# Spagic Workflow Getting Started

Author Balestrin Daniele

# 1 Document Goal

The goal of this document is to provide you with an introduction on using Spagic Workflow in Spagic platform by designing, deploying and monitoring a sample workflow process.

# 2 Versions History

| | | | |
|---|---|---|---|
| **Version/Release n°** : | 1.0 | **Date** | May 13, 2009 |
| **Description** | First release (English version) | | |
| | | | |

# 3 Related documents

This document comes after **Spagic Getting Started**, **Spagic Studio User Guide**, **How to cook your Spagic Studio** and **Spagic Console**. It doesn't replace their contents, it expands the explanations to other functionalities of the Spagic platform in order to create and execute jBPM compliant process using Spagic platform.

# 4 Introduction

**Spagic workflow** is a Service Engine installed on Spagic platform that permits the execution of jBPM workflows. This solution integrates all Spagic functions (design, deploy, monitoring).

The type of tasks you can design with Spagic workflow is "human task". Human tasks are tasks that need an external interaction to be executed.

The design and deploy are integrated in Spagic Studio, the monitoring is executed by Spagic Console like all the others Spagic processes.

This document is organized in the following way: in the first section we talk about the installation and configuration of the design and back end part of Spagic workflow, then there are the sections related to design, deploy, run and monitor of a very simple process.

For more detail about Spagic platform see the document **Spagic Getting Started**.

For more details about Spagic Studio environment see the document **Spagic Studio User Guide** and **How to cook your Spagic Studio**.

For more details about Spagic Console, see the document **Spagic Console**.

For more details about jBPM see the project on http://www.jboss.org/jbossjbpm.

# 5 Requirements

| Required Tools | URL for download/Notes | Spagic Studio | Spagic Console |
|---|---|---|---|
| Database | MySQL<br>Oracle | Required | Required |
| Spagic Studio | http://forge.ow2.org/projects/spagic | Required | |
| GraphViz | http://www.graphviz.org/ | Required | |
| jUDDI (optional) | http://ws.apache.org/juddi/ | Optional | Optional |
| freebXML (optional) | http://ebxmlrr.sourceforge.net/ | Optional | Optional |
| JDK 1.5.0_X | http://java.sun.com/ | Required | Required |
| Apache Tomcat 5.5.X or later | http://tomcat.apache.org | | Required |
| Mozilla Firefox 2.0.0.x | http://www.mozilla.com | | Required |

# 6 Installation

The installation and configuration of Spagic workflow comes after the normal installation of Spagic (see *Spagic Getting Started)*.

All the jar and zip files necessary to Spagic workflow are already included in the Spagic platform, so you need only to configure ServiceMix and Spagic Studio to refer to the jBPM tables like explained in the next paragraphs.

## 6.1 ServiceMix configuration

Like in Spagic platform configuration you must configure "*apache-servicemix-{SERVICEMIX_VERSION}\conf\jndi.xml*" file to refer your database. There is a datasource to be considered:

- *java:comp/env/jdbc/jbpm*

You must define a user and a scheme in your database to contain jBPM tables, possibly but not necessarily different to the scheme and user of the metadb.

There's no need to run a DDL script to create jBPM tables, because they are created at the first run or deploy of a jBPM process (hibernate driven creation). The jBPM user must have the grants to create tables on his schema at least at the first deploy or run of a JBPM process.

## 6.2 Spagic Studio configuration

In the figure below are illustrated the properties to be configured in Spagic Studio.

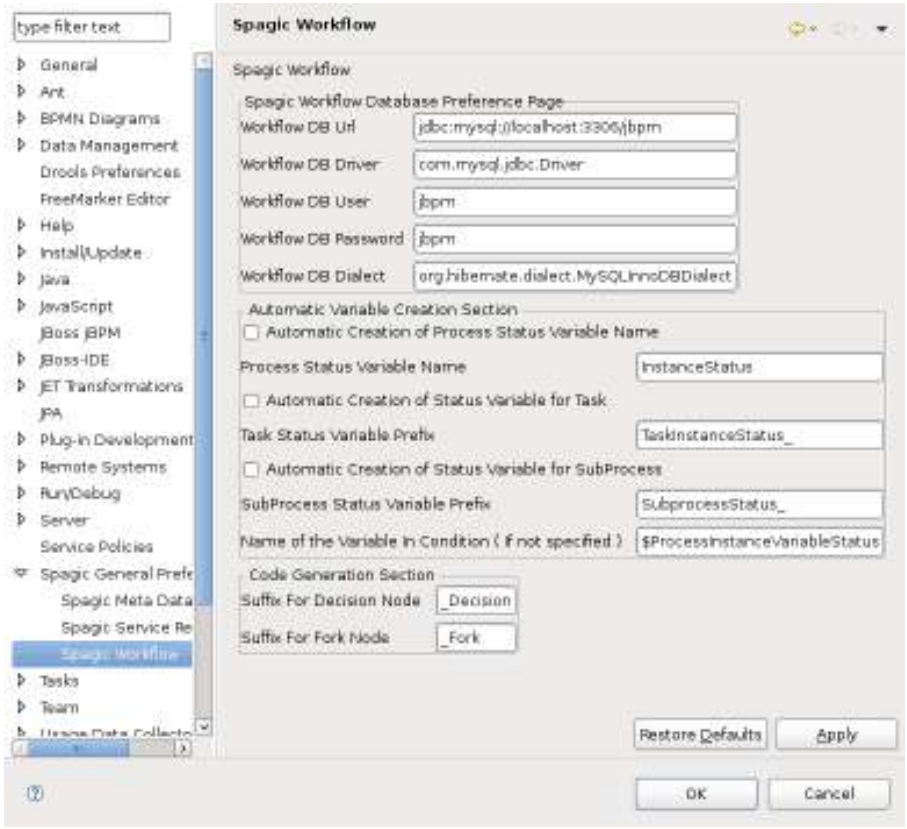



*Figure 1: Spagic Studio workflow properties.*

The first properties are related to the database: location, driver, access and dialect.

