BONITA Workflow Cooperative System
User's Guide

(Version 1.2)

Miguel Valdés Faura
BULL R&D

Index

Foreword

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.

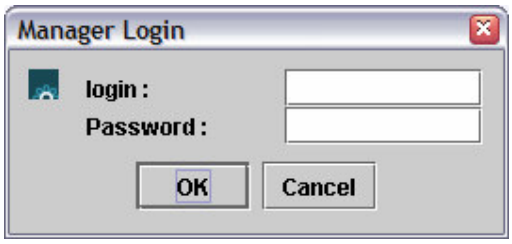
These applications compose the Bonita Workflow Management System, known as “Manager” and formed by the Bonita Workflow Definition Component (“GraphEditor” application) and the Bonita Workflow Execution Component (Worklist application).
The Bonita Workflow Management System is designed in order to facilitate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

In the next sections we explain the basic functionalities of Bonita Workflow Management applications and we illustrate it with some workflow examples.

* Note: In this document terms like project/process are used to identify a workflow process.
In the same context node/activity identifies a workflow task/step/activity.

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:



Worklist Introduction

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

Worklist Example

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

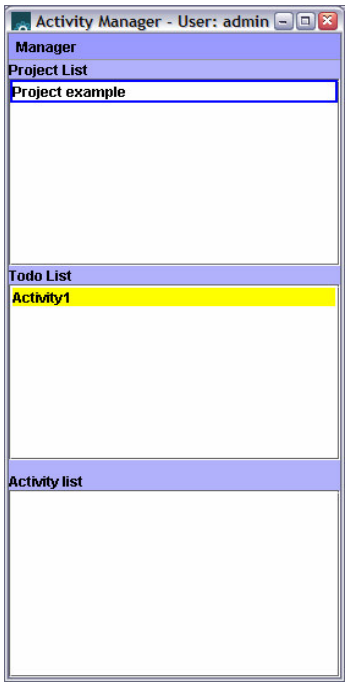


Figure 1: Activities of the project

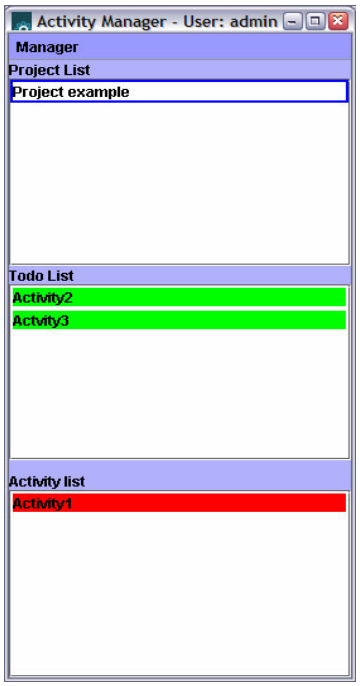


Figure 2: Execution of activity 1

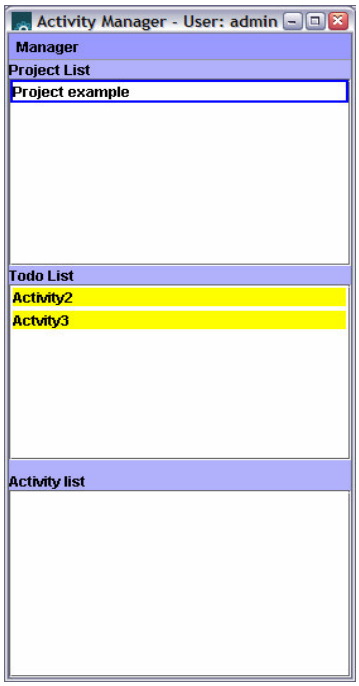


Figure 3: Activity 1 terminated

The following example shows an execution environment with three users (*admin*, *miguel* 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.

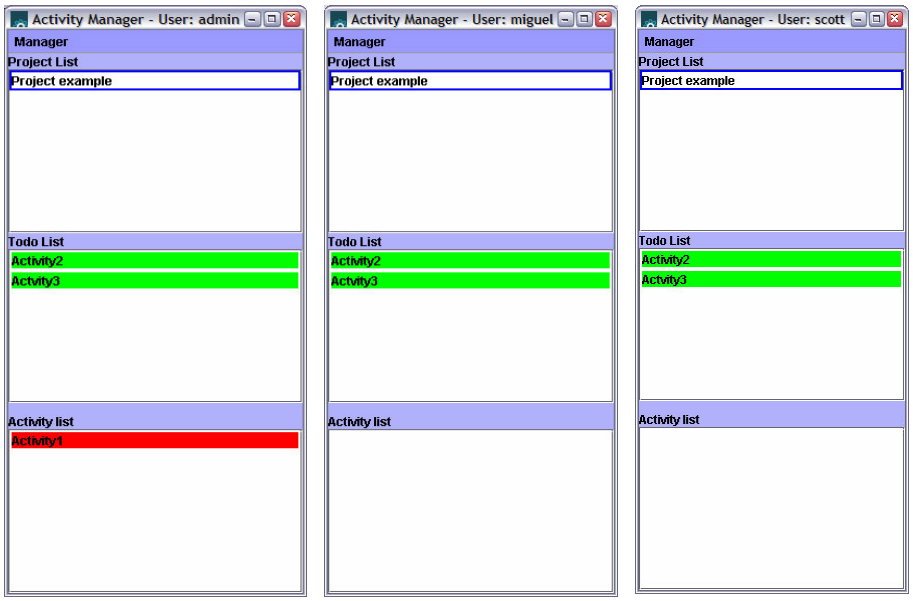


Figure 4: **User *Worklists* when the user *admin* executes the first activity**

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

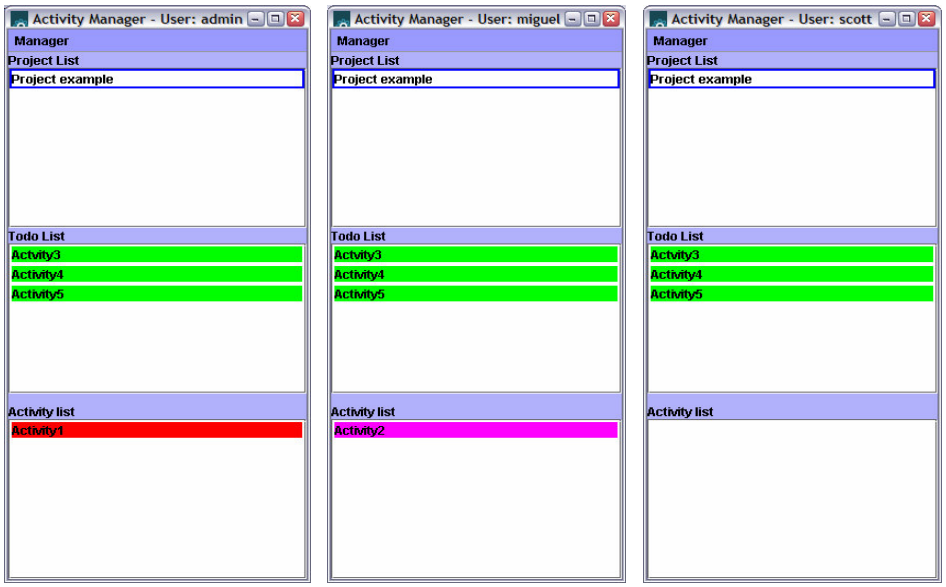


Figure 5: User Worklists when the user *test* anticipates the second activity

Create and Clone Projects

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 copies the hooks and roles of the previous one.

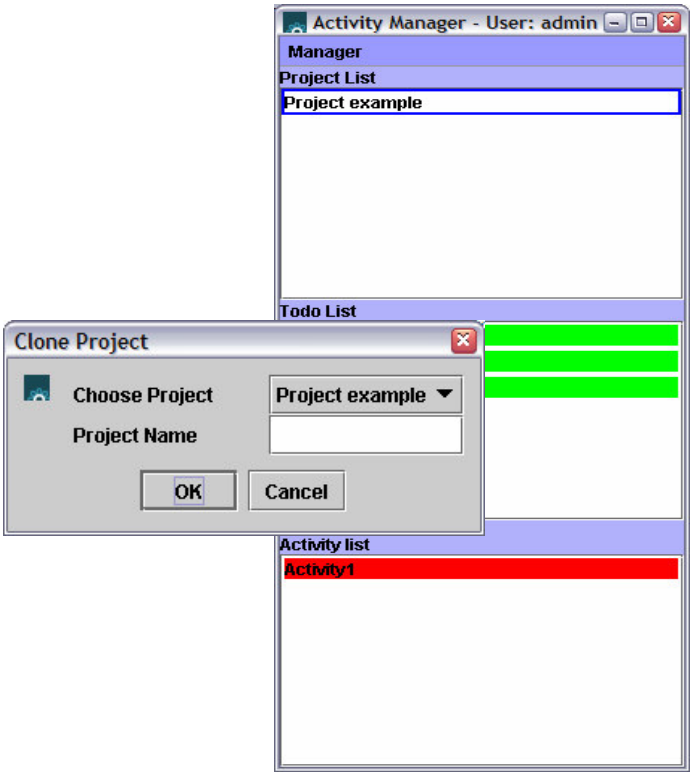


Figure 6: **User Worklists when the user clones an existing project**

Project Instances

Cooperatives workflows are mostly used for long running processes in which participants work together in order to accomplish a common objective. This kind of workflows does not generally use workflow instances. Users are working in the same workflow project instead of one project instance by user.

Otherwise, if you want to use Bonita in a classical workflow administration mode, you can use Bonita process instantiation functionality. In some applications, we need to define a workflow project model that will be performed by different users in different contexts. For example if we apply a workflow system in online shop application, the order sales process will be instantiated by each customer of the system. Each project instance is assigned to a specific user and will be managed by the customer services department.

In Bonita, Worklist application allows the users to instantiate an existing workflow project. This operation creates a new project instance identified with an instance number: if the project name is "Sale order" each project instance name follows the next syntax: "Sales order_instance*#instancesNumber*".

If the user wants to instantiate a project, he must select the option Instantiate Project by clicking on the right mouse button over the Project List. You can also use the Manager Menu to do this operation.

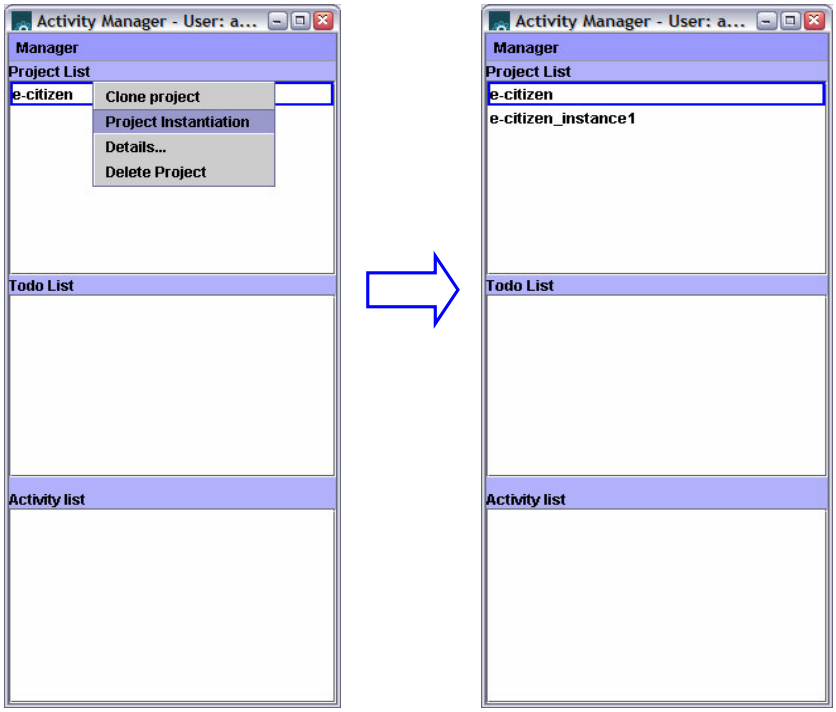


Figure 7: User Worklists when the user instantiates a project

Delete Project

If a user wants to remove an existing workflow project or an existing workflow instance, the Worklist application offers “Delete Project” operation. Only the user has created the project can removed it. For project instances this user is the same user that creates the instance and for projects, the user is the principal administrator of the project.

In some cases, for example when we try to delete a workflow project that has sub-processes inside, the system does not accept project deletion because the system could be in an inconsistent state. In these cases we have to delete different projects and sub-process in a specific order.
The next table shows possible cases:

Delete operation	Correct issue
The user tries to delete a project which contains one or more sub-processes.	We have to delete each sub-process activity from parent workflow project. This operation removes the sub-process activity and project associated.
The user tries to delete a project which is a sub-process.	We have to delete sub-process activity first.
The user tries to delete a project created by another user.	Contact the project administrator to remove the project.

