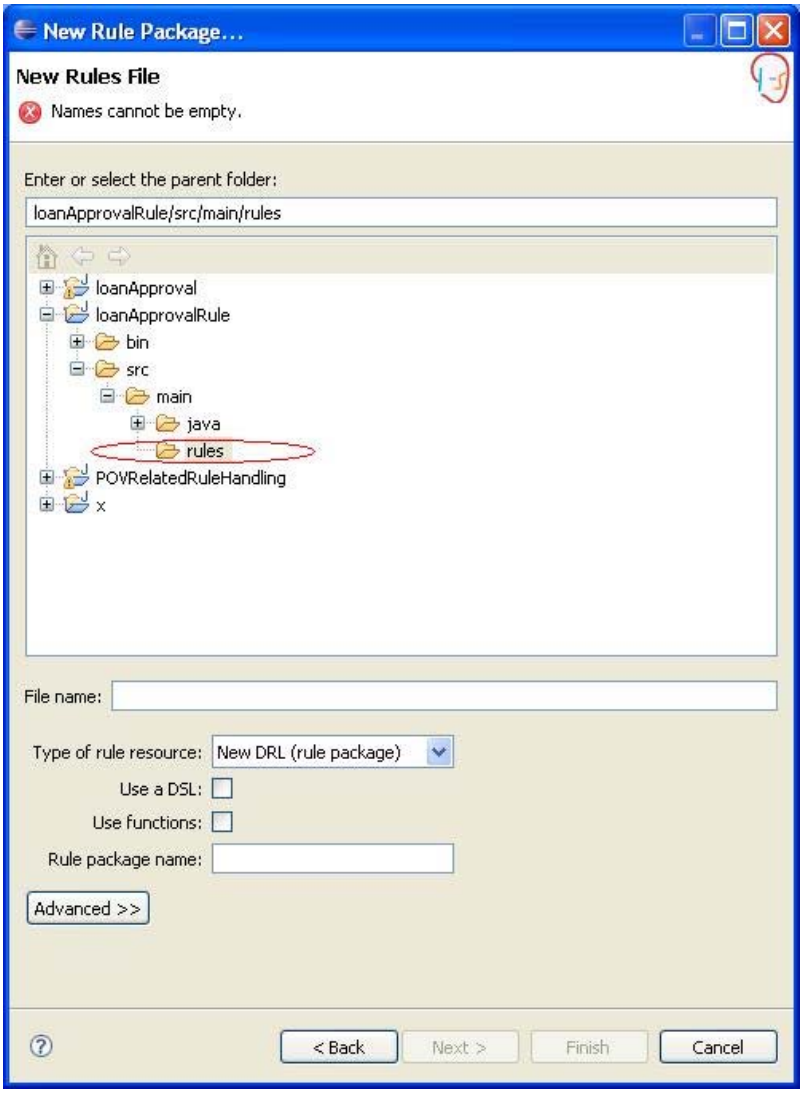
Steps	Description	Comments
Create a Process	1. Create the Process definition as follows: 	
Classpath	2. The following jar files should be in the jBPM project classpath. <jBPM_HOME>/jBPM-jpdl.jar <jBPM_HOME>/jBPM-identity.jar <jBPM_HOME>/lib/activation.jar <jBPM_HOME>/lib/antlr-2.7.6.jar <jBPM_HOME>/lib/asm.jar <jBPM_HOME>/lib/axiom-impl-1.2.4.zip <jBPM_HOME>/lib/bsh.jar <jBPM_HOME>/lib/cglib.jar <jBPM_HOME>/lib/commons-collections.jar <jBPM_HOME>/lib/commons-logging.jar <jBPM_HOME>/lib/dom4j.jar <jBPM_HOME>/lib/hibernate3.jar <jBPM_HOME>/lib/hsqldb.jar <jBPM_HOME>/lib/JBoss-backport-concurrent.jar	