Then you find the properties related to the naming convention of the automatic creation of jBPM task and process variables.

Finally there is the naming convention used generate the code for the sections of jPDL processes definition.

# 7 Process development

Process workflow development with jBPM using Spagic Studio is very similar to normal process development, for some other detail refer to Spagic Getting Started.

## 7.1 Create a new project

To create a new project, select *File -> New -> Project -> Spagic 2.0 -> Spagic 2.0 Project*, insert a Project Name, *StartSpagicWorkflow*, click *Finish*.

The **Deployables** directory will contain the **<Process name>.jpdl** file that you can deploy on jBPM. For the others directories in the project created refer to *Spagic Getting Started* document.

## 7.2 Process design

Like in all the other Spagic process design, select the *Integration Process* folder, open the context menu and *New -> Other -> Bpmn Diagram.* Insert a file name, for example *firstProcess.*

After a few step identical to the other you'll obtain a process like the one in the figure below:

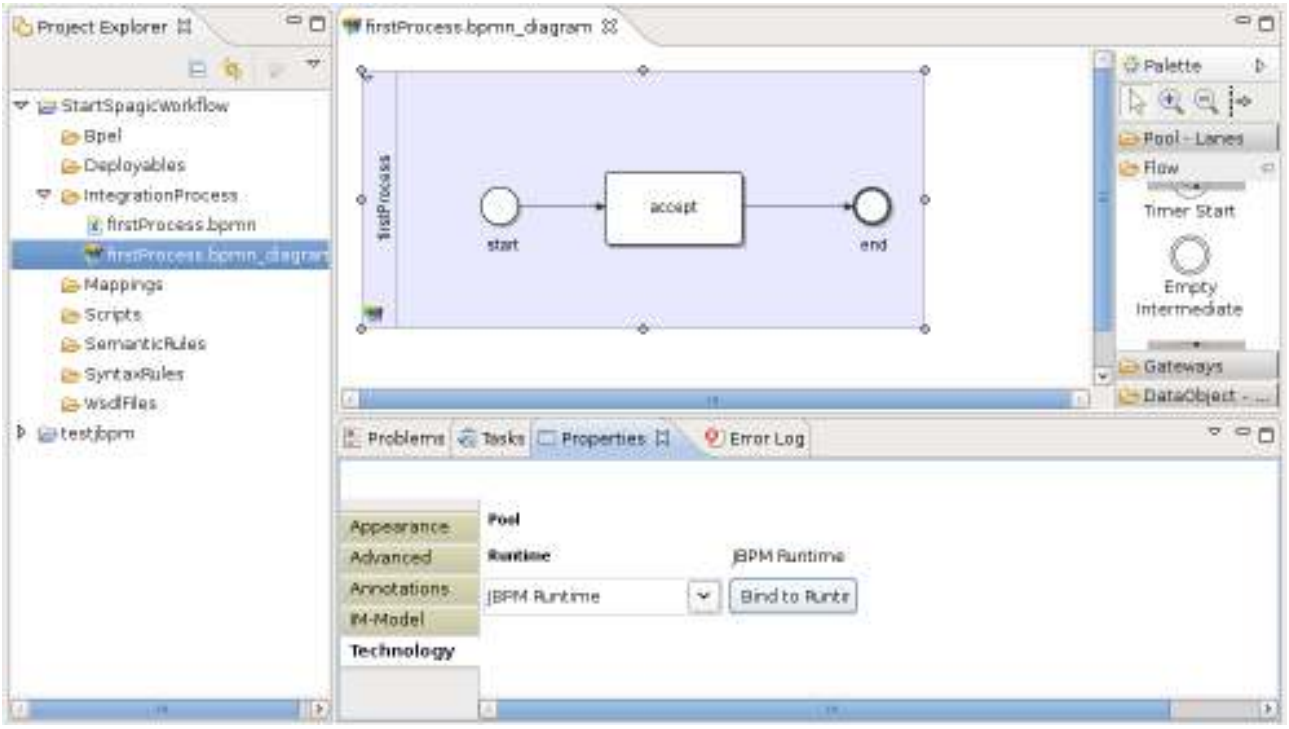



*Figure 2: Firsts steps in process creation*

As you can see in the picture we bind the process to the jBPM Runtime.

## 7.3 Process elements

Any process is composed by different types of elements described below. Annotations are not necessary to define or run process elements, but, in some cases, can aid tasks filtering for the execution.

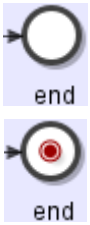
### 7.3.1 Start

Start elements only define the first entry point of a jBPM process.


The pictures above represent the start element in its two forms: without annotation and with annotation. There is no substantial difference between the two forms.

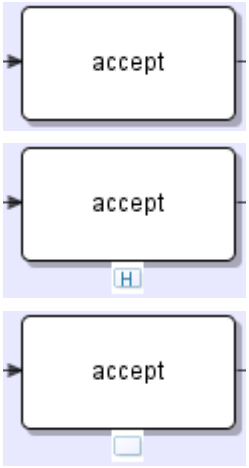
### 7.3.2 End

End elements define end points of a jBPM process.