If the user wants to delete a project, he must select the option Delete Project by clicking on the right mouse button over the project selected in Project List:

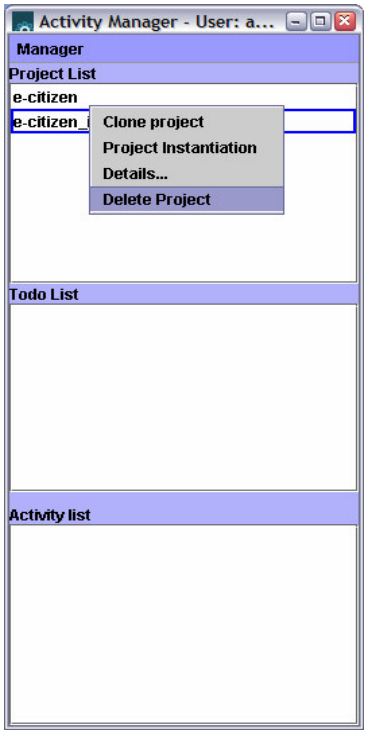
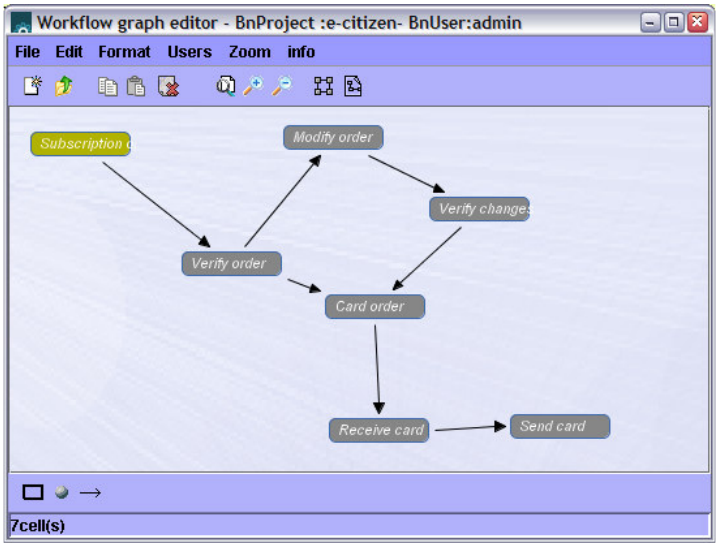


Figure 8: User Worklists when the user tries to delete a project

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.



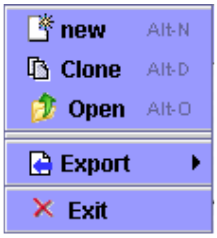
This is the first screen that appears when any user opens a project.

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

Figure 9: GraphEditor when the user opens an existing project.

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.
- **Circle Layout:** Project activities are placed as a circle.
- **Expand Layout:** Project activities are expanded.

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.

Info Menu

- **About Bonita:** Shows a brief description about Bonita definition and execution components.

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 execute 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 obtain 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 JOIN activity routing nodes and then he edits edges to define the split condition.

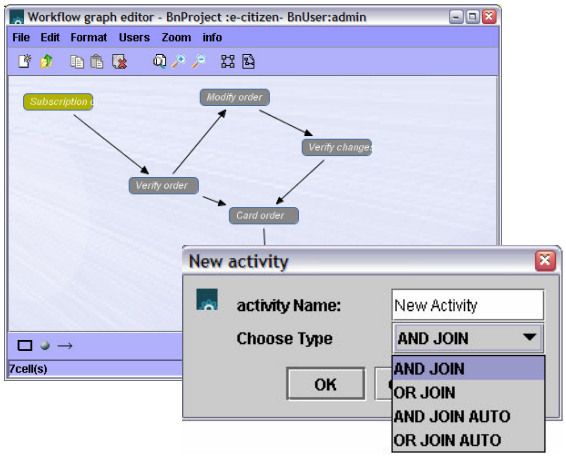


Figure 10: GraphEditor when the user adds a new activity.

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

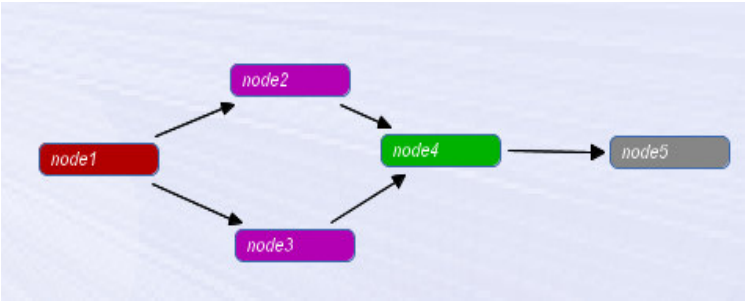


Figure 11: Workflow execution when the activity "node4" with AND JOIN routing mode is active (anticipable state) by two anticipating activities.

In this example all activities are anticipable and use AND JOIN routing mode. When the user starts the first activity ("node1"), activities "node2" and "node3" are in anticipable state. If the user starts theses activities the "node4" activity becomes anticipable. "node4" activity is also AND JOIN, so it will be active (ready or anticipable) only if previous connected activities are started or terminated.

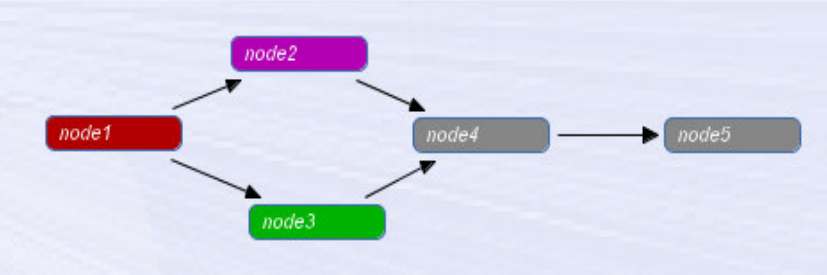


Figure 12: Workflow execution when the activity "node4" with AND JOIN routing mode is waiting (initial state) for the execution of activity "node3".

In this example all activities are anticipable and use AND JOIN routing mode. When the user starts the first activity ("node1"), activities "node2" and "node3" are in anticipable state. If the user starts "node2" activity, this one is anticipating and "node4" activity is waiting for the start of "node3".

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

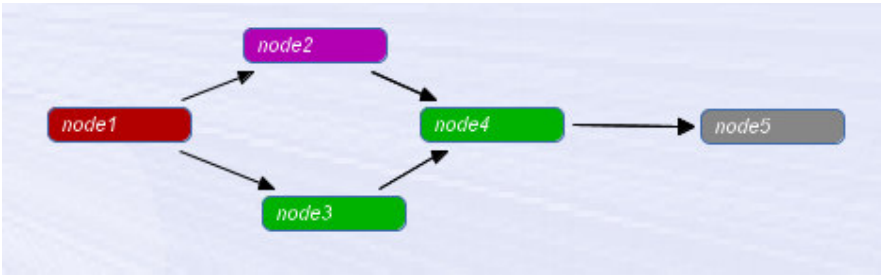


Figure 13: Workflow execution when the activity "node4" with OR JOIN routing mode is active (anticipable state) by the activity "node2".

In the example "node4" activity has OR JOIN routing type, so when the user starts "node2", the first one becomes active (anticipable state in this case). So, an OR JOIN activity becomes active (ready or anticipable state) when one of the previous connected activities are started or terminated.

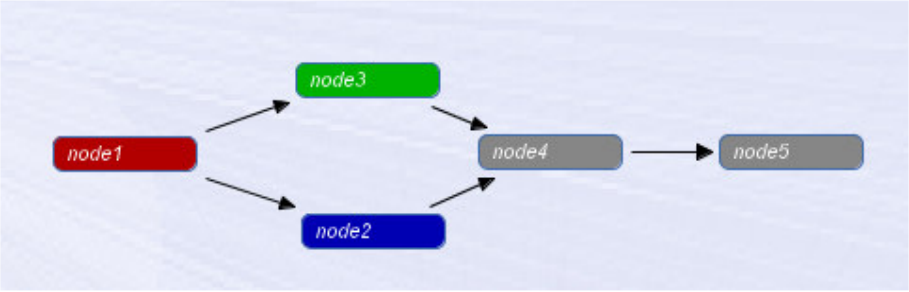


Figure 14: Workflow execution when the activity "node4" with OR JOIN routing mode is waiting (ready state) for the user execution of activity "node3".

In the example, the execution of "node2" was cancelled, so the "node4" activity is waiting for the execution of "node3" activity in order to become active (ready or anticipable state).

Sub-Process Activity

Bonita system allows the creation of a special activity containing a workflow process. This activity is called Sub-Process activity. We can use this kind of activity in order to specify a jerarquical structure of workflow processes. This feature is possible by assigning an activity to an existing process, so in order to create a new sub-process activity the system makes a clone of the process attached to it.

After activity sub-process creation, a new dependence relationship between sub-process and the workflow project has been created in terms of parent-child relationship.
The life cycle of this sub-process is controlled by the parent. When the user starts sub-process activity from parent workflow project, the life cycle of sub-process is activated.
In order to finish/terminate the sub-process activity, the system verifies if sub-process activities are finish/terminate.

When we creates a new sub-process, the workflow relevant data of the parent project must be accessible by this one, so we can use all Bonita properties available in sub-process activity within the sub-process. Evidently, properties created by sub-process during its execution are mapped within the parent activity after sub-process termination.

The next figures shows, step by step, the correct sequence to add a sub-process activity within an existing workflow project:

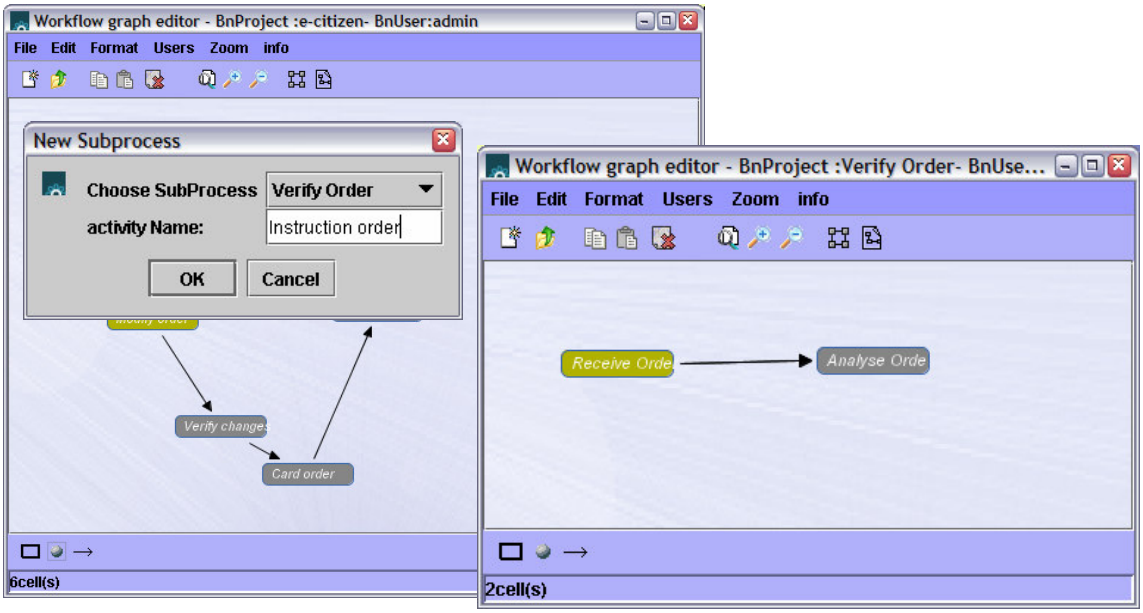


Figure 15: **GraphEditor when the user creates a new sub-process activity**

In the last figure the workflow administration of “e-citizen” workflow project wants to insert a sub-process activity called “Instruction order”. This activity will be attached to “Verify Order” workflow project which is composed by two activities: “Receive order” and “Analyze Order”.
After that “Instruction order” sub-process activity is created in “e-citizen” workflow project and a new project is created with the same name. This project is a sub-process of the previous one.