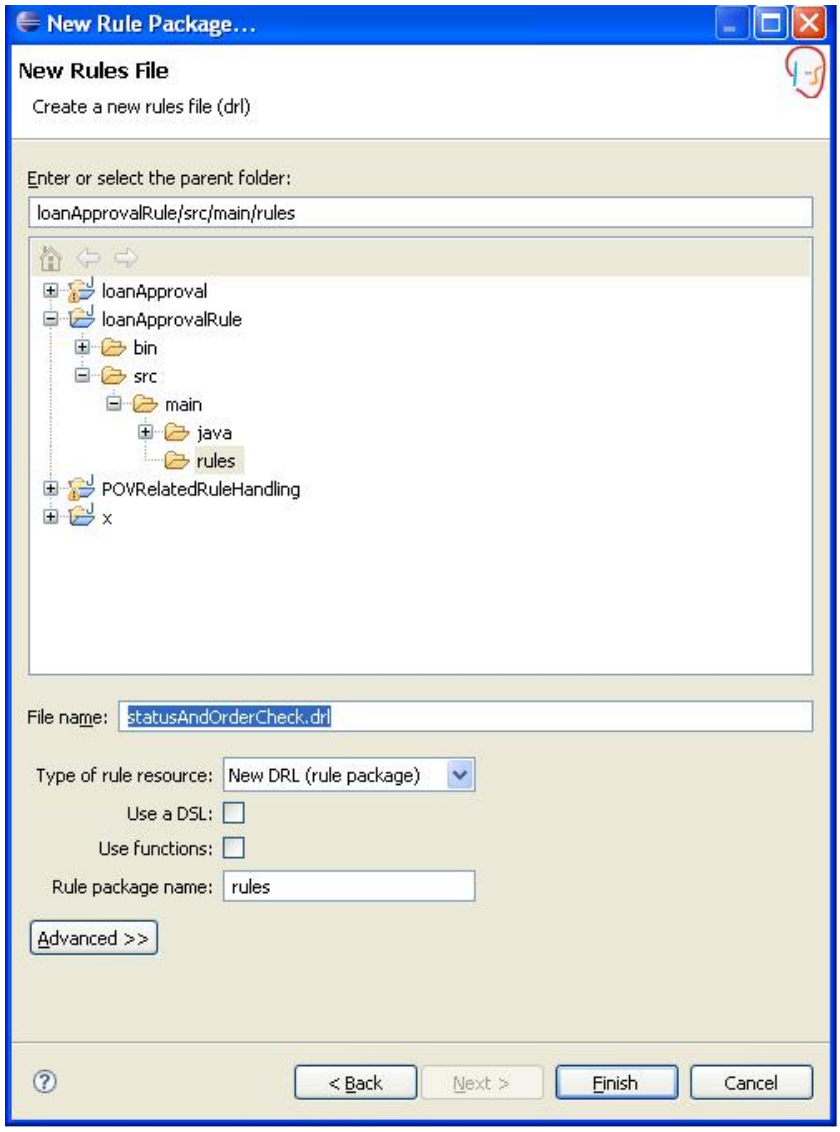
Steps	Description	Comments
	<jBPM_HOME>/lib/JBoss-j2ee.jar <jBPM_HOME>/lib/JBoss-retro-1.1.0-rt.jar <jBPM_HOME>/lib/jcr-1.0.jar <jBPM_HOME>/lib/junit.jar <jBPM_HOME>/lib/log4j.jar <jBPM_HOME>/lib/mail.jar <jBPM_HOME>/lib/ojdbc14.jar <jBPM_HOME>/lib/servlet-api.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/antlr-runtime.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/drools-compiler.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/drools-core.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/drools-decisiontables.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/drools-jsr94.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/jsr94.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/junit.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/jxl.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/mvel14.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/xercesImpl.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/xml-apis.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/xpp3.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/xpp3_min.jar <ECLIPSE_HOME>/configuration/org.eclipse.osgi/bundles/577/1/.cp/lib/xstream.jar <ECLIPSE_HOME>/plugins/org.eclipse.jdt.core_3.3.1.v_780_R33x.jar <ECLIPSE_HOME>/plugins/org.drools.eclipse_4.0.3.jar	
	org.drools.eclipse_4.0.3.jar(latest available drools jar) should also be copied into the server lib folder.	
Create a rule project	3. Click on the Drools workbench. 	

Steps	Description	Comments
Create a rule project (Contd...)	4. Click on new rule project. 	
Create a rule project (Contd...)	5. Give a name of the project in the wizard. Click next when done. 	
Create a rule project (Contd...)	6. Click finish . 	Uncheck sample HelloWorld rule and sample java class if default files are not required.
Create a rule project (Contd...)	7. This will create a new project with the project name in left of the project explorer pane. 	

