BONITA Workflow Cooperative System
User's Guide

Miguel Valdés Faura
INRIA Lorraine (ECOO TEAM)

Index

Introduction

This document is based on the new Workflow model proposed by the ECOO Team for applications supporting business processes.
This approach incorporates the anticipation of activities as a more flexible mechanism of workflow execution. This flexibility allows a considerable increase of speed in the design and development phases of cooperative applications

The development of the Bonita Workflow engine is built on EJB (Enterprise Java Beans) technology that is the server-side component architecture for the J2EE (JavaTM 2 Platform, Enterprise Edition). EJB enables rapid and simplified development of distributed, transactional, secure and portable Java applications.
We have chosen JOnAS application server ((http://jonas.objectweb.org/) in order to deploy Bonita Cooperative Workflow System.

Bonita Workflow includes also an 100% browsed based application example created with the Struts Framework that provides a simple environment to define and control the workflow processes by means of your favourite browser. In the same way the use of the Java Web Start Applications and the Bonita Web Services technology allows the users and the business organisations to generate your web representation of Bonita Worklow System.

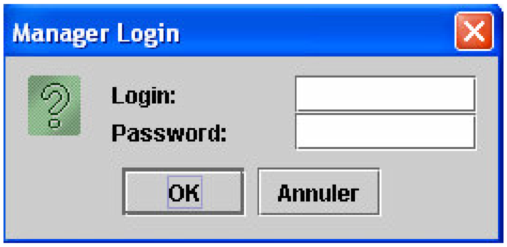
The principal objective of this guide is an initiation to the generic model of Bonita Workflow System dedicated to specify, execute, monitor and coordinate the organizations flow of work. Bonita offers a comprehensive set of graphical tools integrated into a single package in order to perform process conception and definition, the instantiation and control of this process and the interaction with the users and other applications.

These applications compose the Bonita Workflow Management System, known as “Manager” and formed by the Bonita Workflow Definition Component (“GraphEditor” application) and the Bonita Workflow Execution Component (Worklist application).
The Bonita Workflow Management System are designed in order to facilitate the definition and execution of workflow processes.

In the next sections we explains the basic functionalities of Bonita Workflow Management applications and we illustrate these by means of some workflow examples.

Chapter 1. Worklist application

When the user launches the *manager* application (by clicking on the manager link at Bonita's Web Menu or by typing ant manager in your command line) the next authentication screen appears:



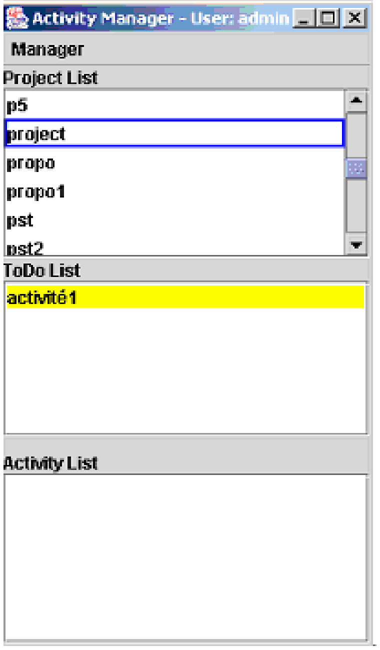
If the user is correctly logged in the system, the Worklist application is shown. The User Worklist application allow the users to control the process execution and provides different information about the projects of every user. This information is organized in three list: *Project List, ToDo List* and the *Activity List*.

When the user select one project from its project list, he obtains the list of executing activities (executing or anticipating states) and his assigned activities (ready or anticipable).

The user Worklist provides different information:

Project List	List of user projects. In this list the user can see his projects and he can select one of these in order to executes his assigned activities. The user can also edit the project by double-clicking and launch the Bonita Workflow Definition Component.
Todo List	List containing ready and anticipable activities of the current project associated with a user. These activities are divided in two different colours: yellow for the activities in ready state and green for the activities in anticipable state. The important difference between these two activities types is the execution mode. The ready activities are executed in a traditional workflow model and the anticipable activities follows the Bonita flexible model.
Activity List	List of executing and anticipating activities of the current project . These activities are also divided in two different colours: red for the activities in execution state and violet for the activities in anticipating state. Only the activities started by the user are showed here.

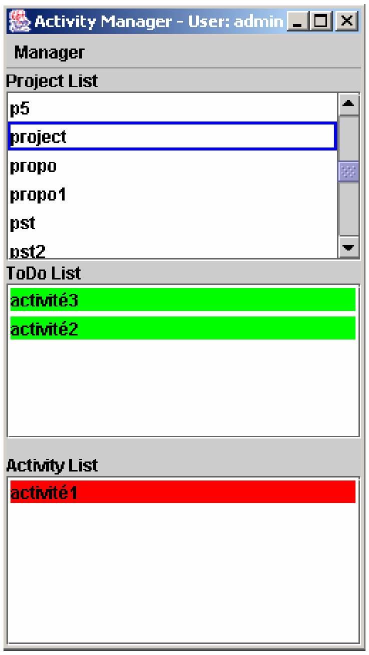
The next figures show the state changes when the user begins an activity:



p5
project
propo
propo1
pst
pst2

activité1

Figure 1 : Activities of the project

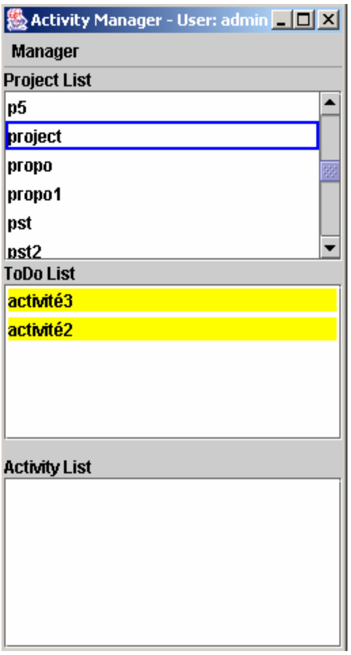


p5
project
propo
propo1
pst
pst2

activité3
activité2

activité1

Figure 2: Execution of activity 1

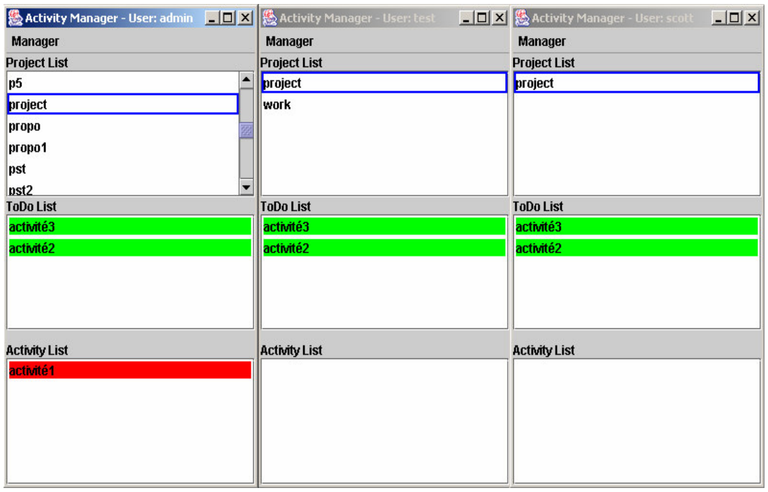


p5
project
propo
propo1
pst
pst2

activité3
activité2

Figure 3: Activity 1 terminated

The following example shows an execution environment with three users (*admin*, *test* and *scott*). The user *admin* executes the first activity and the user worklists are automatically updated (*Figure 3*). Only the user who has executed the activity is able to finish it.



p5
project
propo
propo1
pst
pst2

activité3
activité2

activité1

project
work

activité3
activité2

project

activité3
activité2

Figure 3: User *Worklists* when the user *admin* executes the first activity

If the user *test* would like to anticipate the second activity (with anticipable state in the TodoList), the manager application updates all TodoLists and the user's ActivityList (*Figure 4*).