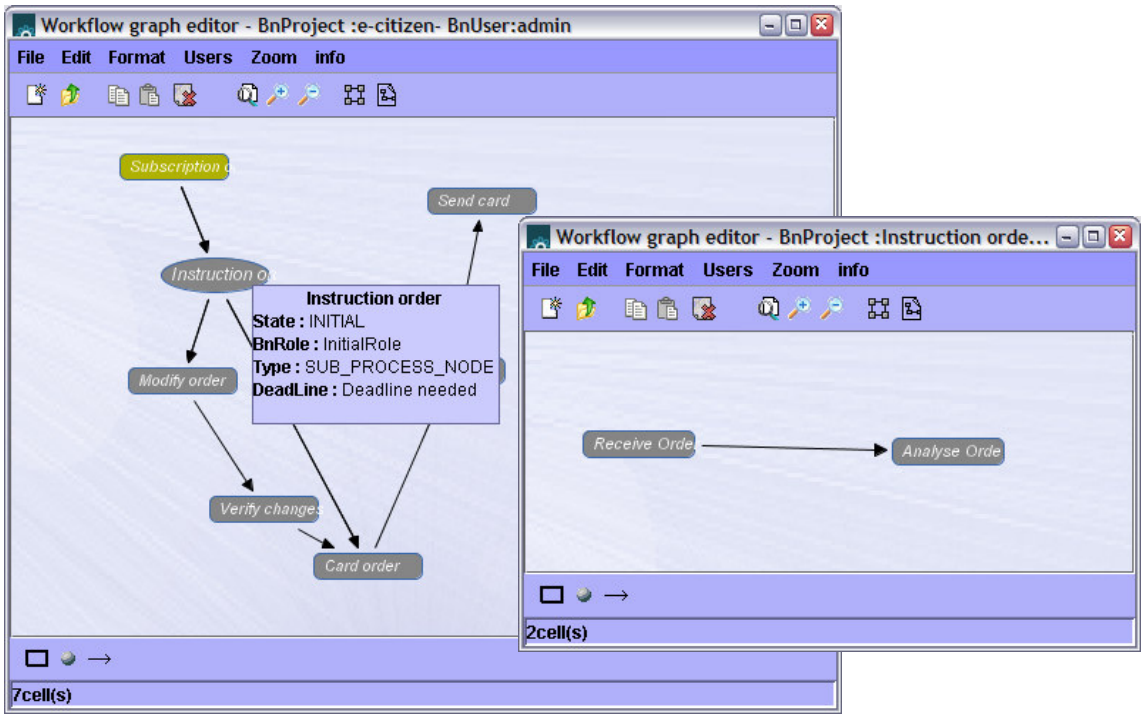
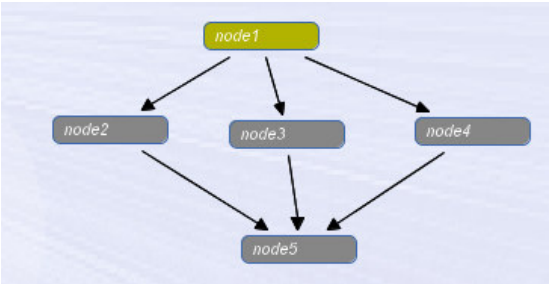


Figure 16: **GraphEditor after sub-process activity creation**

Activities Iteration

In this version, and as beta functionality for the moment, you can iterate between different activities in Bonita. For this purpose, and when the user decides to iterate one or more activities, the system is able to find the *iteration path* between two activities and then iterates between all activities contained within this path.



If you want to iterates between “node5” and “node1” activities, the system iterates also activities placed in the iteration path, so when the user terminates “node5” activity and iteration condition is true, “node1” activity is ready again and “node2” “node3” and “node4” activities are also iterated.

Figure 17: Activities “node2”, “node3” and “node4” takes part of iteration between “node5” to “node1”.

In the same context, activities that are not in iteration path and having edges with one of previous activities are standby until iteration is finished.

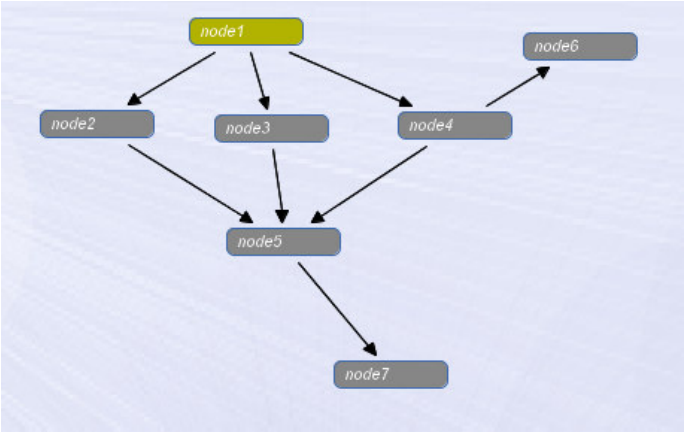


Figure 18: Activities "node6" and "node7" will be standby until iteration is finished.

With GraphEditor application you must insert a new iteration between two nodes by selecting the option "Add Iteration" and clicking on the right mouse button over the last activity you want iterate. In previous example you have to add a new iteration from "node5" activity to "node1". After this operation next icons are showed in your graph:

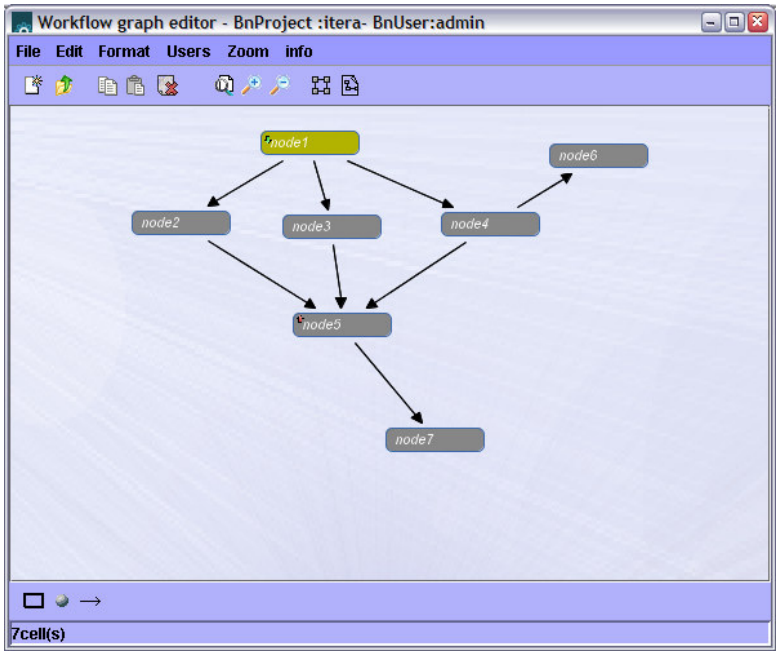


Figure 19: GraphEditor after insertion of iteration from "node5" activity to "node1".

Activity transitions

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.

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 use the activities properties values to select the progress of the workflow process execution. In the example, when the activity "Verify order" is finished, the execution engines evaluates out edges conditions and if the value of the property "correct" is equal to "ok" the system actives the activity "Card order".

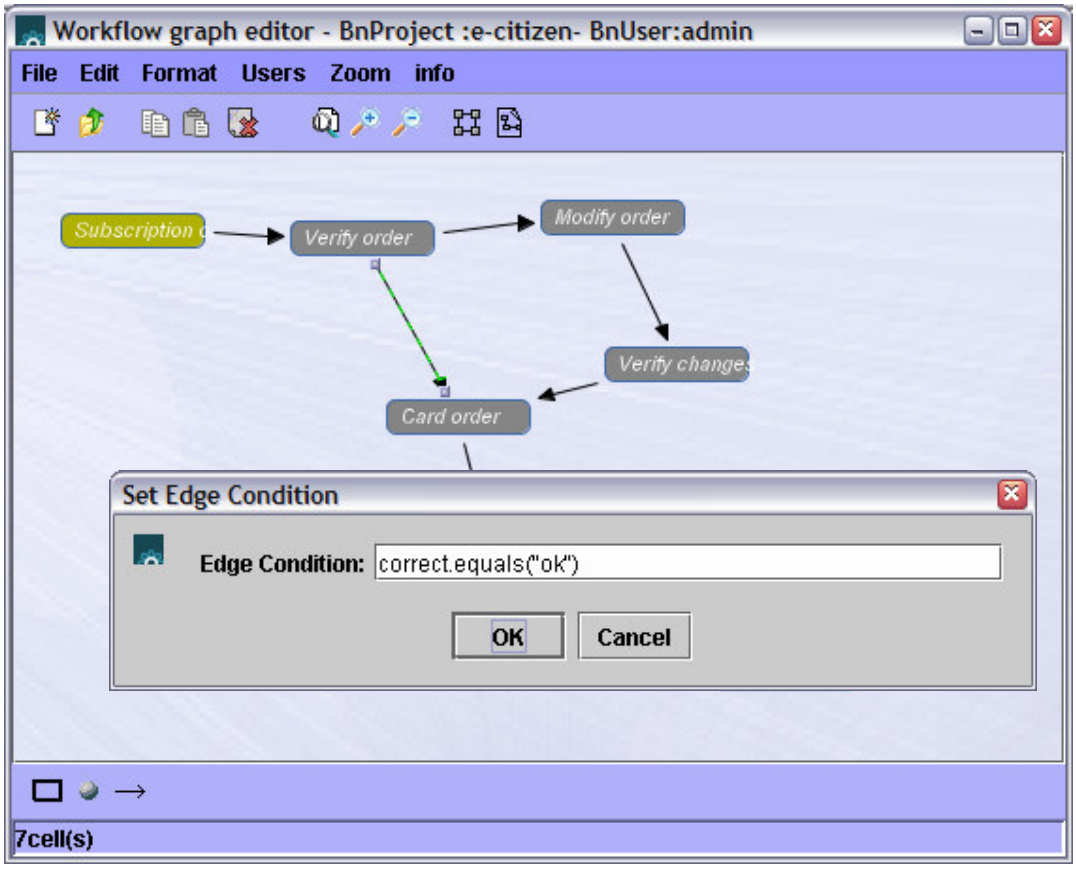


Figure 20: GraphEditor when the user sets an edge condition.

In the previous example we can consider that edge transition between activities "Verify order" and "Modify order" has the opposite condition than the last one, so *correct.equals("nok")*. In this case, if the "correct" property is equals to "ok", the branch starts by activity "Modify order" will be cancelled.

Activity Properties

Bonita Cooperative Workflow uses the concept of properties to control the process execution data. Each Bonita Workflow process can have different 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 propagate attribute. This Boolean allows the property propagation when the associated activity is executed.
The GraphEditor application allows also the addition of new properties to workflow projects by clicking on the right mouse button over the project definition surface. These properties are defined as activity properties but does not have the propagate attribute. Project properties can be used by all Bonita components takes part of the project: activities, edge conditions and hooks.

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.

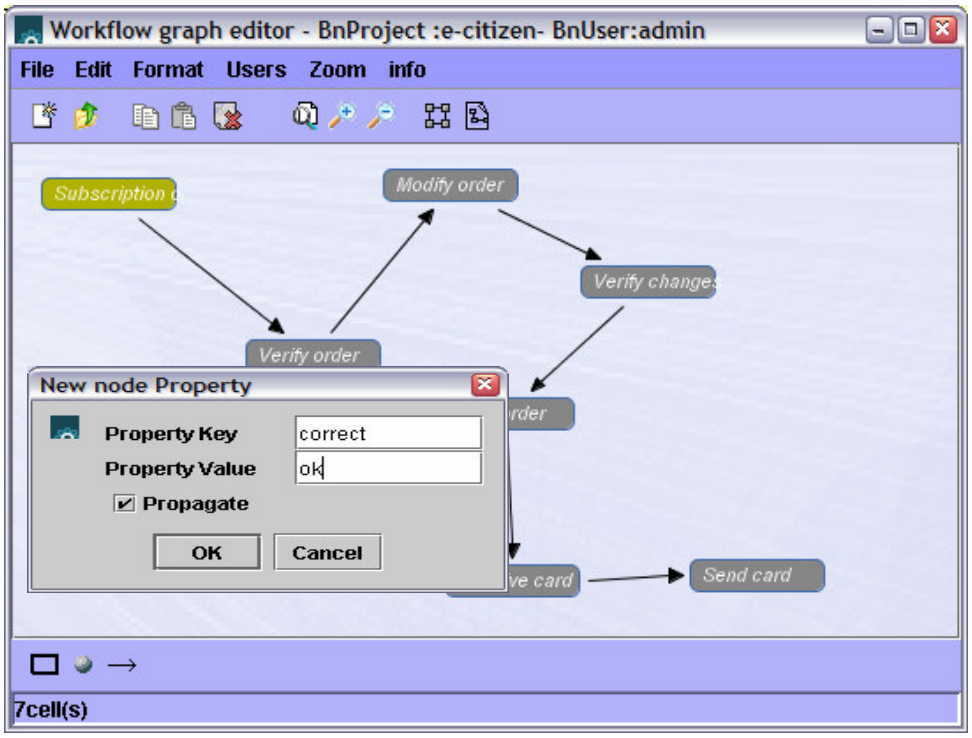


Figure 21: GraphEditor when the user set a new activity property.

We continue to talk about properties in section "Activity Hook", and we will explain the steps to follow in order to create properties dynamically at run time.

Role Mappers

In the context of administrative workflow, we need some mechanisms to fill in automatically the roles of workflow projects when these projects are instantiated. Bonita is able to obtain the correct people/users corresponding to the each activity role declared during the project definition process. This operation is possible by means of what we have called mappers, Bonita mappers can retrieve user-role information from different sources: LDAP directories, external database, web services API or whatever you want.