Steps	Description	Comments
Create a rule project (Contd...)	8. Right Click on the loanApprovalRule and Click on New->Other. 	
Create a rule Resource	9. Double Click on the Drools. 	
Create a rule Resource (Contd...)	10. Click on the Rule Resource. Click next when done.	

Steps	Description	Comments
		
Create a rule Resource (Contd...)	11. Select the Rule project from the Project Explorer. Select rules folder as shown below:	

Steps	Description	Comments
		
Create a rule Resource (Contd...)	12. Enter the name of the Rule resource. Pattern of the rule resource file is *.drl. Enter the name of the rule resource file. Enter the package name as 'rules' or any package name. Click finish when done.	This step is mandatory.

Steps	Description	Comments
		

Steps	Description	Comments
Create a rule Resource (Contd...)	13. Create a customer bean object which consists of customer name and status.	

```
import java.io.Serializable;

public class CustomerObject implements Serializable{
	private String customerName;
	private String customerStatus;

	public CustomerObject(){

	}
	public CustomerObject(String name,String
status){
		this.setCustomerName(name);
		this.setCustomerStatus(status);

	}
	public String getCustomerName() {
		return customerName;
	}
	public void setCustomerName(String
customerName) {
		this.customerName = customerName;
	}
	public String getCustomerStatus() {
		return customerStatus;
	}
	public void setCustomerStatus(String
customerStatus) {
		this.customerStatus = customerStatus;
	}
}
```

Steps	Description	Comments
Create a rule Resource (Contd...)	14. Create an Order bean which consists of OrderId and orderValue as shown below.	
Create a rule Resource (Contd...)	15. statusAndOrderCheck.drl as follows	

```
import java.io.Serializable;

public class Order implements Serializable{
	private int orderValue;
	private int OrderId;

	public Order(){

	}
	public Order(int value,int id){
		this.setOrderId(id);
		this.setOrderValue(value);
	}
	public int getOrderValue() {
		return orderValue;
	}
	public void setOrderValue(int orderValue) {
		this.orderValue = orderValue;
	}
	public int getOrderId() {
		return OrderId;
	}
	public void setOrderId(int orderId) {
		OrderId = orderId;
	}

}
```

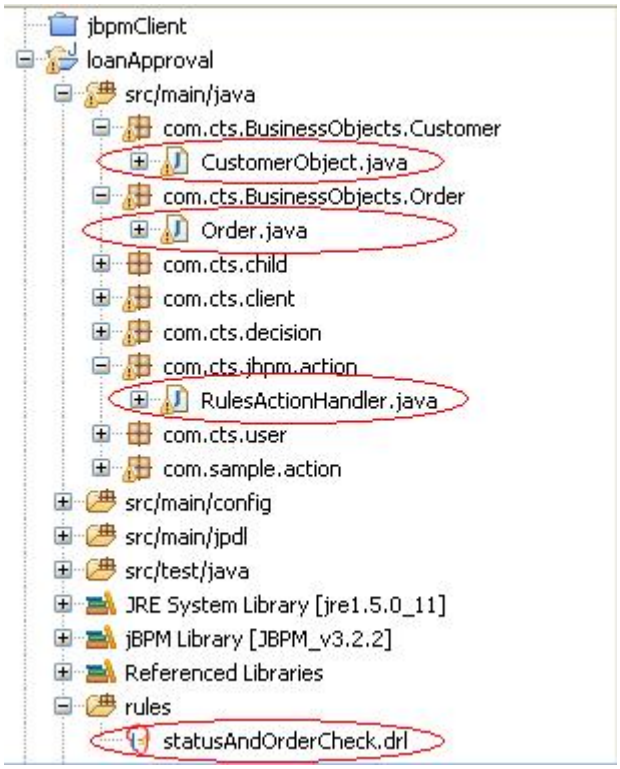
```
#created on: Jan 11, 2008
package rules

import
com.cts.BusinessObjects.Customer.CustomerObject;
import com.cts.BusinessObjects.Order.Order;
import org.jbpm.context.exe.ContextInstance;

global ContextInstance ci
rule "Determine Manager Approval Flag"
	when
		CustomerObject (customerStatus ==
"Silver") or (CustomerObject (customerStatus ==
"Gold") and Order( orderValue>=1000 ))
	then
		ci.setVariable("approvalFlag","true");

end
```