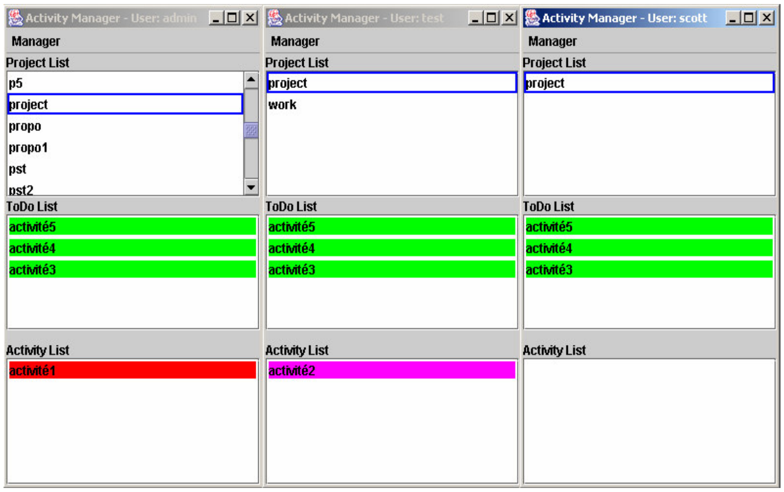
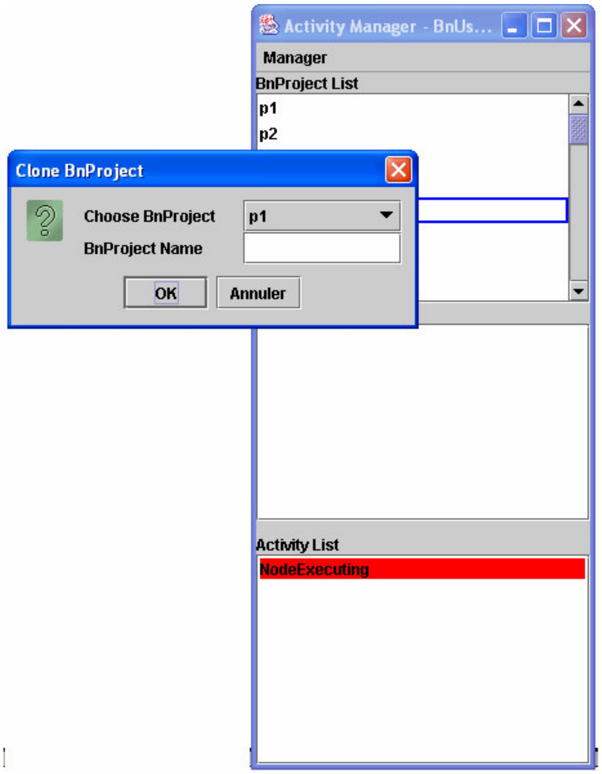


Figure 4: User Worklists when the user *test* anticipates the second activity

Every user, provided that he is correctly authenticated, can execute, terminate and anticipate activities. With the *manager* menu the user may create a new project or clone existing projects and also ask for the project and activities information.

Create new project and Clone existing project functionalities, allow the users to begin the definition of a new Bonita Workflow process.

With the Clone project option we can use an existing Bonita process in order to create the new process. The new project contains the same activity graph (at the initial state) and copy the hooks and roles of the previous one.



Choose BnProject: p1

BnProject Name:

Figure 5: User Worklists when the user clones an existing project

Chapter 2. GraphEditor application

This application allows the users to define the workflow process. This definition includes activity definition, activities connection, users and roles definitions…

The user can run the application by double-clicking on one project of a previous *manager* application.

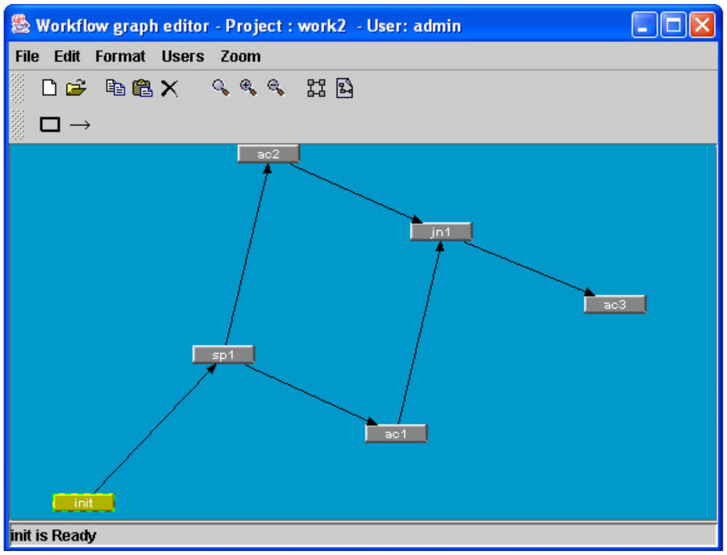


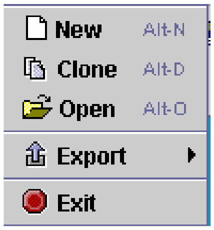
Figure 5: GraphEditor when the user opens an existing project.

This is the first screen that appears when any user opens a project.

The GraphEditor application provides several visual definition functionalities to make the workflow definition easy and automatic.

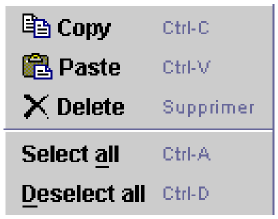
Using the GraphEditor Menu

File Menu



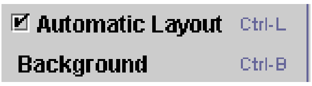
- **New**: Creates a new empty project.
- **Clone**: Creates a new project with activities and edges from an existing project.
- **Open**: Opens an existing project.
- **Export**: Stores the graphical workflow representation into different image formats (png, jpeg, jpg).
- **Exit**: Finishes the GraphEditor session

Edit Menu



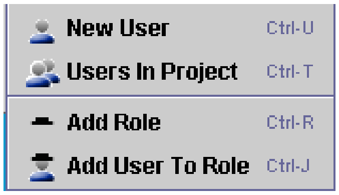
- **Copy**: Copies selected activity/activities and edges of the project.
- **Paste**: Pastes activity/activities and edges in the project.
- **Paste:** Deletes selected activity/activities and edges of the project.
- **Select all:** Selects all activities and edges of the project.
- **Deselect all:** Deselects all activities and edges of the project.

Format Menu



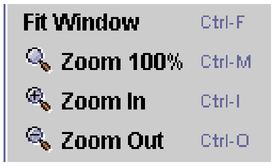
- **Automatic Layout**: Activates the GraphEditor layout to place automatically all activities in the screen.
- **Background**: Selects the background colour.

Users Menu



- **New User**: Adds a system user to this project.
- **Users In Project**: Lists the users of this project.
- **Add Role:** Adds new role to the system.
- **Add User to Role:** Assigns a new role to a user.

Zoom Menu



- **Fit Window**: Fits the workflow graph in the window..
- **Zoom 100%**: Applies a 100% zoom to the graph.
- **Zoom In:** Applies zoom in to the graph.
- **Zoom Out:** Applies zoom out to the graph.