With the GraphEditor application, you can add mappers by using Bonita Users menu. The Add Mapper dialog box lets you to set following fields: project role, mapper name and mapper type. Please take a look at $BONITA_HOME\doc\mappers&performers.pdf document for more information about mappers configuration within Bonita.

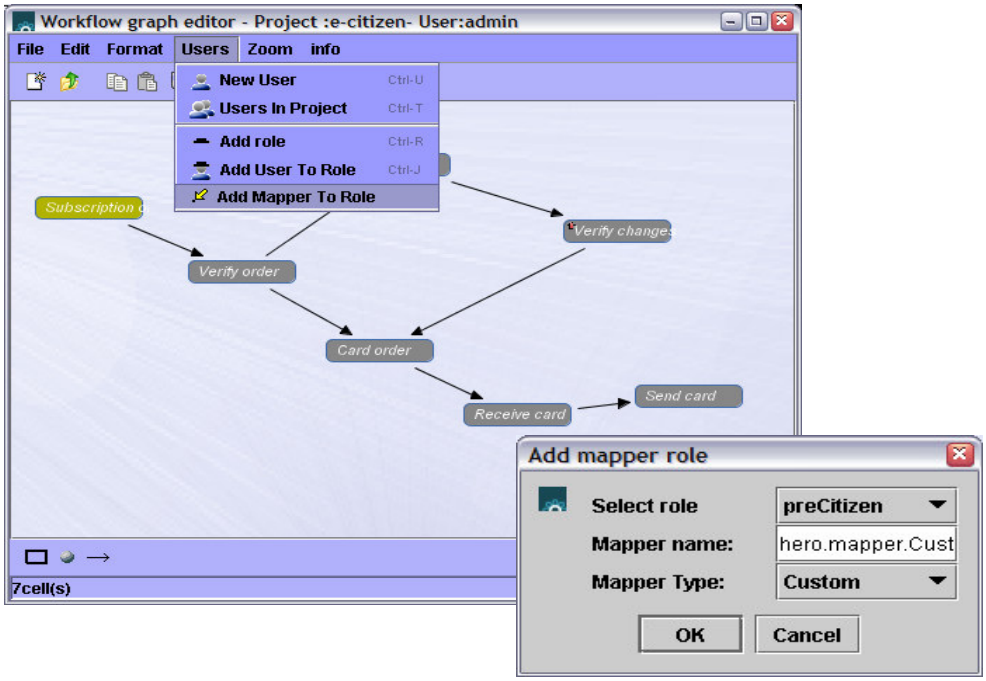


Figure 22: GraphEditor when the user adds new mapper.

Performer Assignment

New functionality oriented to administrative workflows and, in some cases, complementary to mappers described above.Bonita performer assignments are components able to select one of the possible users having the role assigned to an activity. Like this, the user who is responsible to execute the activity is resolved during the workflow instance execution.Instead of mapper services, Bonita performer assignments are loaded inside workflow instances.

Each workflow activity could have a performer assignment that will be executed during the activity activation. There are two types of Performer assignments available: callback and properties.

The Grapheditor application allows the performer assignment addition using the context menu of workflow activities:

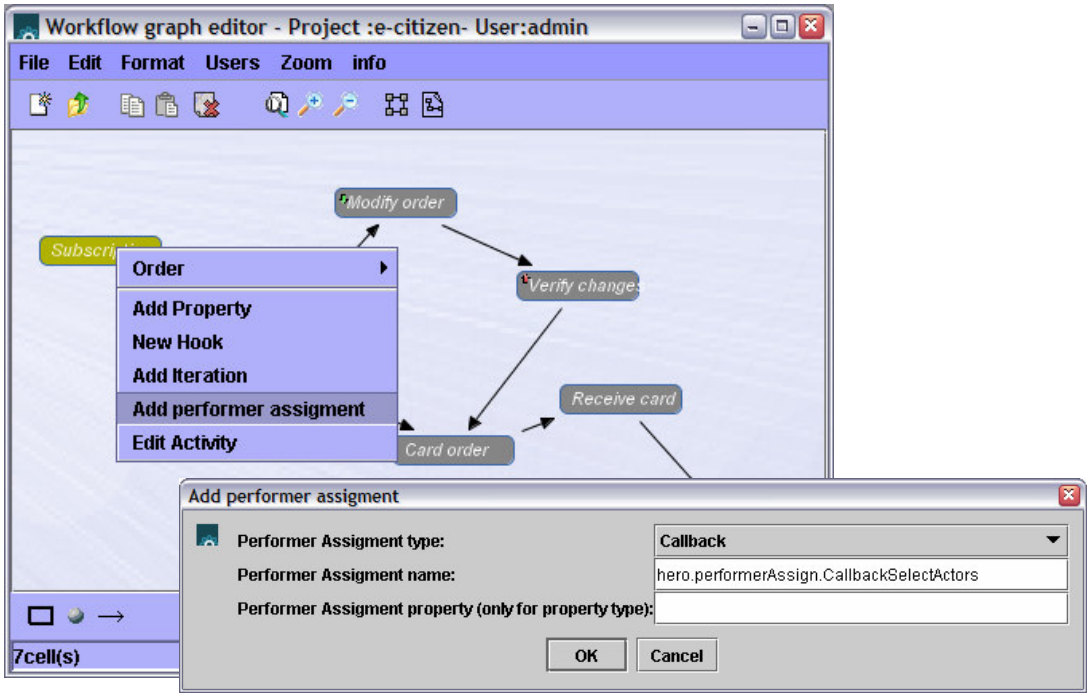


Figure 23: GraphEditor when the user adds new performer assignment

Please take a look at $BONITA_HOME\doc\mappers&performers.pdf document for more information about performer assignments configuration within Bonita.

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 authorized to execute the activity and the activity description text.

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.

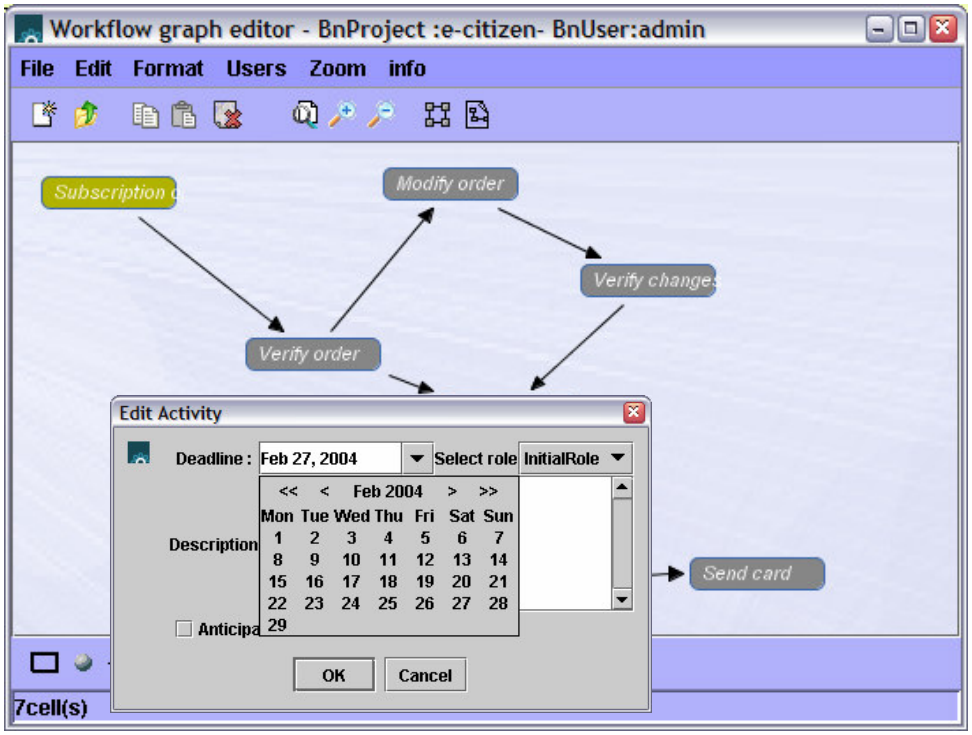


Figure 24: **GraphEditor when the user edits the activity parameter**s.

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, for example, an email or a jabber instant message.

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

In this dialog box we can also change the activity execution model. By default, all created activities are in anticipable mode, but if we want to execute one or more activities in a traditional mode; the anticipable checkbox must be disabled.

Activity Hooks

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

In Bonita, hooks are divided in two types: Hooks and InterHooks. Each type can be assigned to workflow projects and workflow activities. When the user sets a new hook for a workflow project, this one is assigned to all activities of this project.
The key difference between these two types is the persistence data. In InterHooks, logical syntax is stored in the database as a string field, so when the interpreter has to execute this type of hook it loads this data before hook execution. Otherwise, Hooks uses, for example, java classes in order to store the logical data, so the interpreter looks for this class in the classpath and executes the hook.

With the GraphEditor application, the user can set and define activity actions by means of the BeanShell interpret hooks (InterHooks).
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.

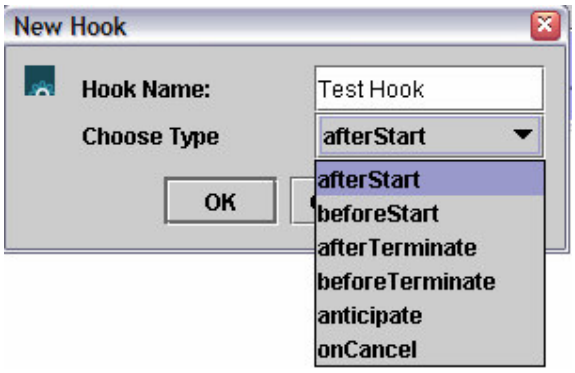


Figure 25: **GraphEditor when the user adds new hook.**

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 in the system. Now, its time to take a look at hooks functionalities. The user can use all Bonita API in order to define hooks actions, so all the Bonita EJB's are accessible from the hook interface.

If you want to use external functionalities in your hook actions, like external web services or other applications, you can access to all classes available in your JOnAS 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. Within the dialog hook window we can identify the default imports and the action type function associated to this hook. Within this function the user can write

the actions of this hook by using Java syntax. These actions will be executed at run time by the execution engine in a specific moment of the activity execution.

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

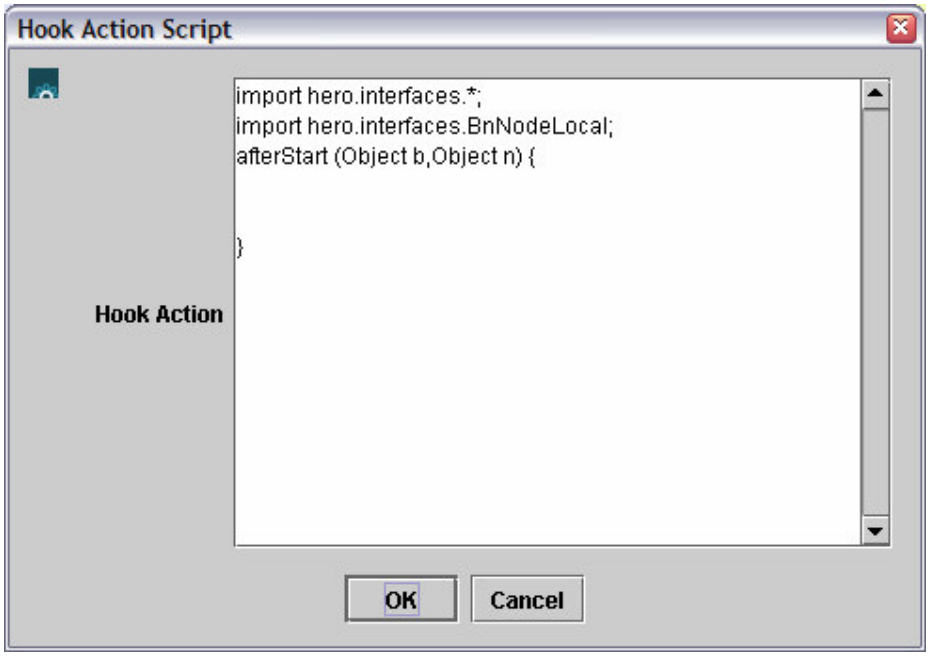


Figure 26: GraphEditor when the user defines the hook actions.

Sample Application

In order to explain the hook functionalities we can take an example of workflow application that uses 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 profiles of the users which access to the system.

The Bonita Workflow representation of this CRM system is 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.