### 7.3.3 Task

Tasks must have a unique name in the process and can be annotated or not. Annotated tasks can hold attributes that can be used by the workflow API to filter tasks in filtered lists.
The pictures below represent a task without annotation, a task with "HumanTask" annotation and a task with "SubProcess" annotation:




#### 7.3.3.1 HumanTask

Below there is a sample of the association of a task attribute to a *HumanTask*:

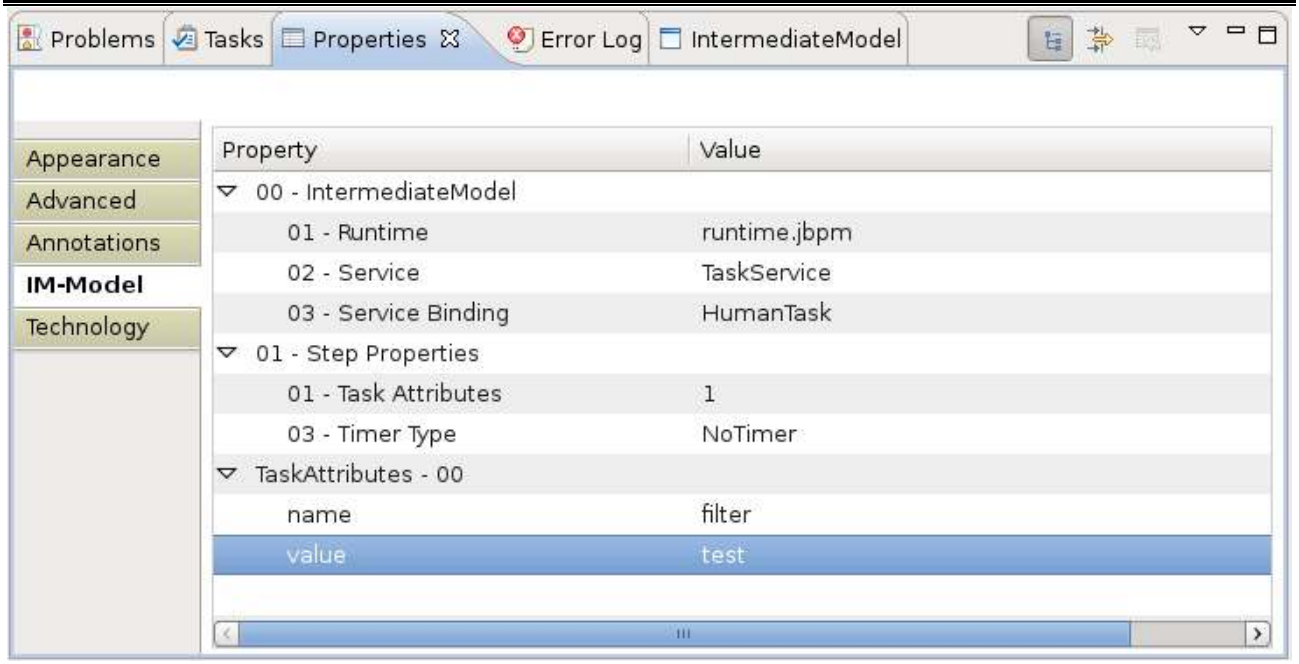



*Figure 3: Task attribute example.*

Annotated tasks can be used to define two types of expiration time of the task:

- **absolute**
- **relative**

**AbsoluteExpirationTimer** defines a date or a date plus hour for the task expiration.

**RelativeExpirationTimer** define an interval of time for the task expiration, starting when the task is available for execution.

### 7.3.3.2 SubProcess

A SubProcess is a task in a process that is a process itself. First of all you must design and deploy the subprocess, and then you can assign that process to a task as *SubProcess*.

When you assign a subprocess annotation to a task you must choose the process associated like in the picture below:

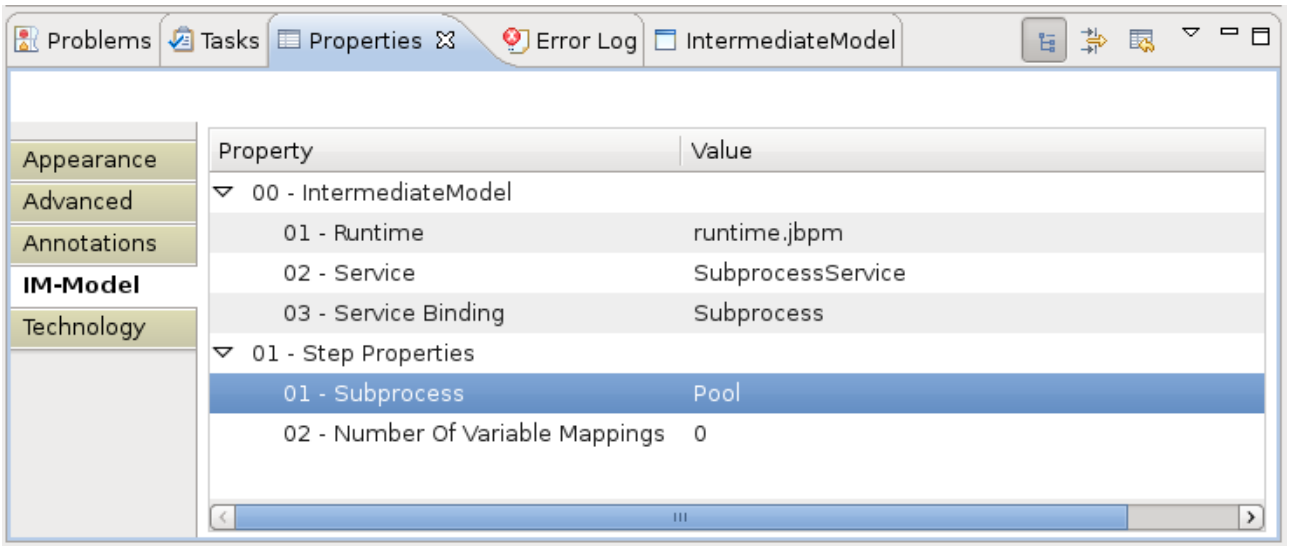



*Figure 4: SubProcess properties.*

Then you can map some variables of the main process to the variables of the subprocess (see next paragraphs for detail).

Every subprocess can be *standalone* or not. A subprocess declared as *not standalone* cannot be executed outside its caller instance.

### 7.3.4 Router

Routers can be omitted in your design process, it's sufficient to exit from a task with two or more arrows. For every arrow exiting from a task you must explicit a condition value, this value is related to the result of the execution of the task, if the value of *Name of the Variable In Condition (If not specified)* is equal to **$TaskInstanceStatus**, or the global result of the process in that moment, if the value of *Name of the Variable In Condition (If not specified)* is equal to **$ProcessInstanceStatus**.