Activity Colors

With the GraphEditor application the user can visualize the Workflow process state. A Bonita Workflow process is composed by the activities and the connections between these activities. The user can identify the workflow execution by means of the activities color changes. Each activity has a color associated with the state of this activity.
The next table shows the different activity colors:

Color	Activity State
Yellow	Initial state. Only for the start point activity and for the activities with no connections.
Green	Anticipable state. You can executes these activities in anticipation mode.
Red	Executing state. The activity is executing.
Violet	Anticipating state. The activity has been executed in anticipation mode.
Light Blue	Terminated state. The activity is finished.
Dark Blue	Cancel state. The activity was cancelled

Activity Information

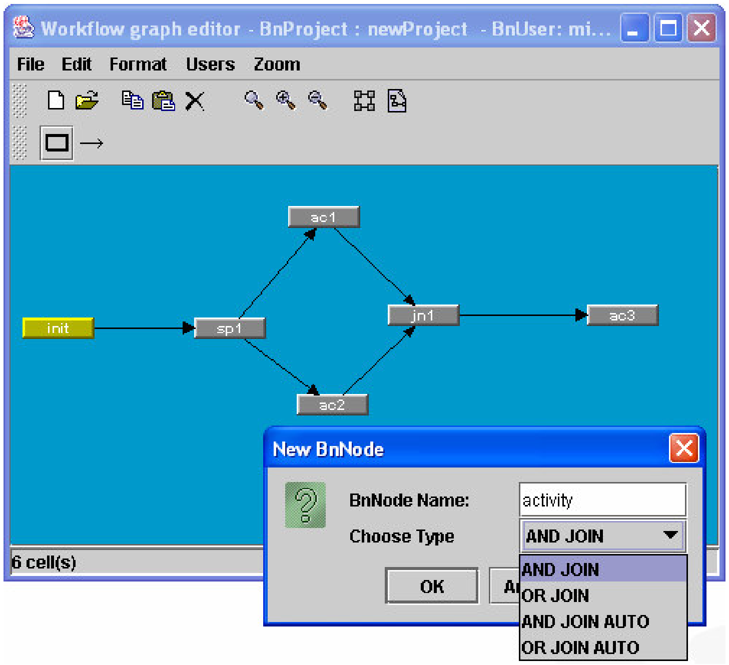
You can move your mouse over the activities to obtains a little description: activity name, activity state, activity deadline, activity role…

Activity Routing

Bonita Workflow System provides an integrated routing activity model, so each activity integrates the process execution point of control. The routing modes available in Bonita are: AND-JOIN, OR-JOIN, AND-SPLIT and OR-SPLIT. These routing modes can be used in a flexible (activities anticipation) or traditional (isolation activities) execution modes. The activities execution control can be effectuated by the user or automatically by the execution engine (non-automatic/automatic activities).

When the user adds a new activity to the workflow process he must select the activity routing mode. This routing mode will be AND JOIN or OR JOIN to make available the control execution to the user or AND JOIN AUTO or OR_JOIN AUTO to give the control to the execution engine.

In order to insert AND SPLIT and OR SPLIT routing modes the user must define the activity connections (edges) conditions, so first he chooses AND JOIN or OR SPLIT activity routing nodes and then he edits edges to define the split condition.



BnNode Name: activity

Choose Type: AND JOIN

Figure 6: GraphEditor when the user adds a new activity.

The next figures shows the workflow process execution when the user selects AND JOIN routing mode:

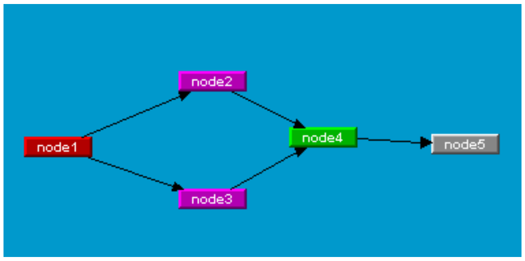


Figure 7: Workflow execution when the activity "node4" with AND JOIN routing mode is active (anticipable state) by two anticipating activities.

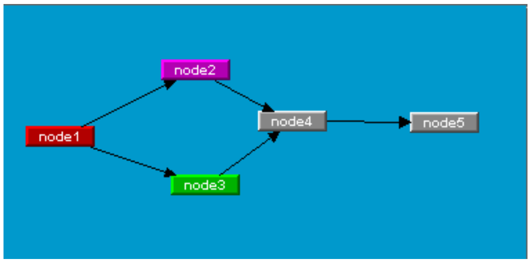


Figure 8: Workflow execution when the activity "node4" with AND JOIN routing mode is waiting (initial state) for the execution of activity "node3"

The next figures shows the workflow process execution when the user selects OR JOIN routing mode:

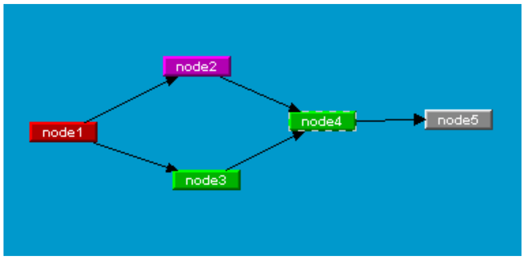


Figure 9: Workflow execution when the activity “node4” with OR JOIN routing mode is active (anticipable state) by the activity “node2”.

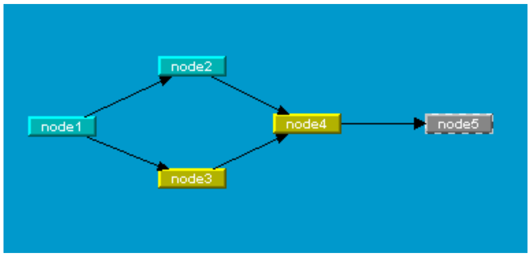
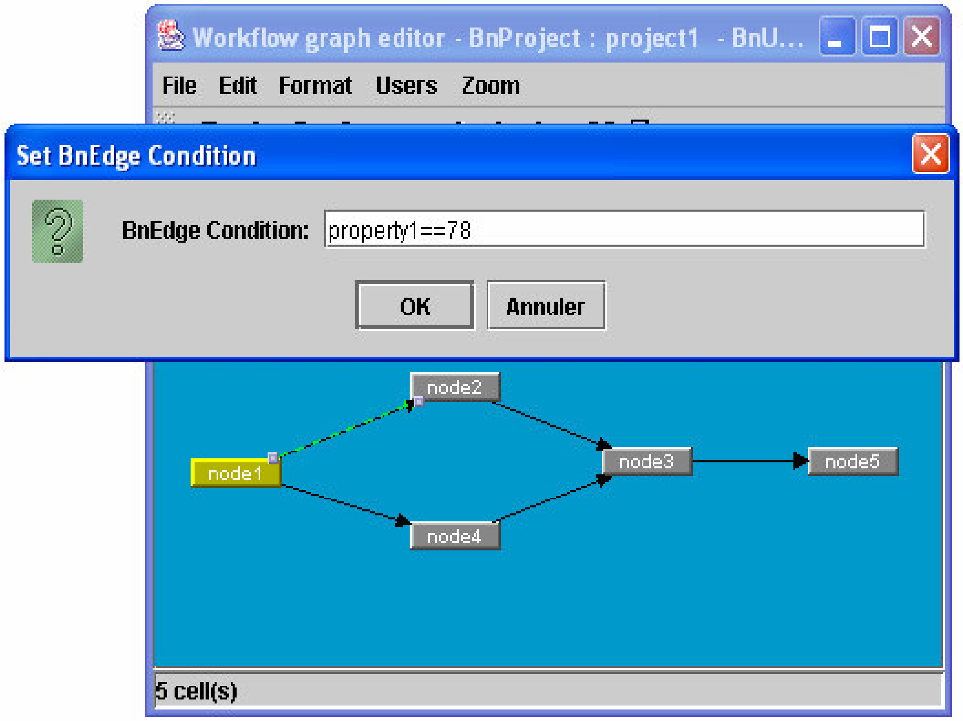


Figure 10: Workflow execution when the activity “node4” with OR JOIN routing mode is waiting (ready state) for the user execution.

In the next figure the user defines activities with AND SPLIT or OR SPLIT routing modes by setting edges conditions. When the user terminates an execution activity, the Bonita execution engine evaluates the activity out edges conditions in order to propagate or not this event to connected activities.

Normally, edges conditions uses the activities properties values to select the progress of the workflow process execution. In the example, when the activity node1 is finish, the execution engines evaluates out edges conditions and if the value of the property “property1” is equal to “78” the system actives the activity “node2”.



BnEdge Condition: property1==78

Figure 11: GraphEditor when the user set an edge condition.