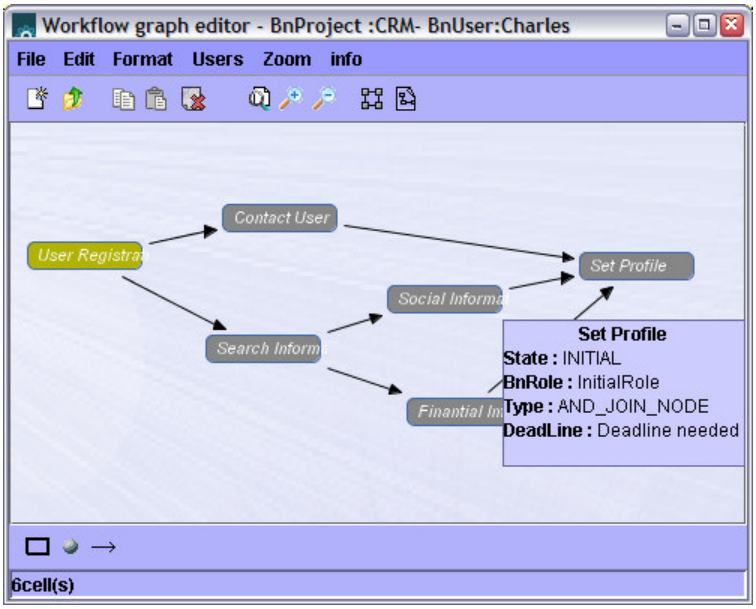


Figure 27: **Workflow Representation of the CRM System**.

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:

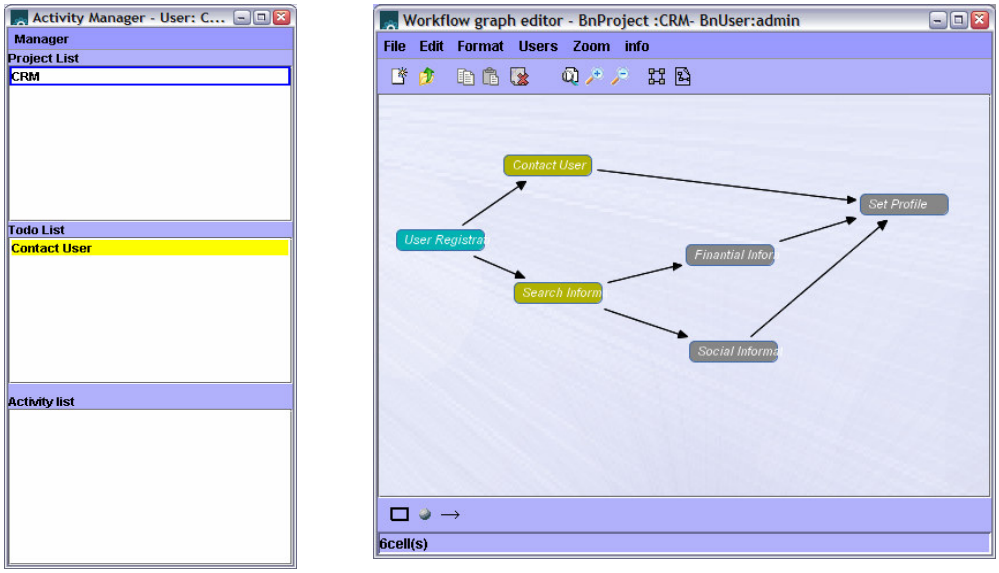


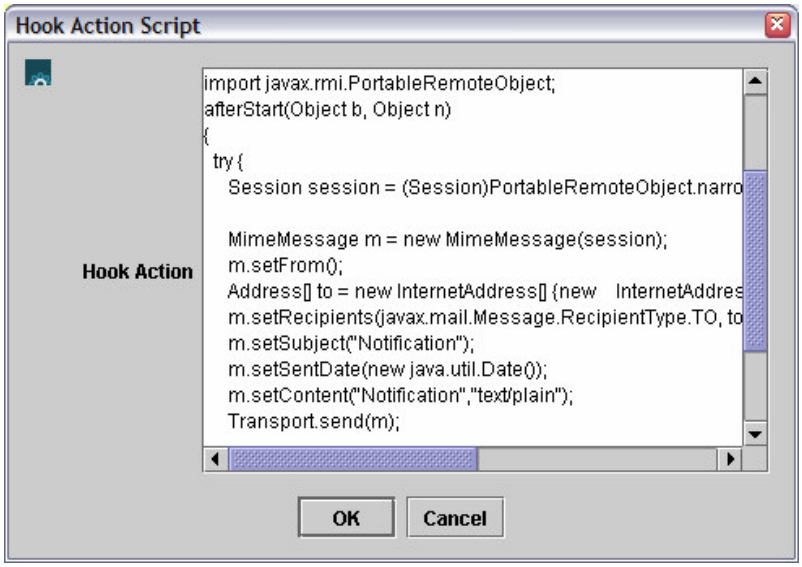
Figure 28: **Worklist and GraphEditor application for the CRM System example. The "Contact User" activity is associated to the employee *Charles* in the todo list.**

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:


```
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);
  }
}
```



* 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 29: **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 fit his needs, we can filtrate the basic user information before sending the email.

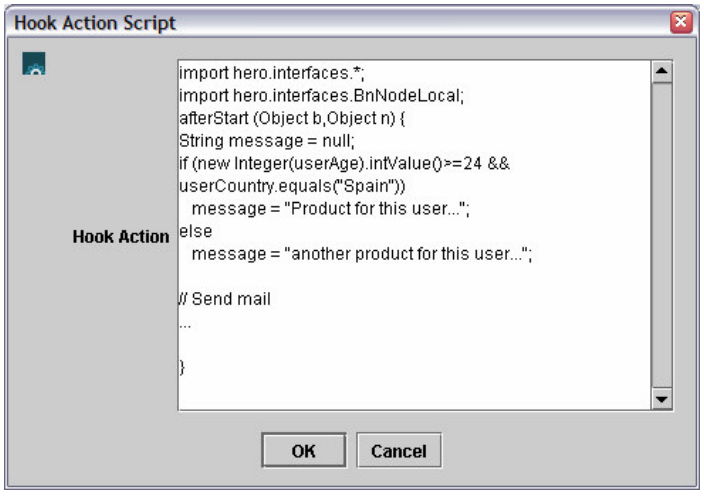


Figure 30: **After start hook assigned to "Contact User" activity with filtration of user information.**

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 controls the user information search, so if he decides to start this, connected activities become anticipable: "Financial information" and "Social Information".

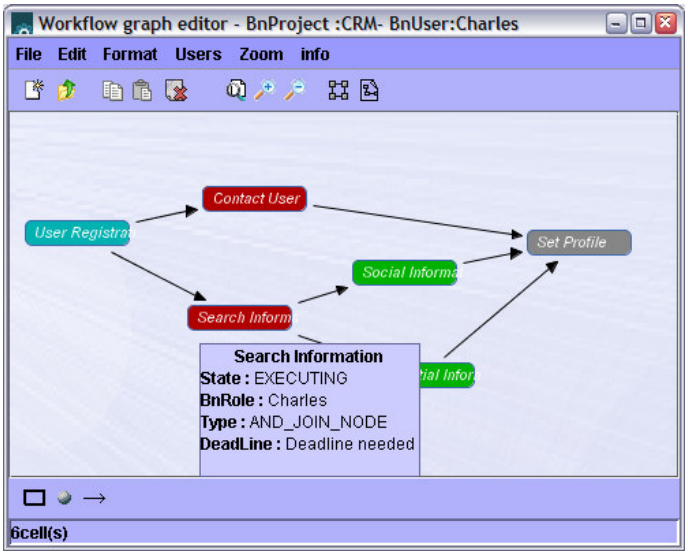


Figure 31: CRM workflow system when the user *Guillaume* executes "Search Information" activity.

Now, employees responsible 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.

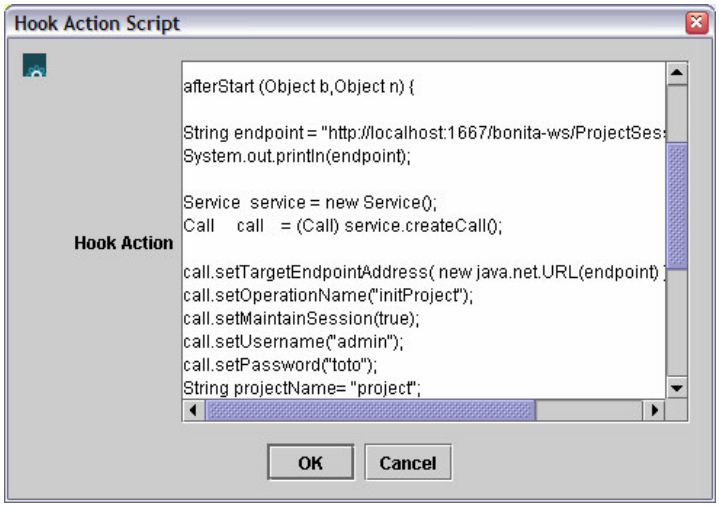


Figure 32: After **start hook assigned to "Financial Information" activity that uses web services calls**

After that, the user responsible of the last activity “Set User Profile” inserts previous user information into a database to obtain the 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.util.Date	**getCreationDate**()
hero.interfaces.BnUserLocal	**getCreator**() Retrieve the BnNode's Creator.
java.util.Date	**getDeadline**() Retrieve the BnNode's Deadline.
java.lang.String	**getDescription**() Retrieve the BnNode's Description.
java.util.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.util.Date	**getModificationDate**()

java.lang.String	**getName**() Retrieve the BnNode's Name.
java.util.Collection	**getOutBnEdges**()
java.util.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	**setBnNodeLightValue**(hero.interfaces.BnNodeLightValue v)
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.util.Date pDate)
void	**setCreator**(hero.interfaces.BnUserLocal pCreator) Set the BnNode's Creator.
void	**setDeadline**(java.util.Date pDeadline) Set the BnNode's Deadline.
void	**setDescription**(java.lang.String pDescription) Set the BnNode's Description.
void	**setEndDate**(java.util.Date pEndDate) Set the BnNode's EndDate.
void	**setExecutor**(hero.interfaces.BnUserLocal pExecutor) Set the BnNode's Creator.
void	**setModificationDate**(java.util.Date pDate)
void	**setStartDate**(java.util.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.

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

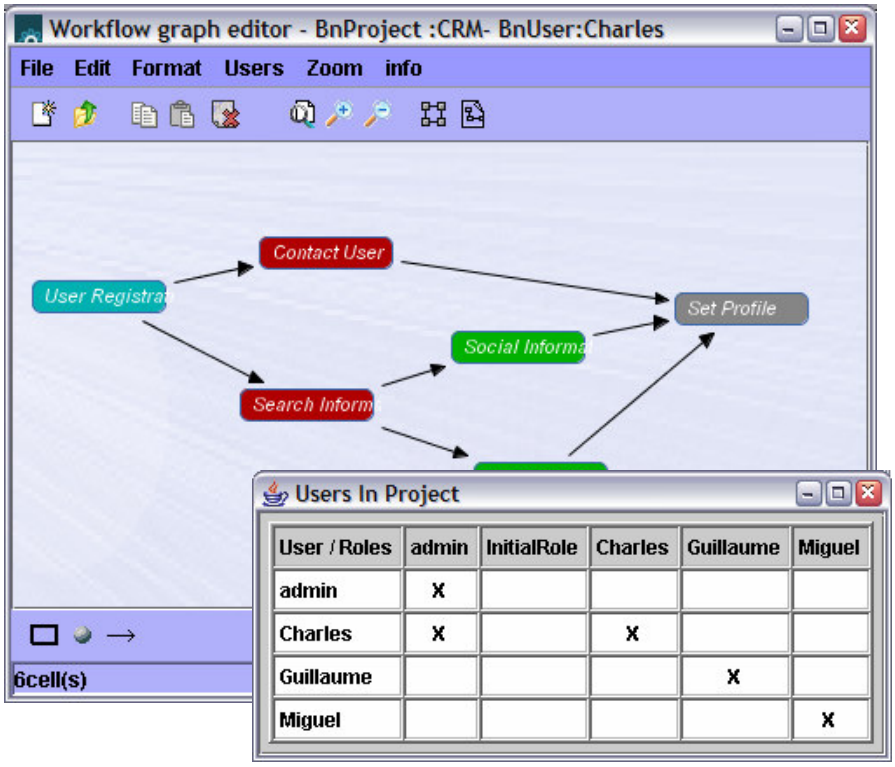


Figure 33: **GraphEditor when the user clicks to the Users in Project option**.

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.

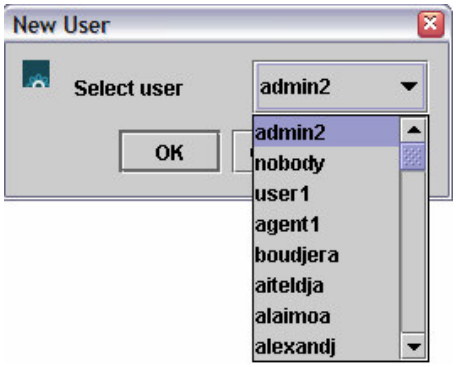


Figure 34: **GraphEditor when the user clicks to the Add User option.**

Add Project Role

A role 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.