The status of the task or process must be equal to the value of the condition to execute the task; the values are compared as strings.

In the picture below we report a simple example:

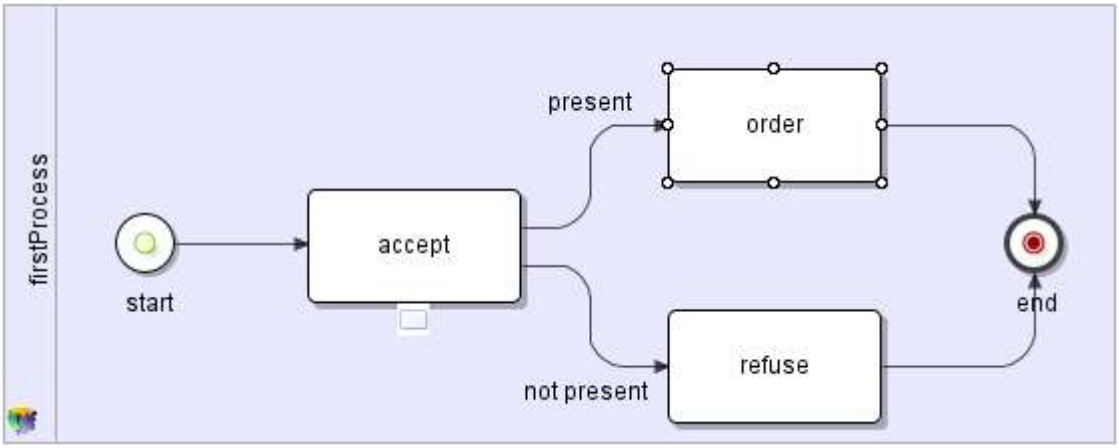



*Figure 5: Router example.*

### 7.3.5 Parallel

The parallel element split tasks of a process into activities that will be executed contemporaneously. The join parallel node must be explicitly added to the process.

There are two type of join node:

- **andJoin** where the next action will be executed when all the tasks of the split end their actions
- **orJoin** (Structured Discriminator pattern) where the next action will be executed when at least one task of the split end its action

Whereas andJoins are the default behavior, orJoin must be annotated.

The figure below report a simple example of a parallel process with an andJoin:

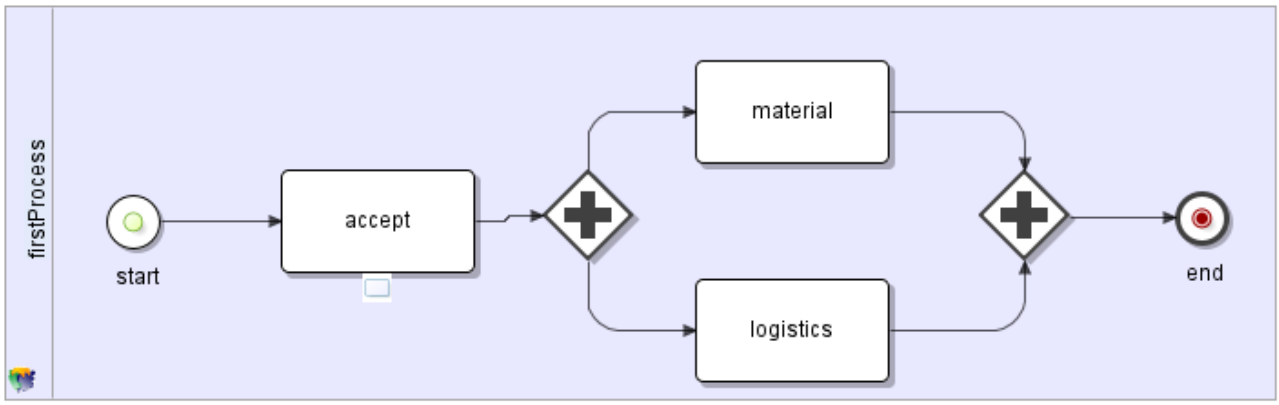



*Figure 6: Parallel example.*

### 7.3.6 Lane

Lanes are intended to assign tasks execution to some users that have the role and permissions to run those actions.

In the picture below there is an example where the process is split in two zones the part up where there are tasks assigned to the users of the warehouse and the part down where there are trucks' tasks. The tasks that are between the zones are executed by each role *warehouse* and *trucks*.

Red highlights and green texts are present only for clearance.