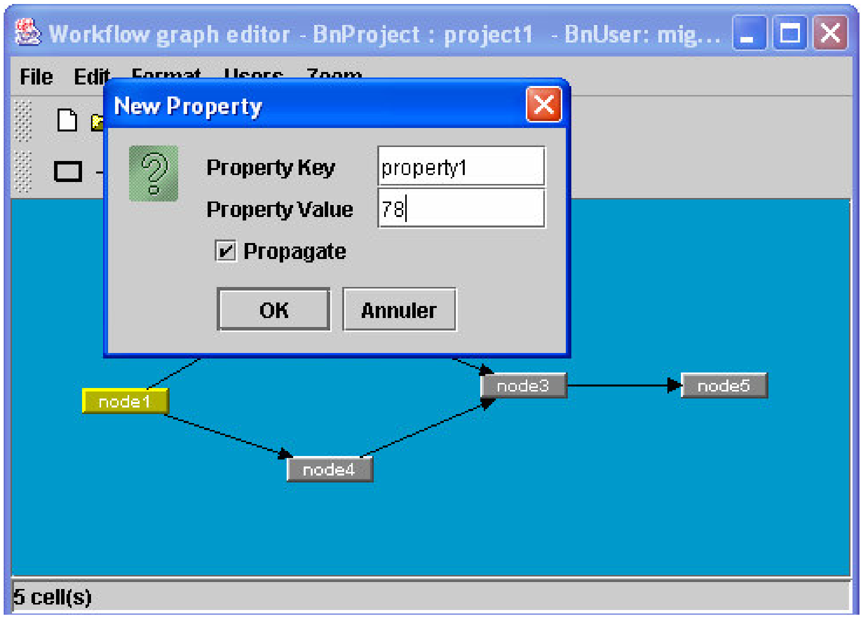
The Bonita integrated routing mode allows the user to select between AND JOIN or OR JOIN routing modes and by default AND SPLIT execution propagation is set. So, if you want to control this propagation you can set edge conditions to implement the OR SPLIT routing mode.

Activity Properties

Bonita Cooperative Workflow uses the concept of properties to control the process execution data. Each Bonita Workflow process can be many properties which contains the process execution data. These properties can be attached to the workflow process and/or the process activities. The Flexible execution engine of Bonita allows the users to interchange intermediate data/results. This flexibility is possible by means of the anticipation activity model and the activity properties takes and important role.

If the user wants to add a new property to an activity, he must select the option Add Property by clicking on the right mouse button over the activity. The “new property” dialog is composed by the property key and value fields and the propagate Boolean. This Boolean allows the property propagation when the associated activity is executed.

The use of the activity properties in anticipation mode offers many possibilities to evaluate intermediate results of the anticipating activities and it can be used to do backtracking operations.



Property Key: property1
Property Value: 78
☑ Propagate

Figure 12: GraphEditor when the user set a new activity property.

We continue to talk about the properties in section “Activity Hook”, and we explain the steps to create this properties dynamically at run time.

Edit Activity

With the GraphEditor application you can edit and modify some activity parameters. These parameters are the activity deadline, the activity role and the activity description. By setting these parameters the user defines the end date of the activity, the users that can execute activity and the list of steps that compose the activity.

If the user wants to edit an activity, he must select the option Edit Activity by clicking on the right mouse button over the activity.

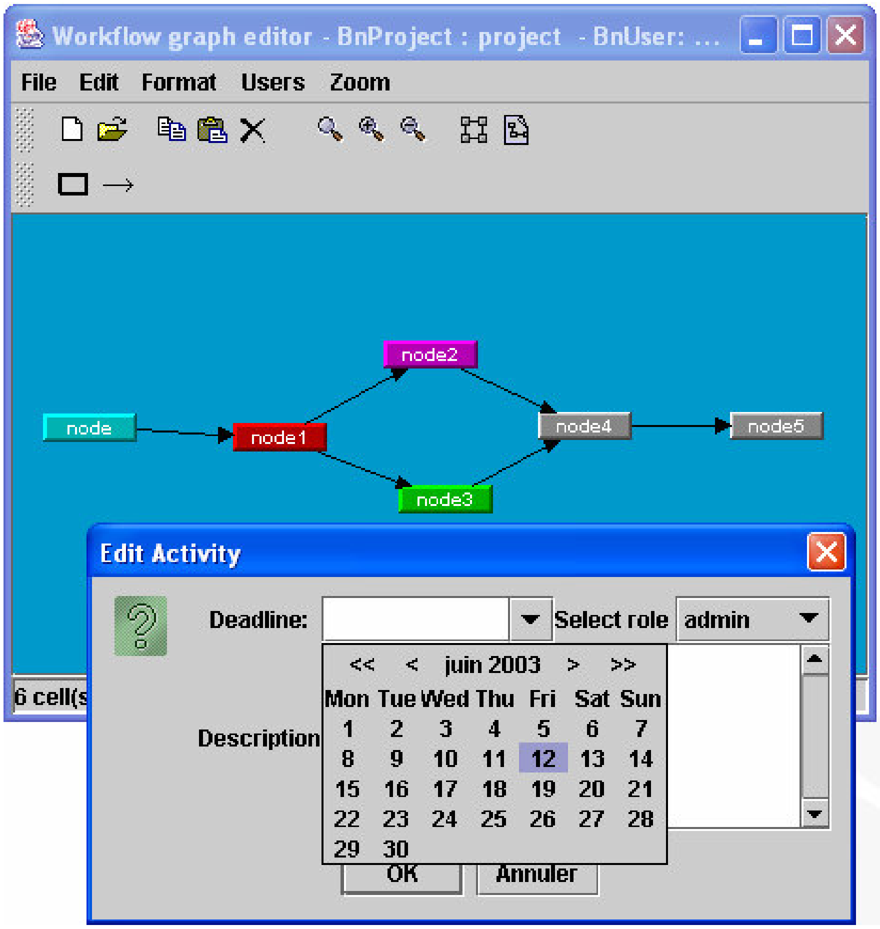


Figure 13: GraphEditor when the user edit the activity parameters.

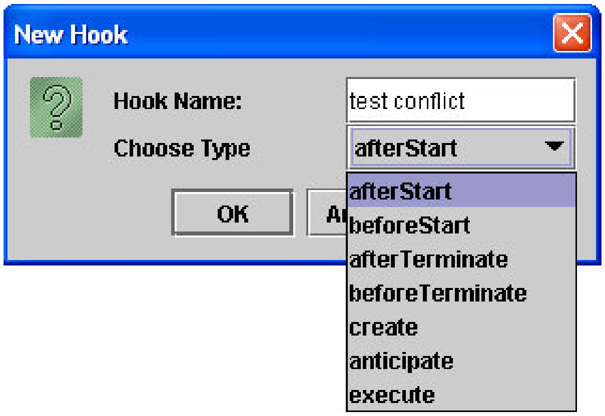
When the user performs the workflow process definition he usually sets the activity deadline value for each activity. The Bonita Workflow Systems uses the timer service in order to add the new deadline and it sends a notification event to the client in order to warn the user that the activity must be finished. This notification could be an email or a jabber instant message.

All the activities in a workflow process has assigned a role. Only the users that has the same role can start the activity, so only the users that has this role will see these activities in the Manager application todo list.

Activity Hooks

The activity hooks are units of source code, associated to process activities, that will be executed at runtime by Bonita execution engine. These hooks should be written in Java or in one of the object scripting languages available in Bonita (TCL, BeanShell).

With the GraphEditor application, the user can set and define activity actions by means of the BeanShell hooks.
If the user wants to set a new hook to an activity, he must select the option New Hook by clicking on the right mouse button over the activity and the next dialog window will be shown.



Hook Name: test conflict

Choose Type afterStart

Figure 14: GraphEditor when the user adds new hook.