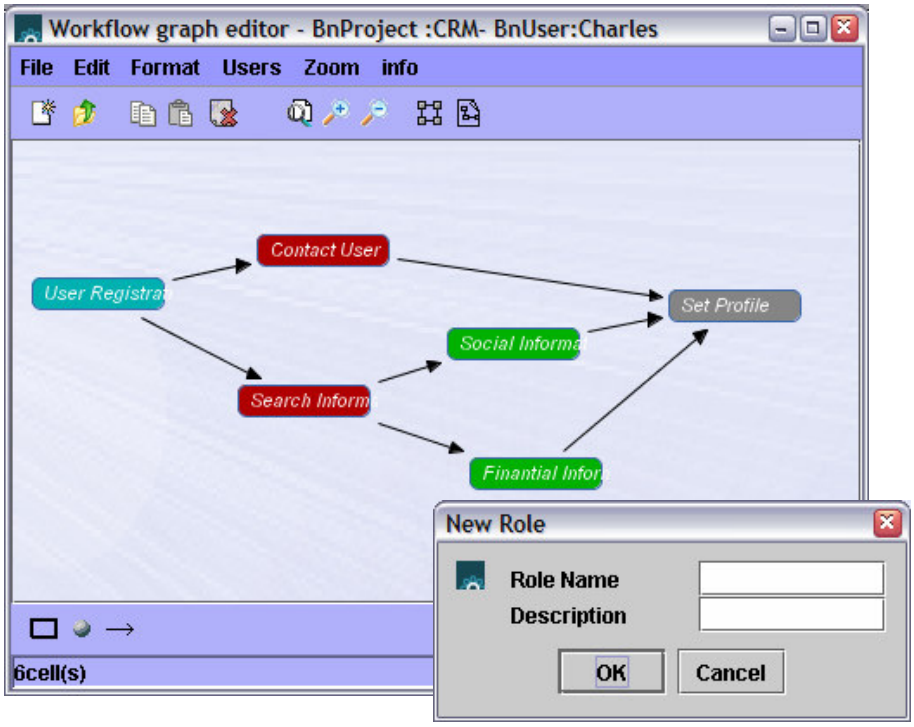


Figure 35: **GraphEditor when the user creates a new role into the project**.

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.

Figure 36: GraphEditor when the user assigns a role to another user.

Chapter 3. Business Process Example

Introduction

This example illustrates different features of Bonita Workflow System like: projects, subprojects, project instances, users, roles, hooks, properties and conditions. This example does not represent a real world workflow example or the better solution to resolve a specific workflow problem. In this example we will use the Manager application (Worklist + GraphEditor) to perform the workflow definition project and to simulate/execute a workflow execution example.

This workflow example represents an order entry system in which a customer interacts with the customer service to achieve his order.
This business example is composed by two projects (Bonita workflow process): *Order Processing* project and *Customer Service* project.

Order Processing project is the main project. This project takes three strings as input data corresponding to the customer name, product name and number of items for this product. After processing this order the system indicates if the order was accepted or rejected.
This project contains the following steps:

- The customer sends his order to the system.
- The customer data is checked in order to verify the stock availability for this product.
- If the stock status is correct, the previous customer data is sent to a subProcess that confirms the sale.
- When this subProcess is finished the system starts or rejects the sale order.

The *Customer Service* project is a subproject of the previous one. This project asks to the customer for the order confirmation and then, the customer service employee in charge of accepts the order notifies the user choice.
The project steps are the following:

- The system shows the order data to the customer and waits for the customer confirmation or rejection.
- After user confirmation/reject the customer service authorizes this sale or cancels it.

Do it with Bonita

Let's go to define the previous example using Bonita Manager application. This graphical application is very useful in some kinds of workflow projects and it can be extended by accessing directly to different Bonita's API.

Once you correctly logged on Bonita Manager application, the Worklist tool should appeared in your screen. Now, we start the Business Process definition example by creating *Customer Service* sub-process, so you can use the Manager Menu for create a new Bonita project called *Customer Service.*
This sub-process is composed by two activities; the first one will be performed by the customer and the second one by one employee of customer service. These activities are not anticipable and uses AND JOIN default mode.

Customer Service project

- Add “Ask Customer” activity: Activity which asks to the user if he wants to confirm the order. For example we can imagine that this activity will show the customer order data: customer name, product name, number of product.

- Add “Notify Sales” activity: Customer service notifies if the user accepts the order.

After activities connection (from “Ask Customer” to “Notify Sales”), the sub-process view is the following:

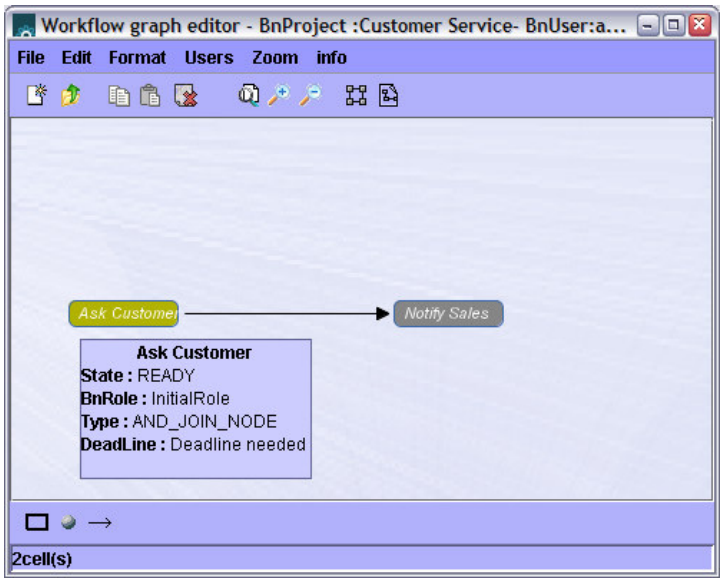


Figure 37: **GraphEditor for sub-process “Customer Service” after activities creation.**

In order to unset the default anticipable attribute for these activities you have to click on your right mouse button over each activity, select “Edit Activity” menu and disable anticipable value.

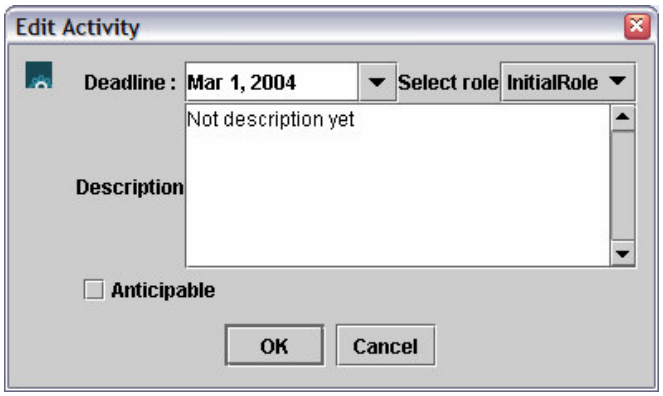


Figure 38: “**Edit Activity” dialog box for a specific activity.**

When this operation is finished, you have to define the roles that are going to be assigned to previous activities. In this example, we can imagine we have two roles: *agent* and *customer*. The *agent* role is attached to the “Notify Sales” activity and *customer* role is attached to the first one.

The next figure shows step by step the process to create these roles:

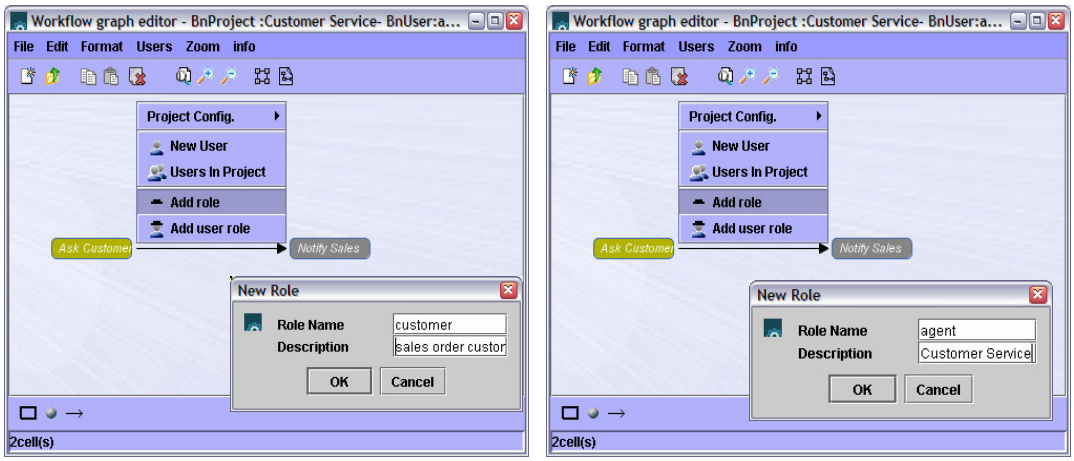


Figure 39: ***GraphEditor*** **when the user adds *customer* and *agent* roles to the project.**

Now you can change the default role assigned to each activity at creation time (*InitialRole*), by new roles available for this project. To perform this operation you have to use "Edit Activity" functionality:

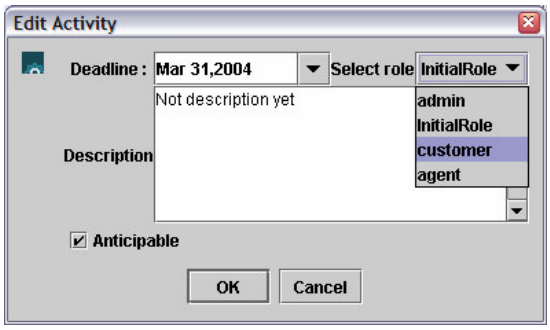


Figure 40: **"Edit Activity" dialog box for "Ask Customer" activity.**

To familiarise you with Bonita InterHooks we are going to insert two hooks in this sub-process. The first one is a basic InterHook that shows you in the command line[(1)] the customer data for a specific order. In Bonita, properties (workflow relevant data) are automatically mapped to be access from a hook or from an edge condition, so, in our example, when the user sends a new order from "Order Processing" project; he introduce his name, the name of the product and the number of items for this product he wants to buy. As we will see later, this data will be stored as Bonita properties.

When this data arrives to the "Customer Service" sub-process, you can use directly each property as a Java attribute within your Hooks.

[(1)] In a real world application, we can use other mechanisms to present order data to the customer, for example if we use Bonita within a web application.

The next figure shows you how to insert a new hook for the “Ask Customer” activity:

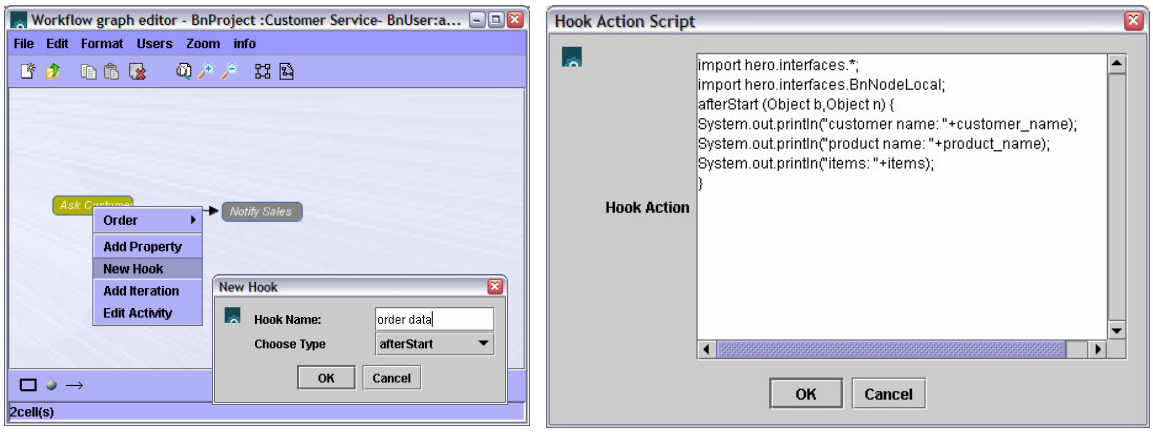


Figure 41: **New *AfterStart* hook for “Ask Customer” activity**

Hook Source code:

```
import hero.interfaces.*;
import hero.interfaces.BnNodeLocal;
afterStart (Object b,Object n) {
  System.out.println("customer name: "+customer_name);
  System.out.println("product name: "+product_name);
  System.out.println("items: "+items);
}
```

The second hook is assigned to “Notify Sales” activity and it could be in charge of creating a new property containing the user order confirmation choice.
If the user accepts the order this activity (executed by the customer service) sets a new property “partial_sales_status” to “ok”. Otherwise, if customer cancel previous activity we this property is set to “nok”.