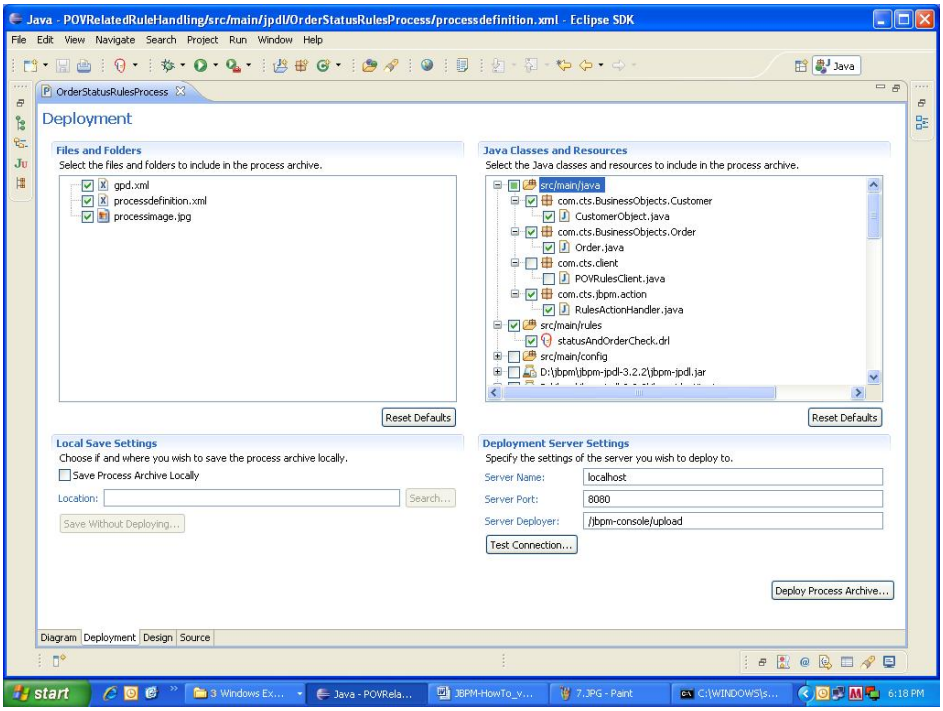
Steps	Description	Comments
Create a rule Resource (Contd...)	16. Create a rule action handler as follows.	

```
public class RulesActionHandler implements
ActionHandler {
      private static final long serialVersionUID = 1L;
      public List objectNames;
      public String ruleFile;
      public List queryStrings;
      /**
       * The RulesActionHandler gets variables from
the ContextInstance, and asserts
       * them into the Rules Engine and invokes the
rules.
       */
public void execute(ExecutionContext executionContext)
throws Exception {
            System.out.println("Rules action handler
class called");
            // load up the rulebase
            RuleBase ruleBase = readRule(ruleFile);
            WorkingMemory workingMemory =
ruleBase.newStatefulSession();
            // get an iterator of fully qualified
object names
            Iterator iter = objectNames.iterator();
            String objectName = "";
            ContextInstance ci =
executionContext.getContextInstance();
while (iter.hasNext()) {
                  objectName = (String) iter.next();

// assume the objects are stored as process variables
      Object object = ci.getVariable(objectName);
                  workingMemory.insert(object);

            }
// now assert the context instance as a global, so
that the rules
// can update the process, and fire the rules
          workingMemory.setGlobal("ci", ci);

            workingMemory.fireAllRules();

            if
(executionContext.getVariable("approvalFlag") ==
null){
executionContext.setVariable("approvalFlag", "false");
            }
        // propogate the token so that the process
continues
          executionContext.getToken().signal();
      }
```