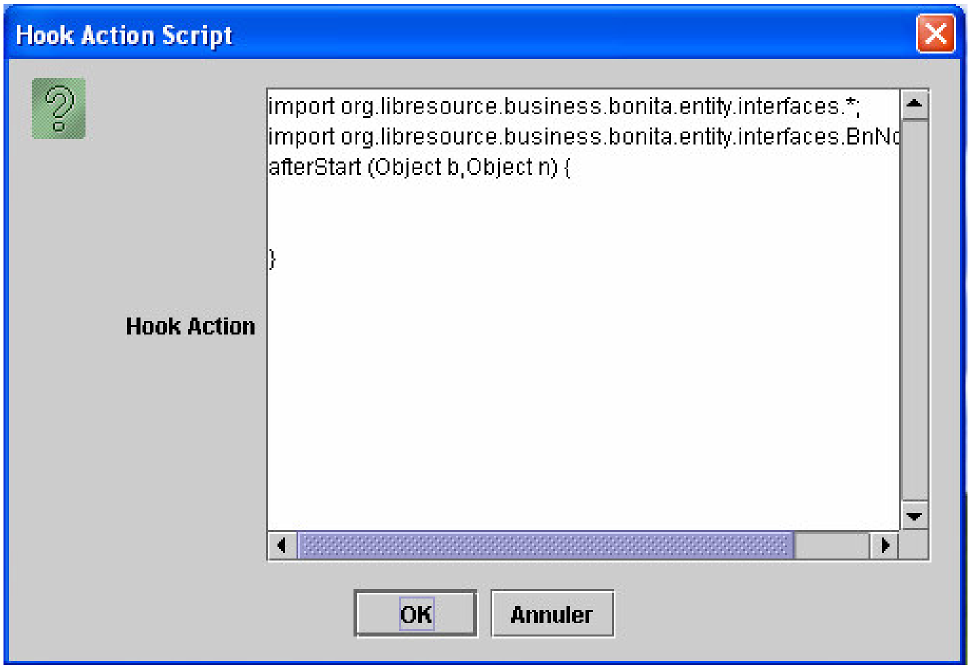
In this dialog window, the user must insert the name of the hook and he must choose the hook type. This hook type explains the hook execution moment of the activities life cycle : before start, after start, create, before terminate, after terminate, anticipate, create and execute.

The previous operation creates the activity hook on the system. Now its time to define the actions of this hook. The user can use all Bonita functionalities in order to define the hooks actions, so all the Bonita EJB are accessible by the hook interface.

If you wants to use external functionalities in your hook actions, like external web services or other applications, you can do it because you can access to all classes available in the classpath of the application server used to deploy Bonita.

When the user inserts the hook name and selects one of the possible hook types, the Hook Action Editor appears. This dialog window, shows the default classes imports and the action type function associated to this hook. Within this function the user can defines the actions of this hooks in Java. These actions will be executed at run time by the execution engine in one moment of the activity execution.

The next figure shows the Hook Action Editor when the user sets a new afterStart hook:



```
import org.libresource.business.bonita.entity.interfaces.*;
import org.libresource.business.bonita.entity.interfaces.BnNo
afterStart (Object b,Object n) {


}
```

Figure 15: GraphEditor when the user defines the hook actions.

Sample Application

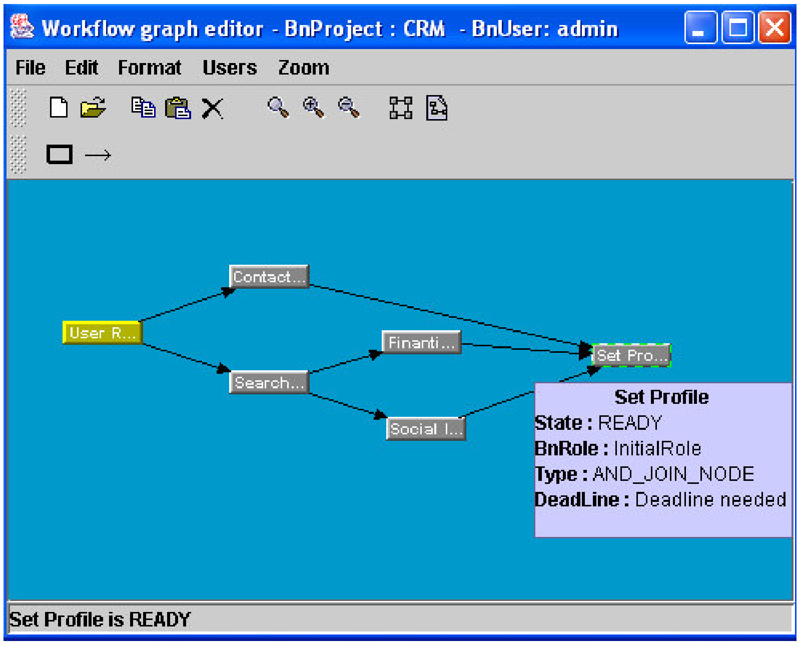
In order to explain the hook functionalities we can take an example of workflow application that use activity properties and hooks.
We can imagine a basic CRM system for an e-commerce organisation. In this organisation different employees of the client relationship department work together in order to obtain the profiles of the users which access to the system.

The Bonita Workflow representation of this CRM system are composed by six activities: User Registration, Contact User, Search Information, Financial Information, Social Information and Set Profile.
This workflow process, allows the organisation to sets the user profile (the last activity of the process) while is performing the registration (the first activity of the process). For that, the CRM system must obtain some user's information in order to present the best products lines for him.

The learning information process about the user will be performed by the other activities in the Workflow representation. For example, we can try to contact the user by email or telephone from evaluation of some of the basic user information: its age, its nationality…
We can use also delegate to other employees the responsibility of search specific information about this user: financial information or social information to fits the user's needs products.

So, with Bonita we can integrate a great variety of systems, for example this basic CRM system, in order to define and control the execution phases of the process.

The next figure shows the activities which compose the CRM workflow representation in order to set the user profile.

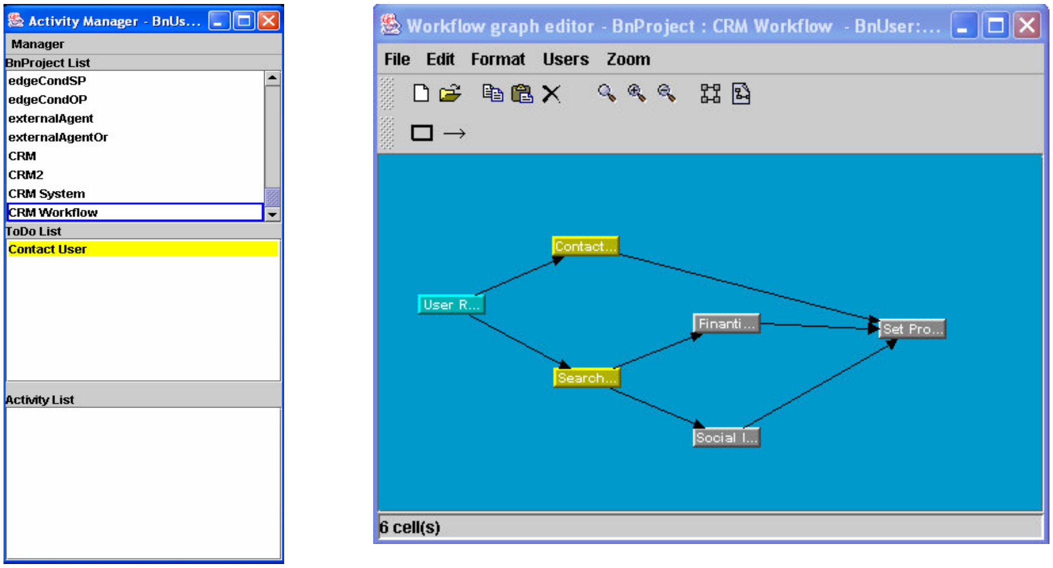

Workflow graph editor - BnProject : CRM - BnUser: admin
File Edit Format Users Zoom
Contact...
User R...
Finanti...
Set Pro...
Search...
Social I...

Set Profile
State : READY
BnRole : InitialRole
Type : AND_JOIN_NODE
DeadLine : Deadline needed

Set Profile is READY

Figure 16: Workflow Representation of the CRM System.

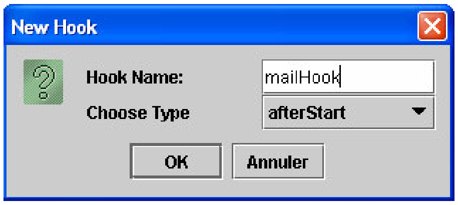
We can suppose the first activity "User Registration" is associated to a form web page. This page is the new user account page, so the CRM workflow example starts when the user submits his personal information: first name, second name, age and nationality. After that, the first activity is automatically executed and terminated and the "Contact User" and "Search User" activities are activated (ready state). See the next figures:



6 cell(s)

Figure 17: Worklist and GraphEditor application for the CRM System example. The "Contact User" activity is associated to the employee "Charles" in the todo list.