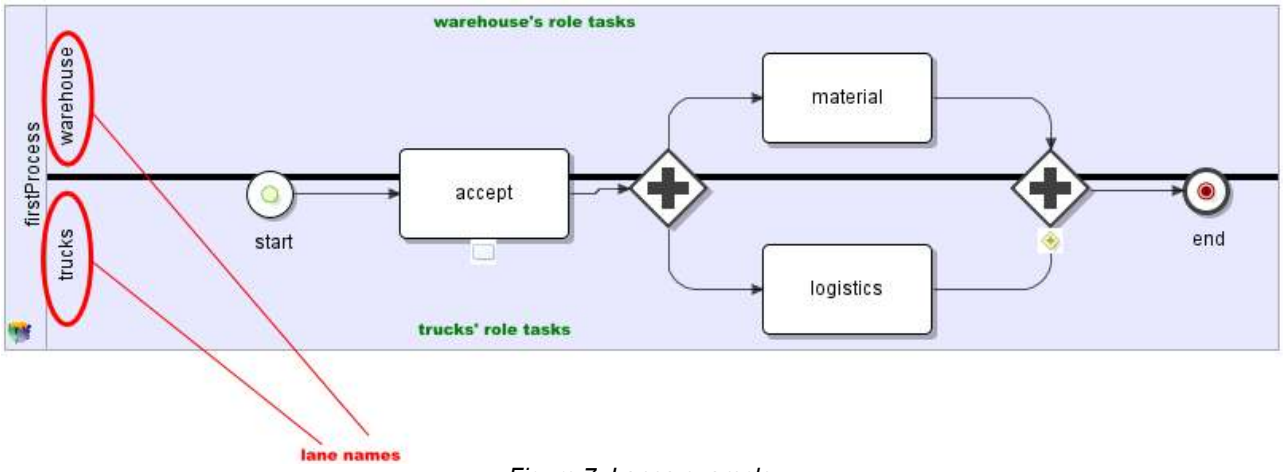



*Figure 7: Lanes example.*

To use lanes is necessary to modify the standard *jbpm.cfg.xml* file adding at least some code like below:

```
</jbpm-context>
  <!--
       We added the service bellow to manage roles on tasks execution.
        This is a very simple implementation of org.jbpm.security.AuthorizationService.
        Users and their roles are stored in a xml file that resides on classpath.
  -->
  <service name="authorization">
    <factory>
     <bean class="org.spagic.workflow.jbpm.authservice.SimpleAuthorizationServiceFactory">
            <field name="file">
                    <string value="authorizations.xml"></string>
            </field>
          </bean>
   </factory>
  </service>
</jbpm-context>
  <!--
```

```
        The org.spagic.workflow.jbpm.authservice.PermissionTaskInstance is an extention of the
        default TaskInstance of jBPM that call the authorization service to execute tasks.
        This PermissionTaskInstance needs a new declaration in hibernate.cfg.xml that points to its
hbm file.
    -->
  <bean   name="jbpm.task.instance.factory"
class="org.spagic.workflow.jbpm.authservice.PermissionTaskInstanceFactory" singleton="true" />
```

This is a very simple implementation of the authorization service interface of jBPM we provide. The concept is to associate a user (normally a string provided by the application caller) with a role (normally a string reported as name of the lane).

All that is need is to implement a *checkPermission* method like in the *SimpleAuthorizationService* class provided. The roles automatically generated by the design model are stored in a *Set* of *PooledActor* objects in *actorId* property.

In order to deploy and run the lane processes we need to modify also the *hibernate.cfg.xml* file of jBPM adding the mapping listed below:

```
  <!--
            The hbm below represents the the relation between PermissionTaskInstance and the
            TaskInstance that it extends from the point of view of the database.
   -->
  <mapping resource="org/spagic/workflow/jbpm/authservice/PermissionTaskInstance.hbm.xml"/>
```

## 7.3.7 Variables

Variables are Data Objects which value represents the status of a process instance. At this time the type of variables supported by Spagic Studio Workflow at design time is *string* data object. The variables are added to the context of a process instance.

At the start of a process all the variables are set to empty value.

All variables are globally defined for every task of the process and their values are set or get via *ControlAPI*.

### 7.3.7.1 Variables definition

The picture below illustrates an example of variable definition at design time.

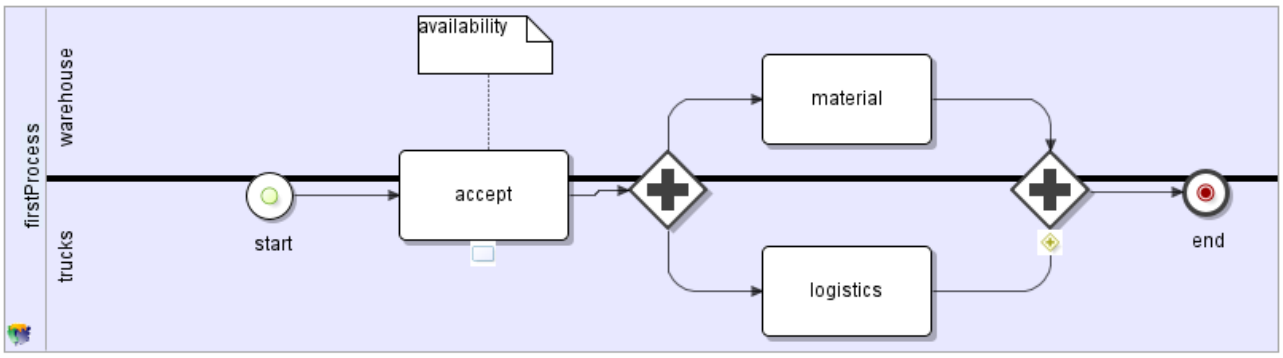
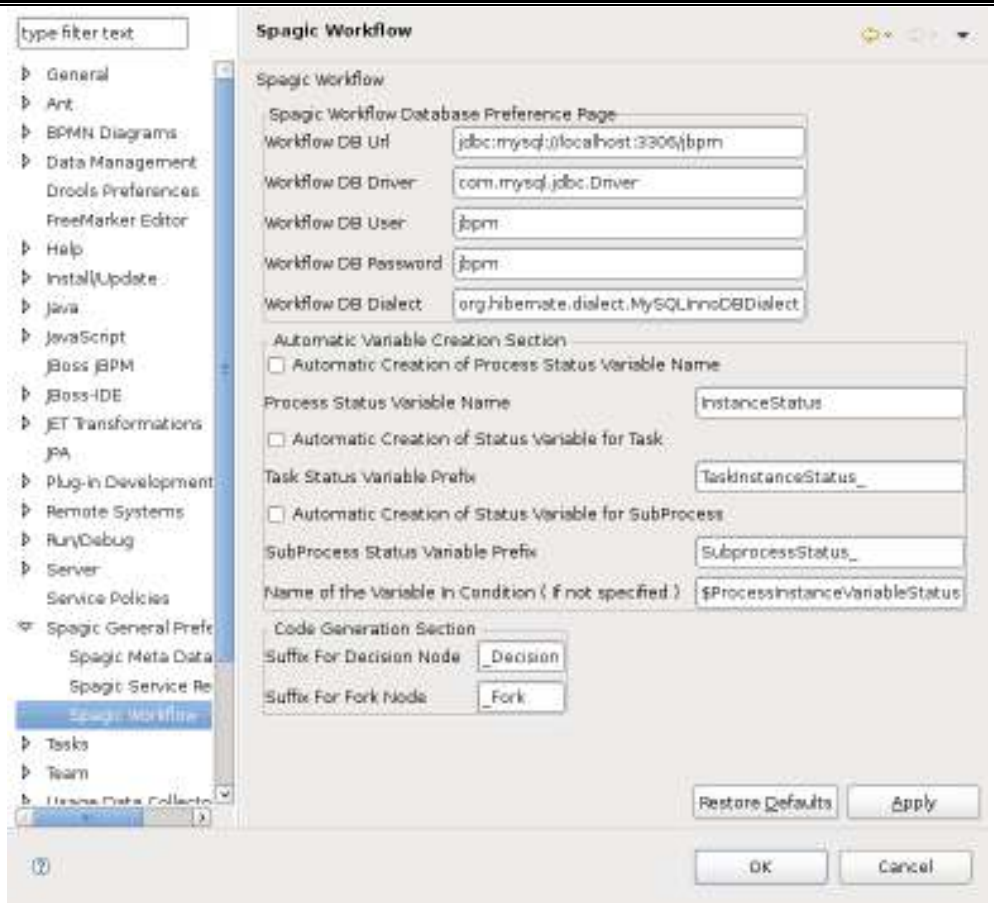