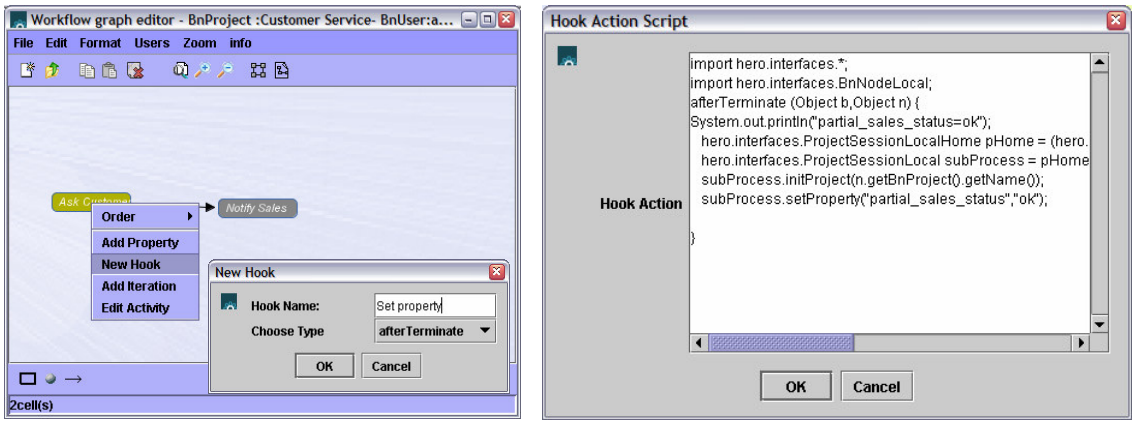


Figure 42: **New *AfterTerminate* hook for “Notify Sales” activity**

Hook source code for “Notify Sales” activity is:

```
import hero.interfaces.*;
import hero.interfaces.BnNodeLocal;
afterTerminate (Object b,Object n) {
System.out.println("partial_sales_status=ok");
  hero.interfaces.ProjectSessionLocalHome pHome =
(hero.interfaces.ProjectSessionLocalHome)hero.interfaces.ProjectSessionUtil.getLocalHome();
  hero.interfaces.ProjectSessionLocal subProcess = pHome.create();
  subProcess.initProject(n.getBnProject().getName());
  subProcess.setProperty("partial_sales_status","ok");

}
```

It’s time to create the principal project of this Business Process example. This project is the ‘Order Processing’ workflow project and it is composed by four traditional activities and one sub-process activity that has been created previously. In this project, all activities will be executed in a traditional mode.

Creates Order Processing project

- Add “Receive Order” activity: This activity is an AND JOIN activity in traditional mode, so you have to disable the anticipable attribute. In this activity the costumer enters his name, the product name and desired number of product items as activity Bonita properties as follows:

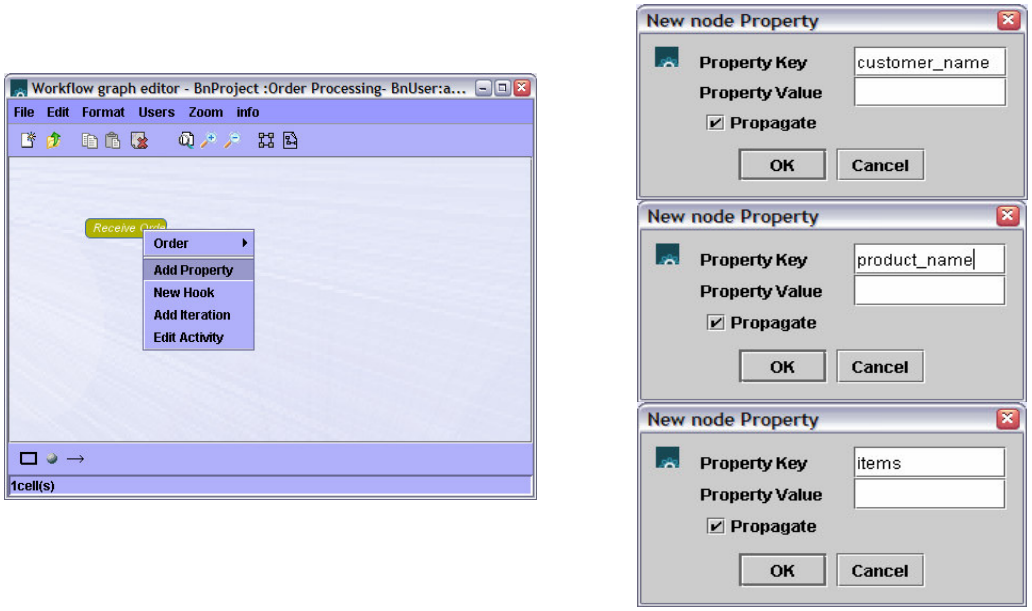


Figure 43: **GraphEditor when user adds “Receive Order” properties.**

Each time, a new customer instantiates the project the values of theses properties are actualized.

- Add “Check Stock” activity: This activity is an AutomaticAndJoin activity which verifies if the customer order must be satisfy. In order to continue the execution in a traditional mode, you have to disable anticipable attribute. After activity creation, you have to define a new edge (transition) between “Receive Order” activity and this one.

When this activity is activated, it is automatically executed and finished by the engine. We can define a simple Hook in order to emulate the check_stock process.
This hook set a new property called “stock_status” with the value “ok” or “nok”, depending of the product availability. In this example the hook only verifies if the user has sent a correct order[(2)].
Source code for this hook:

```
import hero.interfaces.*;
import hero.interfaces.BnNodeLocal;
beforeTerminate (Object b,Object n) {
hero.interfaces.ProjectSessionLocalHome pHome =
(hero.interfaces.ProjectSessionLocalHome)hero.interfaces.ProjectSessionUtil.getLocalHome();
 hero.interfaces.ProjectSessionLocal subProcess = pHome.create();
 subProcess.initProject(n.getBnProject().getName());
if (customer_name!=null && product_name!=null && items !=null)
{
 System.out.println("Stock_Status=ok");
 subProcess.setNodeProperty(n.getName(),"stock_status","ok",true);
}
else
{
 System.out.println("Stock_Status=nok");
 subProcess.setNodeProperty(n.getName(),"stock_status","nok",true);
}
}
```

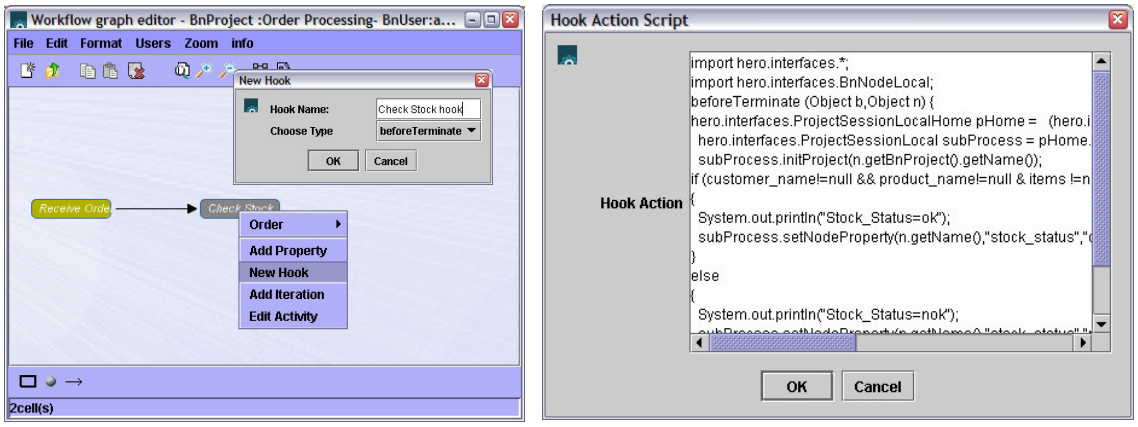


Figure 44: **GraphEditor when user adds beforeStart hook to “Check Stock” activity.**

- Add “Accept Order” subProcess activity: This activity is a subProcess activity attached to “Customer Service” project defined previously.
In order to create this sub-process activity, you have to click on the “New SubProcess” button:

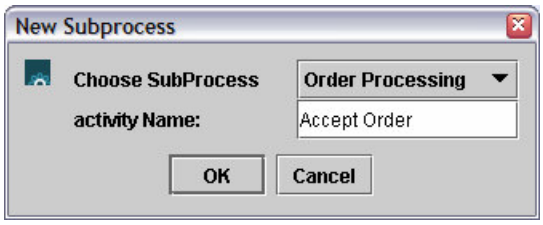


Figure 45: ***New SubProcess*** **dialog box for Accept Order activity.**

[(2)] In a real world application, this hook could access, for example, to an external web services in order to check product availability.

Once sub-process activity is created, the system clones the project "Customer Service" attached to this activity and sets all activities of sub-process to initial state. In execution time, and when sub-process activity is activated, the first sub-process activity will be ready.
The next figure shows you the current state of our Business Process example after the user adds a new edge between "Check Stock" and "Accept Order" activities:

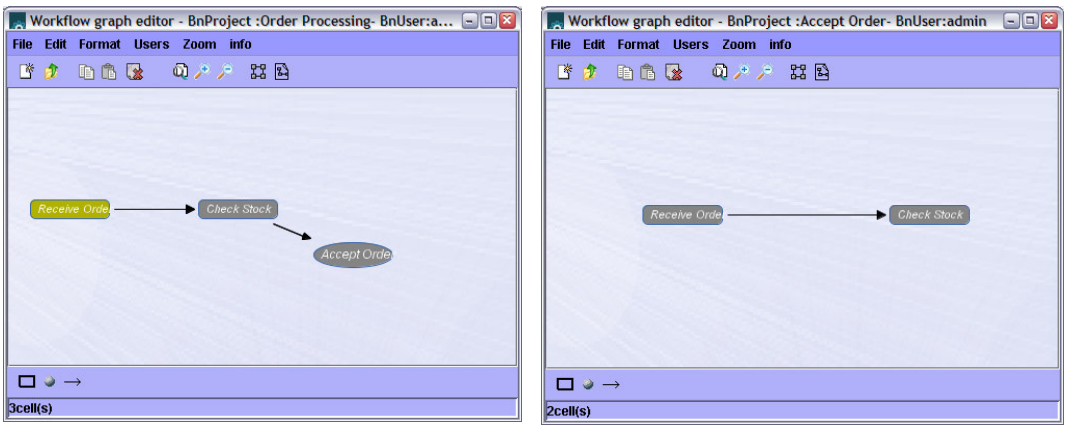


Figure 46: **On the left, "Order Processing" project with the new sub-process activity called "Accept Order". On the right, the "Accept Order" sub-process at initial state.**

The transition between "Check Stock" activity and "Accept Order" activity has a condition which verifies that stock_status property is "ok". In order to inserts this condition you have to double click on the edge and sets the following expression:

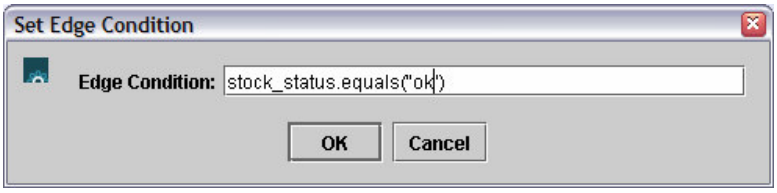


Figure 47: ***Set Edge Condition*** **dialog box for transition between "Check Stock" and "Accept Order" activities.**

The definition of this project ends by defining two activities, connected with "Accept Order" activity:

- Add "Ship & Report" activity: If the user confirms the order. AutomaticAndJoin activity. Not anticipable activity activated if the property partial_sales_status = ok.

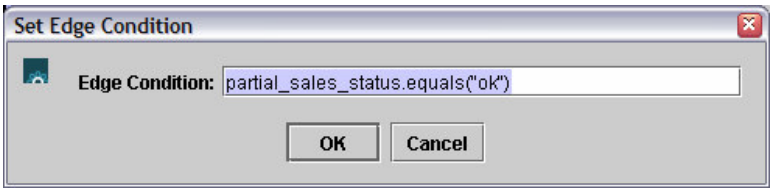


Figure 48: ***Set Edge Condition*** **dialog box for transition between "Accept Order" and "Ship & Report" activities.**

- Add “Cancel Order”: If the user does not confirms the order. AutomaticAndJoin activity. Not anticipable activity activated if the property partial_sales_status = nok.