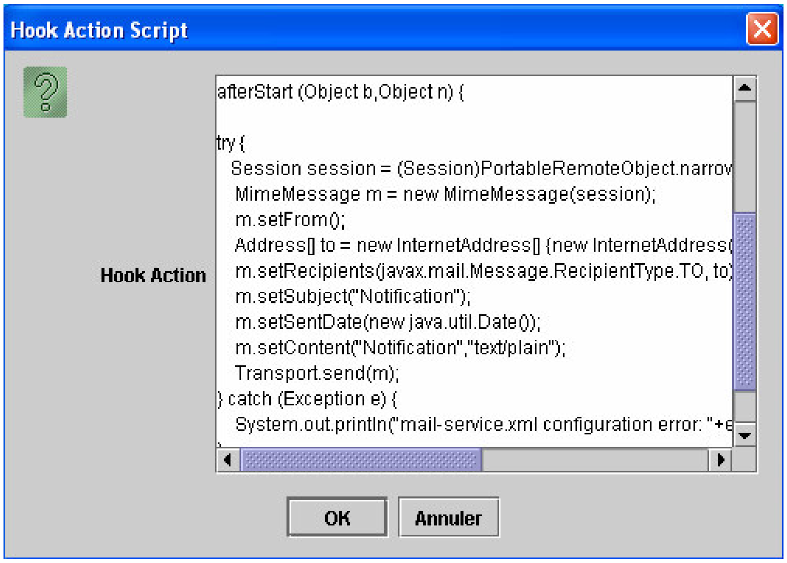
Now, the user “Charles” can execute the activity “Contact User”, and for example this activity sends an email to the user in order to present one of the best products of the current month. To do that, the user administrator has set the following hook to this activity:



Hook Name: mailHook

Choose Type: afterStart

```
import javax.naming.InitialContext;
import javax.mail.Session;
import javax.mail.Address;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.rmi.PortableRemoteObject;
afterStart(Object b, Object n)
{
  try {
    Session session =
(Session)PortableRemoteObject.narrow(new
InitialContext().lookup("java:/Mail"), Session.class);

    MimeMessage m = new MimeMessage(session);
    m.setFrom();
    Address[] to = new InternetAddress[] {new
InternetAddress(usermail) };
    m.setRecipients(javax.mail.Message.RecipientType.TO, to);
    m.setSubject("Notification");
    m.setSentDate(new java.util.Date());
    m.setContent("Notification","text/plain");
    Transport.send(m);

  } catch (Exception e) {
     System.out.println("mail-service.xml configuration error:
"+e);
  }
}
```



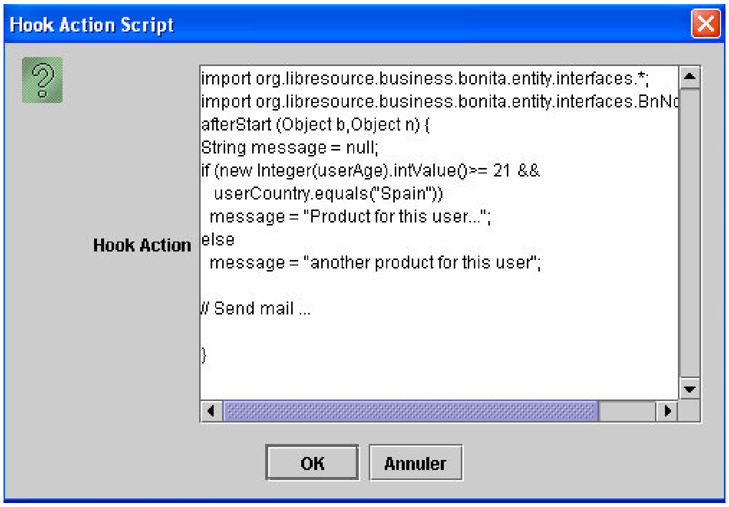
```
afterStart (Object b,Object n) {

try {
  Session session = (Session)PortableRemoteObject.narro
   MimeMessage m = new MimeMessage(session);
   m.setFrom();
   Address[] to = new InternetAddress[] {new InternetAddress
   m.setRecipients(javax.mail.Message.RecipientType.TO, to)
   m.setSubject("Notification");
   m.setSentDate(new java.util.Date());
   m.setContent("Notification","text/plain");
   Transport.send(m);
} catch (Exception e) {
   System.out.println("mail-service.xml configuration error: "+e
```

* Note: In the previous Java source code we find the attribute usermail. This attribute may be a user property set when the user registration account was submitted.

In Bonita, all activities properties will be propagated and mapped to the next activities, so if we defined the usermail property in “User Registration” activity, the “Contact User” activity can access it.

Figure 18: After start hook assigned to “Contact User” activity that send an email to the user.

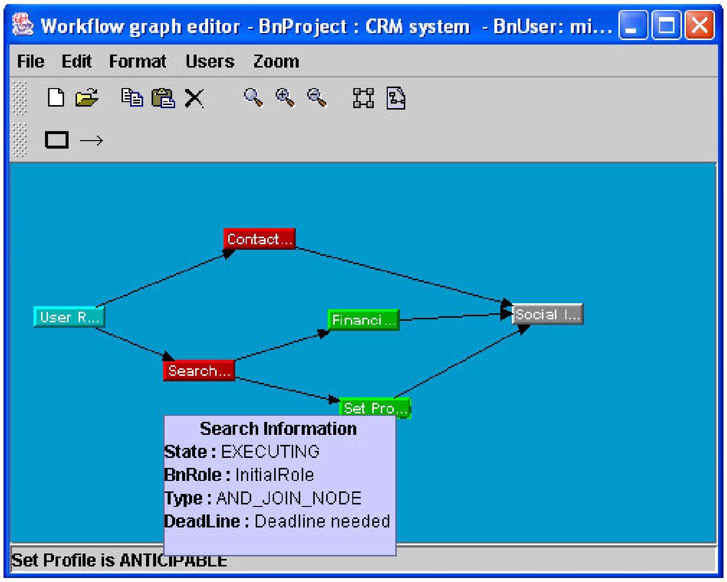
If we want to present to the user only the products that fits his needs, we can filtrate the basic user information before sending the email.



```
import org.libresource.business.bonita.entity.interfaces.*;
import org.libresource.business.bonita.entity.interfaces.BnN
afterStart (Object b,Object n) {
String message = null;
if (new Integer(userAge).intValue()>= 21 &&
  userCountry.equals("Spain"))
  message = "Product for this user...";
else
  message = "another product for this user";

// Send mail ...

}
```

Figure 19: After start hook assigned to “Contact User” activity with filtration of user information.

At the same time that the “Contact User” activity was assigned to the user “Charles”, the “Search Information” activity was appeared in the todo list of another user, for example “Guillaume”. This user is the employee which control the user information search, so if he decides to start this, the connected activities becomes anticipable: “Financial information” and “Social Information”.



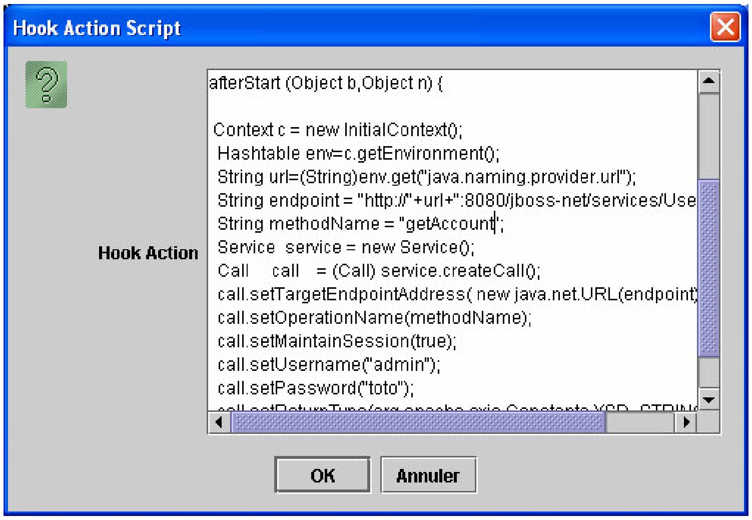
Search Information
State : EXECUTING
BnRole : InitialRole
Type : AND_JOIN_NODE
DeadLine : Deadline needed

Set Profile is ANTICIPABLE

Figure 20: CRM workflow system when the user Guillaume executes “Search Information” activity.