*Figure 8: Variable example. In this example we added the "availability" variable.*

The variable is only graphically assigned to a task.

### 7.3.7.2 Automatically defined variables

Automatic generation of variables is enabled via Spagic Workflow Preferences like in figure below:




The name of variables will have a prefix that you can decide followed by the task name.

## 7.4 Deploy

The first step to deploy a process is to generate workflow definition (jPDL) like in the figure below:

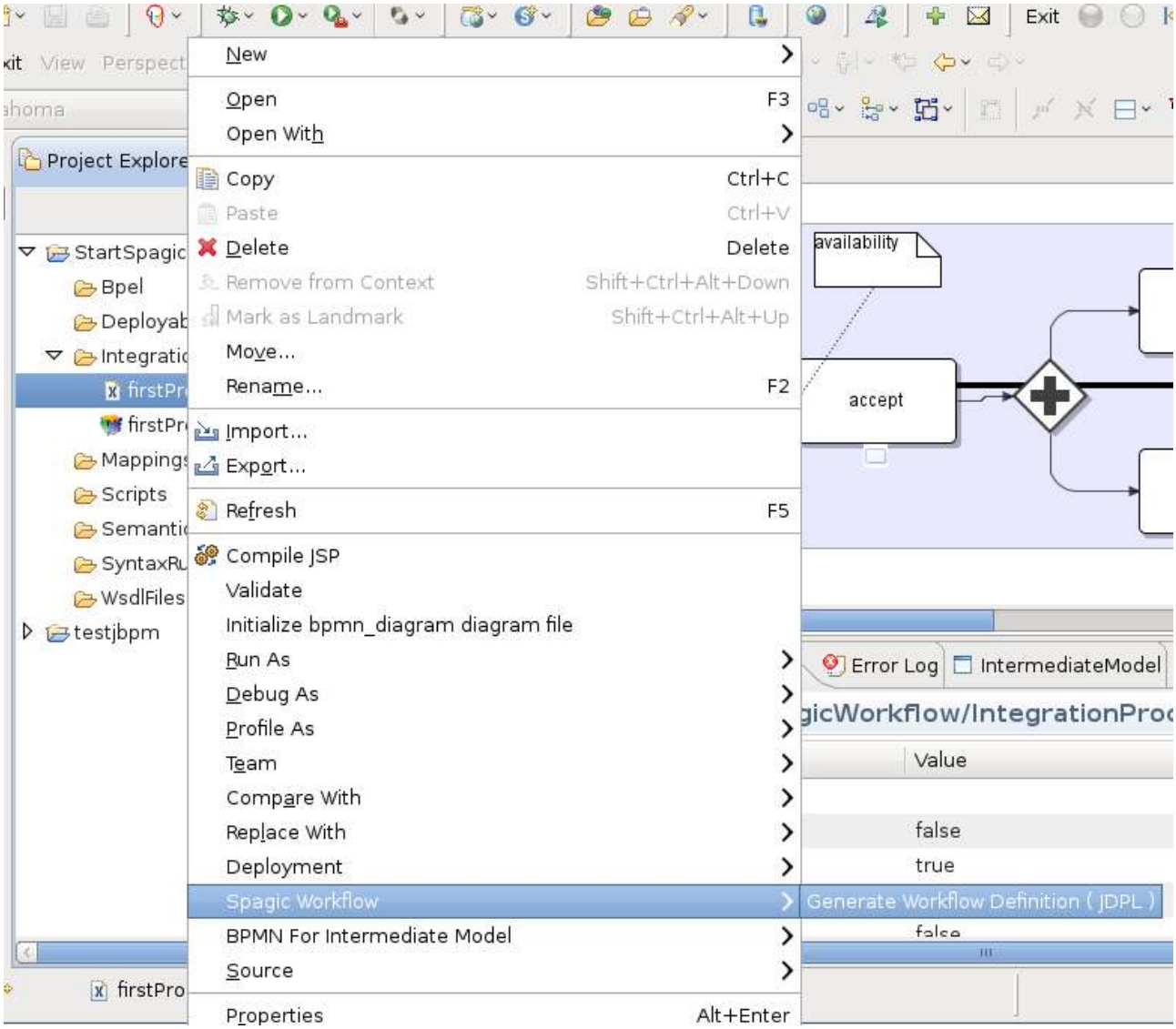



*Figure 9: jPDL generation example.*

This operation will generate an intermediate model file in the same directory of your bpmn file, *IntegrationProcess*, and a jpdl file in the *Deployables* directory. The jpdl file is necessary to deploy the process in the jBPM repository.