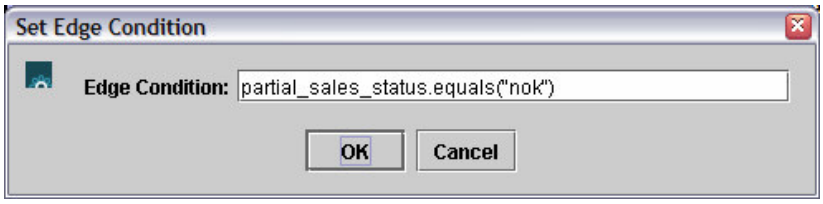


Figure 49: ***Set Edge Condition*** **dialog box for transition between “Accept Order” and “Cancel Order” activities.**

The “Order Processing” project definition is finished with the last operations:

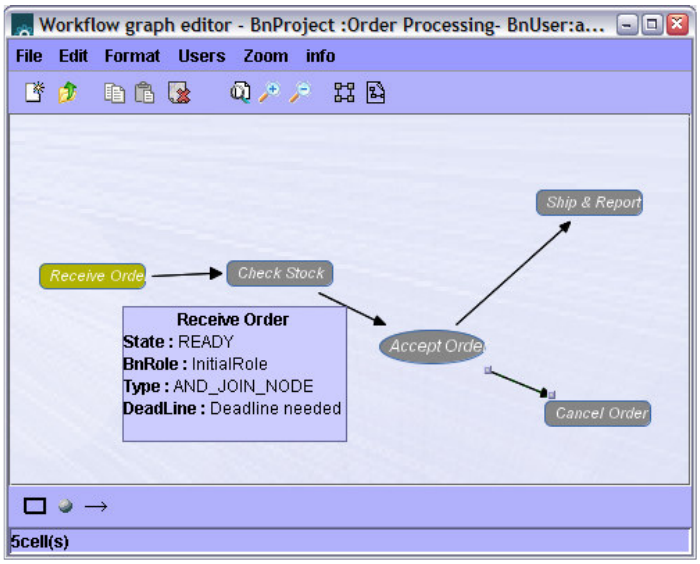


Figure 50: “**Order Processing” project when you finish the process definition.**

Before workflow participants instantiates and executes this Business Process example, you have to assign the correct role to each activity, like we did it for “Customer Service” sub-process.
In this project we have three automatic activities with default role assigned at creation time: “Check Stock”, “Ship & Report” and “Cancel Order”. First activity of the process: “Receive Order” should be assigned to customer so, you have to define a new role called “customer” and assign it to this activity.
The “Accept Order” activity will started by a Customer service employee so you have to create the role “agent” and assign it to this activity.

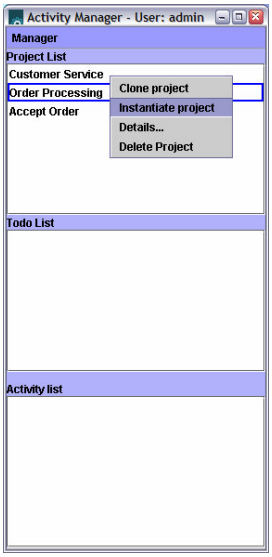
Business Process Execution

The Bonita Manager application allows you to execute workflow projects/instances by using the Worklist graphical component. This tool is useful in order to visualize the state of activities in which a specific user takes part and also to control the projects/instances execution.
When you integrate Bonita with your application you probably want to use your own execution component in order to control the workflow execution process. In our Business Process example this component could be integrated in a web application, so this web application could call Bonita API's to control projects/instances execution.

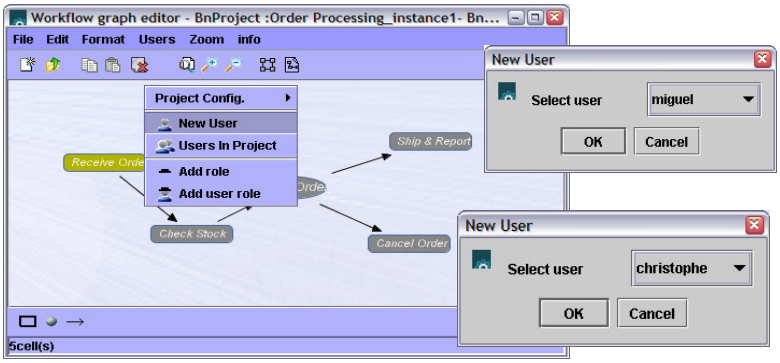
We will use the Bonita Worklist in order to illustrate a common workflow instance execution, so you have to see that as a workflow instance simulation for the Business Process example.

In this example, the administrator of the "Order Processing" workflow project instantiates the project in order to begin a new sales order. For this example we have three users: the workflow administrator, the customer and the employee of customer service. These users could be: *admin*, *miguel* and *christophe*. The first one is the default user who has created the project and other users have been created with Bonita web Interface: *http:///{YourServer}:{YourPort}/bonita* or directly by calling the Bonita UserRegistration API.

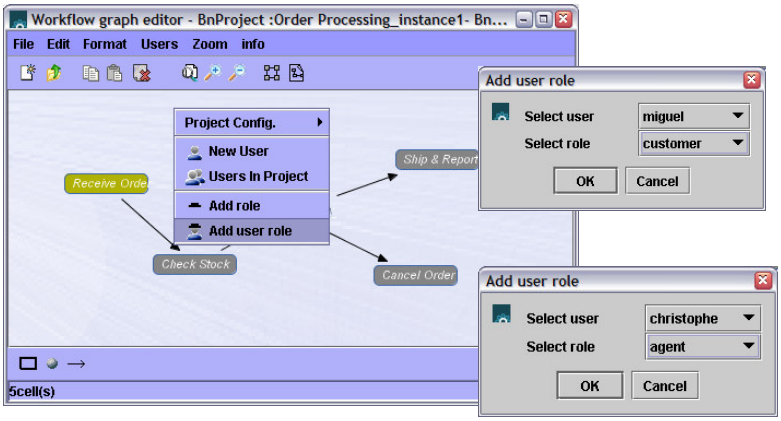
So, let's go to instantiate the "Order Processing" project and then assign to it the previous users with corresponding roles:



The administrator instantiates "Order Processing" project and open new instance called: "Order Processing_instance1".



Administrator adds *miguel* and *christophe* users to this project.



Administrator assigns users to roles.

Figure 51: **"Order Processing" instance and users-roles assignment**

After that we have three participants for this instance with the following roles:

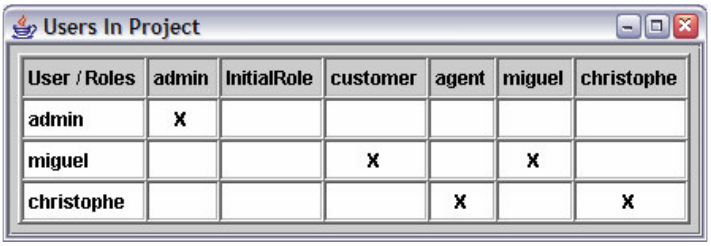


Figure 52: **Users-Roles obtained with "Users In Project" functionality.**

You have to do the same operation for Accept Order instance created automatically by the system[(3)].

Now, if the customer *miguel* launches Manager Application, the "Order Processing_instance1" will appear in his Project List and he can start the "Receive Order" activity by clicking on the right mouse button: "Start Activity". After that, and before terminate the activity he has to set the order data by using Bonita Web Interface[(4)] : right mouse button over activity:"Details…' and go to the edition mode to set the following properties values: "customer_name"="miguel", "product_name"="tv" and "items"="1".

When the customer *miguel* terminates the activity the "Order Processing" project executes automatically "Check Stock" activity, and if order data is correct (all properties are set) the message Stock_Status="ok" message appeared in your server console and sub-process activity is activated:

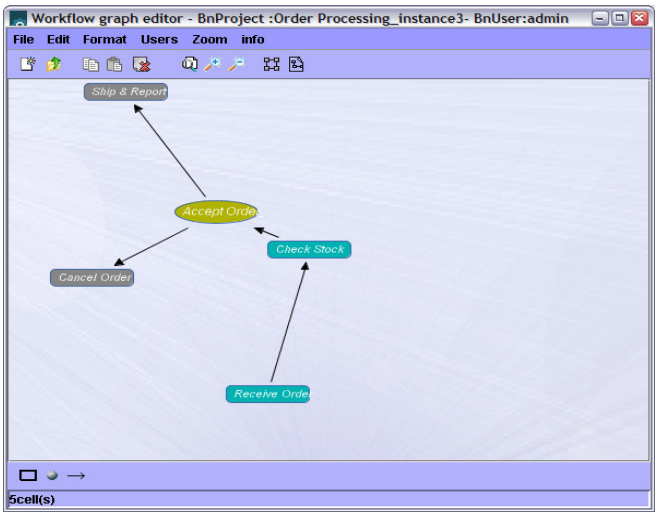


Figure 53: "**Order Processing" instance when the customer terminates first activity without error.**

It's time to play for the customer service employee: *chistophe*. If he starts the "Accept Order" activity, the first activity of sub-process associated is available for the customer *miguel*.

[(3)]In the next version the users-roles assignment operations will be performed automatically by the system by means of specific mappers(LDAP queries, DB queries, properties…) .
[(4)]In your applications these operations will typically performed with your dedicated web form.

The next figures show the state of the instances “Order Processing” and “Accept Order” when the user *Christophe* start “Accept Order” sub-process activity.

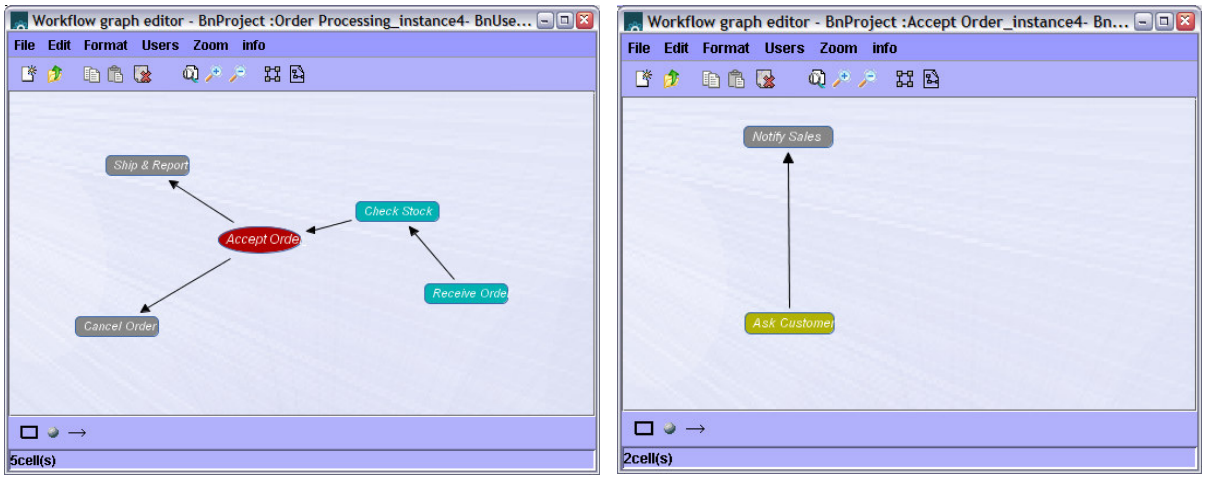


Figure 54: ***Order Processing and Accept Order*** **instances when** ***Christophe*** **starts sub-process activity.**

The first activity of this sub-process is now in ready state, so the customer *miguel* can start the “Ask Customer” activity to visualize his order (this operation shows order data in the server console) and then terminates activity to confirm.

Finally, the customer service employee accepts the user order (start/terminate) “Notify Sales” and terminates “Accept Order” sub-process activity.

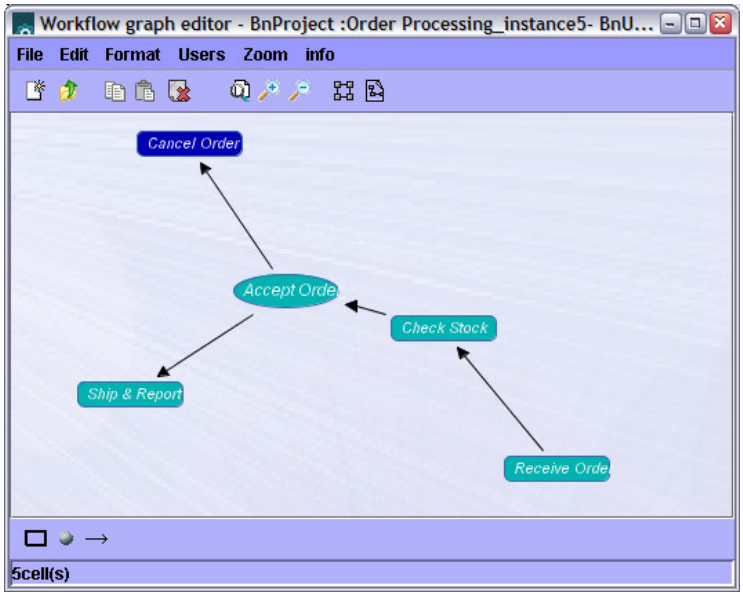


Figure 55: ***Order Processing*** **instance when** ***Christophe*** **terminates sub-process activity.**

This operation launches automatically the “Ship & Report” activity and cancels the other workflow branch by evaluating the “partial_sales_status” property.

Conclusion

The Business Process workflow example shows you some functionalities of Bonita Graphical Components: Worklist and GaphEditor applications. These generic applications are useful in some cases in order to perform the workflow process definition. You can improve and adapt these applications or use them together with your own applications (via Bonita's API), to fit your application needs.

In order to introduce you to Bonita's API, you can take a look at Bonita samples applications available at *$BONITA_HOME/src/main/client/hero/client/samples* directory. Within this directory you have an example of use of Bonita Project Definition API (ProjectSession) and Bonita User Execution API (UserSession).

Index of figures