Now, the responsible employees of activities “Financial Information” and “Social Information” can anticipate the execution of these activities in order to find interesting information about the user. This information could be useful to allow the organisation to propose to the user the products that better fits his profile.

If we focus the attention in the “Financial Information” activity, we can suppose the employee involved in this activity uses the external systems functionalities to obtain information about user bank accounts. In order to access to these functionalities the employee can use, for example, web services technology.



```
afterStart (Object b,Object n) {

 Context c = new InitialContext();
  Hashtable env=c.getEnvironment();
  String url=(String)env.get("java.naming.provider.url");
  String endpoint = "http://"+url+":8080/jboss-net/services/Use
  String methodName = "getAccount";
  Service  service = new Service();
  Call    call   = (Call) service.createCall();
  call.setTargetEndpointAddress( new java.net.URL(endpoint)
  call.setOperationName(methodName);
  call.setMaintainSession(true);
  call.setUsername("admin");
  call.setPassword("toto");
```

After that, the user responsible of the last activity "Set User Profile" inserts previous user information into a database to obtain the first user profile, and the CRM Workflow example is finished.

Hook API

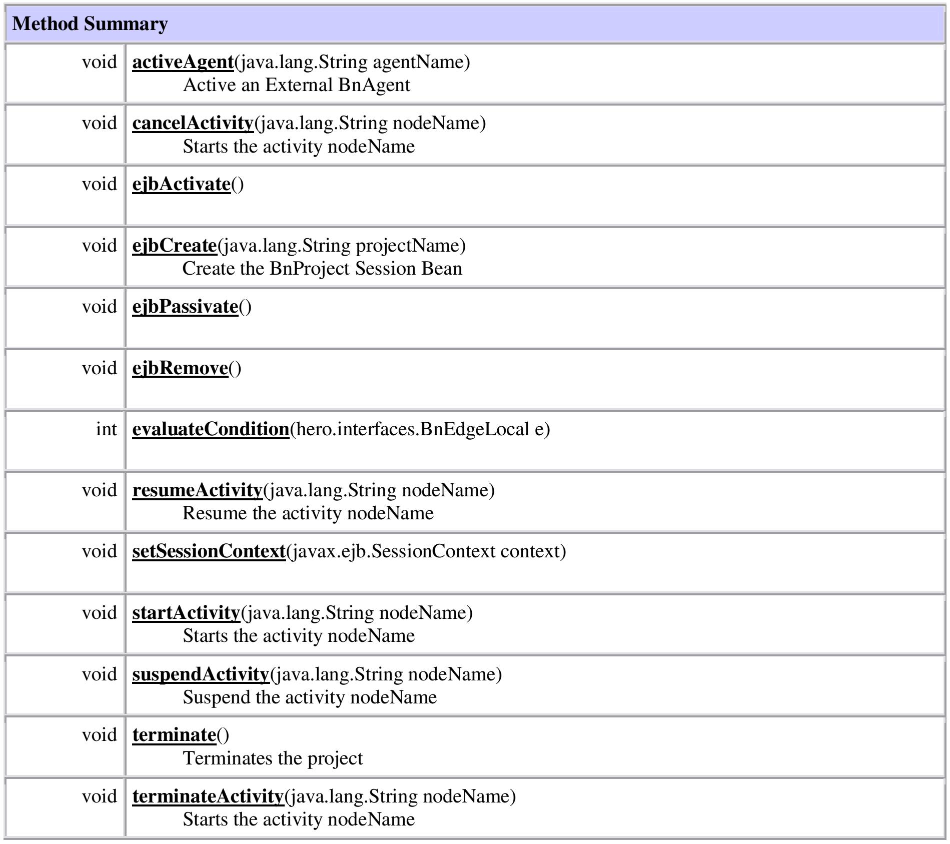
If we take a look to the hook Interface, we can see that each hook method: afterStart, beforeStart… receives two parameters: Object b (EngineBean Session) and Object n (NodeLocal Object). We can use these objects to obtain information about the activity associated to this hook (by using the BnNodeLocal API) and to control the process execution (by using EngineBean Session Bean API).

BnNodeLocal Object methods:

Method Summary	
hero.entity.EdgeState	**getActivation**() Retrieve the BnNode's Activation type.
boolean	**getAnticipable**() Retrieve the BnNode's anticipation mode.
java.util.Collection	**getBnHooks**()
java.util.Collection	**getBnInterHooks**()
hero.interfaces.BnNodeLightValue	**getBnNodeLightValue**()
hero.interfaces.BnNodeValue	**getBnNodeValue**()
hero.interfaces.BnProjectLocal	**getBnProject**()
java.util.Collection	**getBnProperties**()
hero.interfaces.BnRoleLocal	**getBnRole**() Retrieve the BnNode's BnRole.
java.sql.Date	**getCreationDate**()
hero.interfaces.BnUserLocal	**getCreator**() Retrieve the BnNode's Creator.
java.sql.Date	**getDeadline**() Retrieve the BnNode's Deadline.
java.lang.String	**getDescription**() Retrieve the BnNode's Description.
java.sql.Date	**getEndDate**() Retrieve the BnNode's EndDate.
hero.interfaces.BnUserLocal	**getExecutor**() Retrieve the BnNode's Executor.
int	**getId**() Retrieve the BnNode's id.
java.util.Collection	**getInBnAgentEdges**()
java.util.Collection	**getInBnEdges**()
java.sql.Date	**getModificationDate**()
java.lang.String	**getName**() Retrieve the BnNode's Name.

java.util.Collection	**getOutBnEdges**()
java.sql.Date	**getStartDate**() Retrieve the BnNode's StartDate.
int	**getState**() Retrieve the BnNode's state.
hero.entity.NodeState	**getTransition**() Retrieve the BnNode's Transition type.
int	**getType**() Retrieve the BnNode's type.
boolean	**isCancelled**()
boolean	**isExecuting**()
boolean	**isTerminated**()
void	**setActivation**(hero.entity.EdgeState activation) Set the BnNode's Activation
void	**setAnticipable**(boolean pAnticipable) Set the BnNode's Anticipable mode.
void	**setBnHooks**(java.util.Collection pHook) Set the Hooks of the node
void	**setBnInterHooks**(java.util.Collection pHook) Set the Hooks of the node
void	**setBnNodeValue**(hero.interfaces.BnNodeValue v)
void	**setBnProperties**(java.util.Collection pPrp) Set the properties of the node
void	**setBnRole**(hero.interfaces.BnRoleLocal role) Set the BnNode's Creator.
void	**setCreationDate**(java.sql.Date pDate)
void	**setCreator**(hero.interfaces.BnUserLocal pCreator) Set the BnNode's Creator.
void	**setDeadline**(java.sql.Date pDeadline) Set the BnNode's Deadline.
void	**setDescription**(java.lang.String pDescription) Set the BnNode's Description.
void	**setEndDate**(java.sql.Date pEndDate) Set the BnNode's EndDate.
void	**setExecutor**(hero.interfaces.BnUserLocal pExecutor) Set the BnNode's Creator.
void	**setModificationDate**(java.sql.Date pDate)
void	**setStartDate**(java.sql.Date pStartDate) Set the BnNode's StartDate.
void	**setState**(int pState) Set the BnNode's state.
void	**setTransition**(hero.entity.NodeState transition) Set the BnNode's Transition
void	**setType**(int pType) Set the BnNode's type.