Steps	Description	Comments

Create a rule Resource (Contd...)	<pre>/**
	* Please note that this is the "low level" rule assembly API.
	*/
	private static RuleBase readRule(String ruleFileName) throws IOException,
	DroolsParserException,
	RuleIntegrationException,
	PackageIntegrationException,
	InvalidPatternException, Exception {
	PackageBuilder builder = new PackageBuilder();
	builder.addPackageFromDrl(new InputStreamReader(
	RulesActionHandler.class.getResourceAsStream(ruleFileName)));
	RuleBase ruleBase = RuleBaseFactory.newRuleBase();
	
	ruleBase.addPackage(builder.getPackage());
	return ruleBase;
	}
	}</pre>	

Rules Integration	17. Copy all the rule related files into the existing jBPM project created in step-1 from the 'Rule project'.  	This step is done because currently jBPM Process Project and Rules Project come as separate Projects.

Steps	Description	Comments
Deploy Process	18. Deploy Process as mentioned in section-3.6. Before deployment necessary Java classes and Resources such as *.drl files must be included. 	Additionally before deployment the following entry is required in the '.classpath' file <classpathentry kind="src" path="src/main/rules"/> so that the rules folder can be read from the disk by the Rule Action Handler class.

Steps	Description	Comments
Rule Process Standalone Client to test the Rules integrated jBPM process		

```
jbpmConfiguration =
JbpmConfiguration.parseResource("default.jbpmConfiguration.cfg.
xml");
      JbpmContext jbpmContext =
jbpmConfiguration.createJbpmContext();
            try{
                  GraphSession graphSession =
jbpmContext.getGraphSession();
                  ProcessDefinition processDefinition =
graphSession.findLatestProcessDefinition("loanProcess");
                  System.out.println("Process Definition Id"  +
processDefinition.getId());
                  ProcessInstance instance = new
ProcessInstance(processDefinition);

      prepareTestData(instance.getContextInstance());
                  Token tok = instance.getRootToken();
                  tok.signal();
                  System.out.println("Flag is " +
instance.getContextInstance().getVariable("approvalFlag"));
                  jbpmContext.close();
            }catch(Exception ex){
            }
      }
      public static void prepareTestData(ContextInstance
contextInstance){
      com.cts.BusinessObjects.Order.Order order = new
com.cts.BusinessObjects.Order.Order(5000,56);
      CustomerObject customer = new
CustomerObject("Fred","Silver");
      contextInstance.setVariable("order", order);
      contextInstance.setVariable("customer", customer);
}
```

4.0 Annexure

4.1. Glossary

JPDL	jBPM Process Definition Language. JPDL specifies an xml schema and the mechanism to package all the process definition related files into a process archive http://docs.jboss.com/jbpm/v3/userguide/jpdl.html. This might also refer to Java Process Definition Language (http://developers.sun.com/learning/javaoneonline/j1sessn.jsp?sessn=TS-8612&yr=2007&track=7) Whether both refer to the same process definition language or have some common features is out of scope of the document.
The process archive	A process archive is a .par file. The central file in the process archive is processdefinition.xml. The main information in that file is the process graph. The processdefinition.xml also contains information about actions and tasks. A process archive can also contain other process related files such as classes, ui-forms for tasks,rules files etc.
State	A state defines a dependency on a result provided by an external party.
Finite State Machine	Is a model of behavior composed of a finite number of states, transitions between those states, and actions.
BAM/BI	Business Activity Monitoring/Business Intelligence
GOP	Graph Oriented Programming

5.0 References

Document	Source	Comments
JBoss jBPM jPDL 3.2	http://docs.jboss.com/jbpm/v3/userguide/	User Manual for jBPM v 3.2
jBPM Forum For Discussion	http://www.JBoss.com/?module=bb&op=viewforum&f=217	All queries may be raised regarding doubts concepts etc.
jBPM JIRA Issue	http://jira.JBoss.org/jira/browse/jBPM	An issue may be raised against a bug, enhancements etc.
jBPM v5.7 Installation guide	http://www.JBoss.org/wiki/Wiki.jsp?page=JbpmWiki	Wiki's jBPM Installation Guide
Best Practices for Exception Handling	http://www.onjava.com/pub/a/onjava/2003/11/19/exceptions.html?page=1	Best Practices for Java Exception Handling
Hibernate tutorial	http://www.hibernate.org/hib_docs/reference/en/html/tutorial.html	jBPM Persistence concepts are related to Hibernate