In the next figure is illustrated the deploy operation:

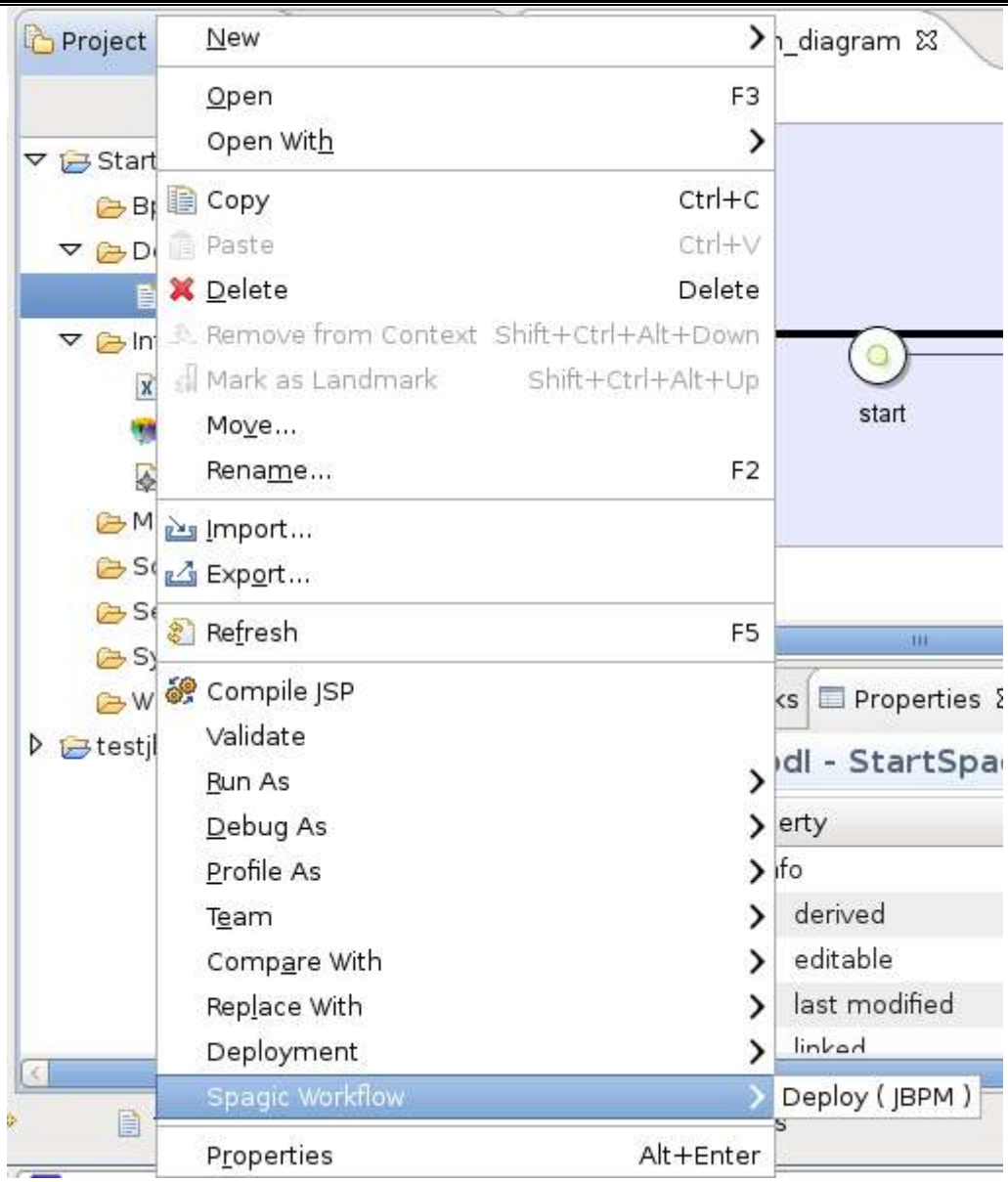



*Figure 10: Deploy example.*

Every time you deploy a process, jBPM versions it. Every new instance of a process is of the last version deployed. BPMN versioning is irrelevant to jBPM versioning.

# 8 Process execution

Once you deployed a process you can call it via Spagic API.

This API is composed by three principal elements: *ProcessEngine, QueryAPI* and *ControlAPI.*

Below we describe in brief what this interface are for, refer to Javadoc for the details.

## 8.1 ProcessEngine

The process engine implementation defines which workflow engine will be used: local or remote engine that encapsulate jBPM.

The process engine is a container of the control and query APIs.

## 8.2 QueryAPI

QueryAPI is used to set or retrieve process elements like processes, tasks, subprocesses and statuses.

## 8.3 ControlAPI

ControlAPI is the core API to run processes, execute tasks and obtain all the process variables.

## *8.4 API usage example*

In the following section you find some sample code to execute (and complete) the sample process used in paragraph Process design [7.2].


```
public static void main(String[] args) {
        IProcessEngine pe = new WSProcessEngine("http://localhost:9090/wfengine/");
        IControlAPI controlAPI = pe.getControlAPI();
        long pid = controlAPI.startByProcessName("firstProcess");
        Variable[] vars = new Variable[1];
        vars[0] = new Variable();
        vars[0].setName("StringVariable");
        vars[0].setValue("some value");
        controlAPI.executeTaskWithStatus(pid, "accept", "TaskEXECUTED", vars);
}
```

You need the following libraries in the client classpath (if you use the Spagic remote engine):

- axis.jar
- commons-discovery-0.2.jar
- commons-logging-1.0.4.jar
- dom4j.jar
- jaxrpc.jar
- saaj.jar
- wsdl4j-1.5.1.jar
- xstream-1.3.jar
- **spagic-workflow-api-{SPAGIC-VERSION}.jar**
- **spagic-workflow-ws-client-{SPAGIC-VERSION}.jar**

You can find them in the Spagic distribution.