EngineBean Session Bean methods:



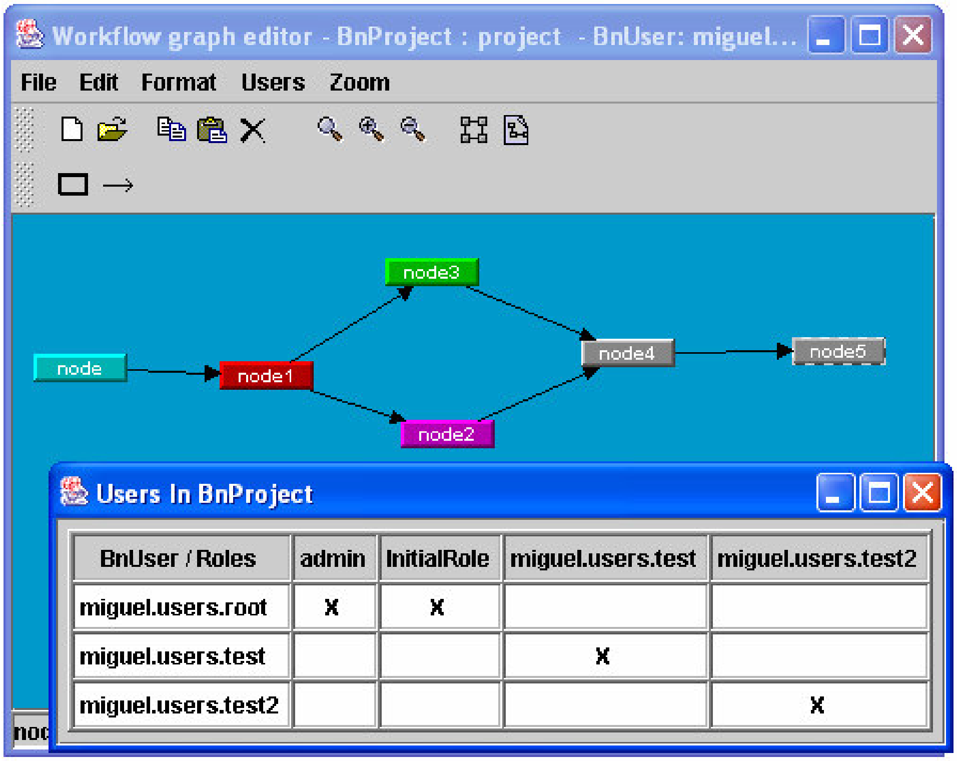
Method Summary	
void	**activeAgent**(java.lang.String agentName) Active an External BnAgent
void	**cancelActivity**(java.lang.String nodeName) Starts the activity nodeName
void	**ejbActivate**()
void	**ejbCreate**(java.lang.String projectName) Create the BnProject Session Bean
void	**ejbPassivate**()
void	**ejbRemove**()
int	**evaluateCondition**(hero.interfaces.BnEdgeLocal e)
void	**resumeActivity**(java.lang.String nodeName) Resume the activity nodeName
void	**setSessionContext**(javax.ejb.SessionContext context)
void	**startActivity**(java.lang.String nodeName) Starts the activity nodeName
void	**suspendActivity**(java.lang.String nodeName) Suspend the activity nodeName
void	**terminate**() Terminates the project
void	**terminateActivity**(java.lang.String nodeName) Starts the activity nodeName

Within the code of our hook we can also use Bonita Workflow functionalities in order to define and control Bonita workflow processes, so Bonita API is available to be used in the hooks. In the same context, you can call your Java applications and objects from this hook environment.

Users in Project

Like in the others Workflow Management Systems, Bonita allows to some users participate in the definition and execution of a workflow process. The Bonita Workflow Definition Component (GraphEditor) controls the users that can access to the process definition.

If the user wants to know the list of the members and roles of this project, he must select the option "Users in Projects" by clicking on the right mouse button over the GraphEditor panel.

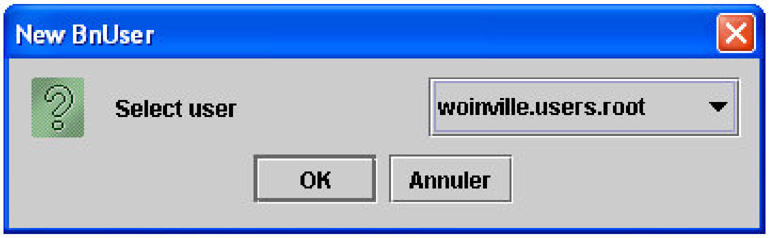


BnUser / Roles	admin	InitialRole	miguel.users.test	miguel.users.test2
miguel.users.root	X	X		
miguel.users.test			X	
miguel.users.test2				X

Figure 15: GraphEditor when the user clicks to the Users in Project option.

Add User to Project

In order to incorporate and existing user to a specific Bonita project, the user must select the option "New User" by clicking on the right mouse button over the GraphEditor panel.



Select user: woinville.users.root

Figure 16: GraphEditor when the user clicks to the Add User option.

Add Project Role

Roles in Bonita represents the users who take part into definition and execution of the workflow process. Each role in Bonita is associated to a project/process, so the role "admin" in project "test" is different than the role "admin" in project "test2".

If the user wants to insert a new role in a specific project, he must select the option "Add Role" by clicking on the right mouse button over the GraphEditor panel.

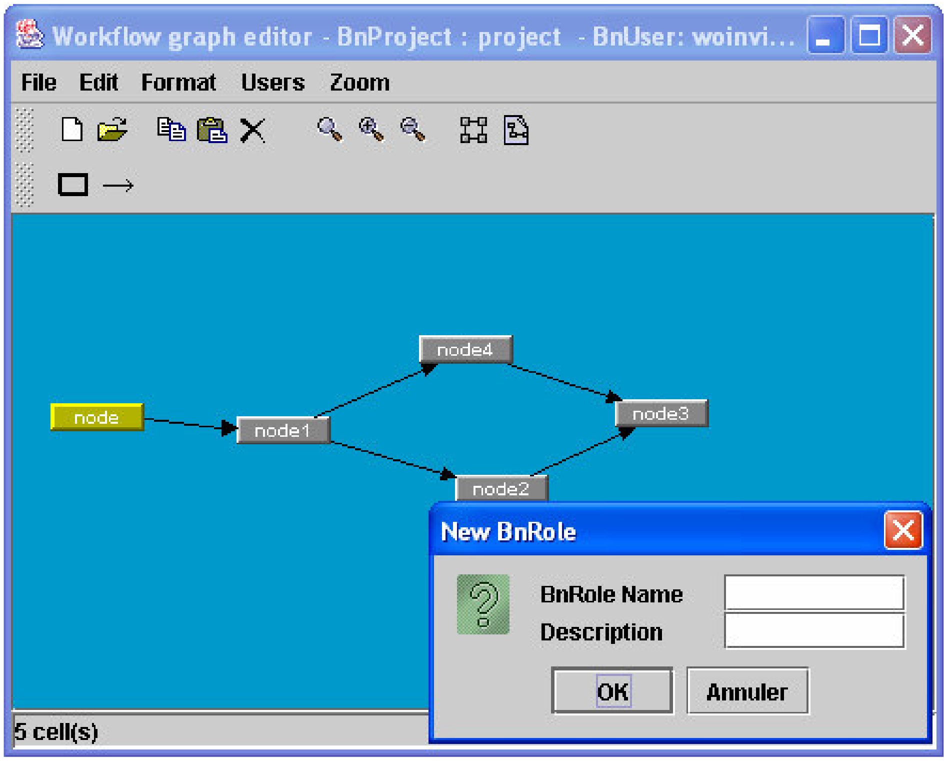
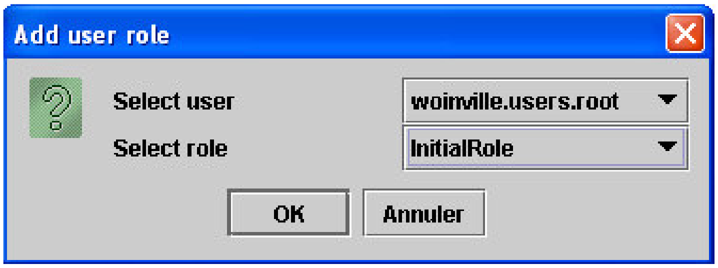


Figure 16: GraphEditor when the user creates a new role into the project.

Add User Role

In order to set a new role to a user in a specific Bonita project, the user must select the option "Add Role" by clicking on the right mouse button over the GraphEditor panel.



Select user: woinville.users.root

Select role: InitialRole

Figure 17: GraphEditor when the user assign a role to another user.