Here we report a more complex sample:

```
public static void main(String[] args) {
        IProcessEngine pe = new WSProcessEngine("http://localhost:9090/wfengine/");
        IControlAPI controlAPI = pe.getControlAPI();
        long pid = controlAPI.startByProcessName("Processo 1");
        controlAPI.abort(pid);
        Variable[] vars = new Variable[1];
        vars[0] = new Variable();
```

```
            vars[0].setName("Variable0Stringa");
            vars[0].setValue("UnValoreStringa");
            controlAPI.executeTask(pid, "Task1", vars);
            controlAPI.executeTaskWithStatus(pid, "Task1", "TaskOK", vars);
            controlAPI.executeTask(pid, 1,vars);
            controlAPI.executeTaskWithStatus(pid, 1, "TaskOK", vars);
            controlAPI.executeTaskByActor("Manager", pid, 1, vars);
            controlAPI.executeTaskByActor("Manager", pid, "Task1", vars);
            controlAPI.setGlobalVariable(pid, "Variable1", "Valore1");
            Variable v = controlAPI.getGlobalVariable(pid, "Variable1");
            Variable[] resultingVars =  controlAPI.getGlobalVariables(pid);
            controlAPI.setGlobalVariables(pid, vars);
    }
```

# 9 Process monitoring

All the processes deployed and started are visible on Spagic Console, workflow processes are highlight by another icon like in the figure:

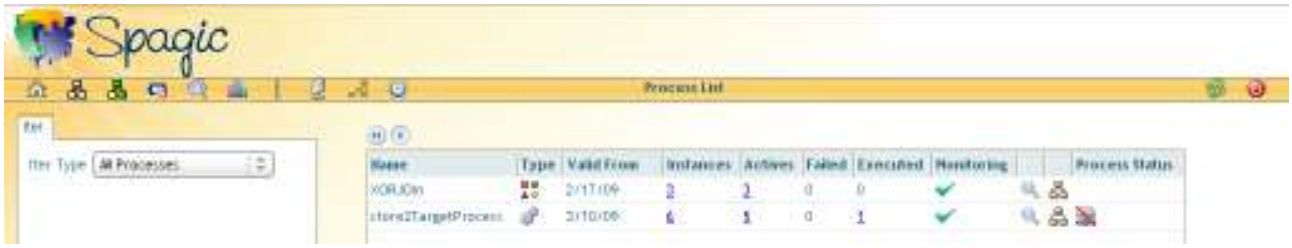



*Figure 11: Process list on Spagic Console. XORJOin process is a workflow process.*

For each process instance it's possible to view its status and the values assumed by its variables at that moment like in the figure below:

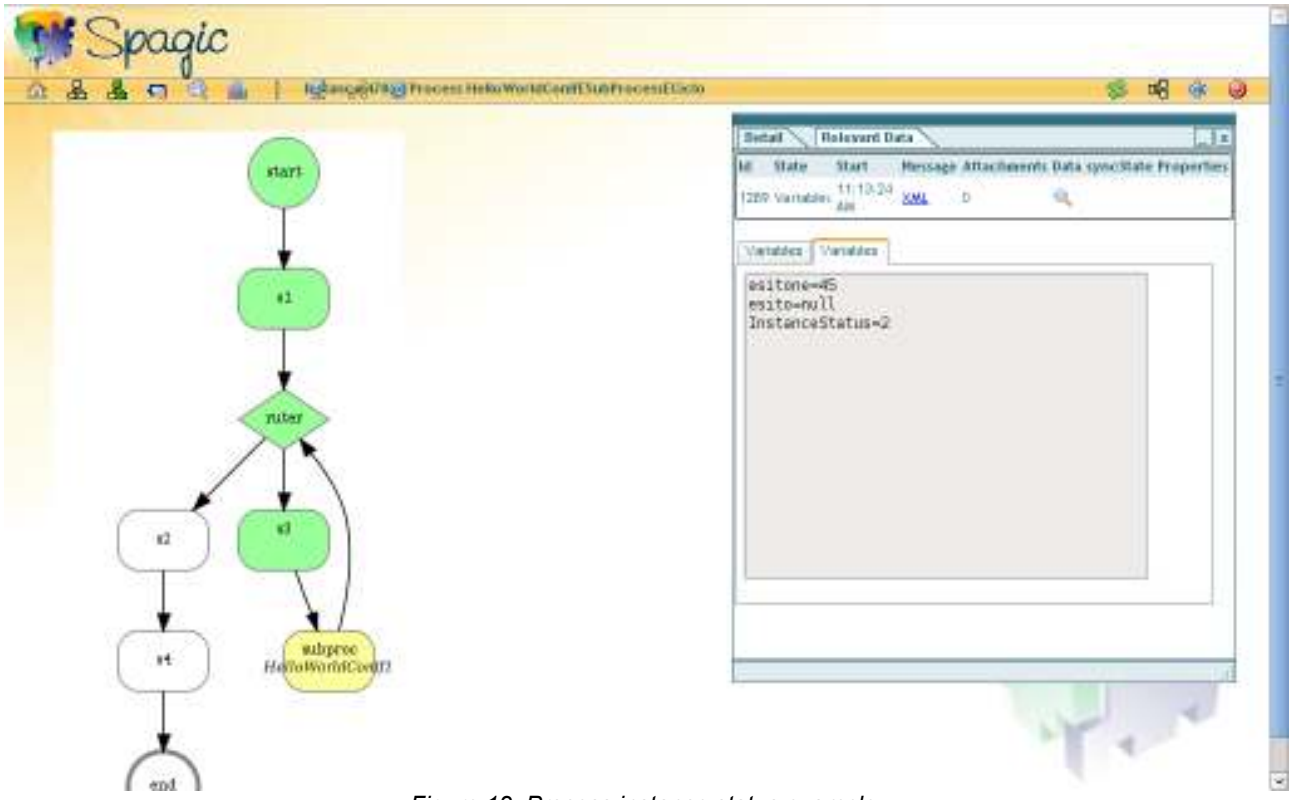



*Figure 12: Process instance status example.*

Each green task was been executed and the task in yellow color is only started, but not executed. If you click on a task you can see its detail and relevant data like in the picture. Among the relevant data there are the variables with their values.

If a task is annotated as SubProcess you can click over and see the status of the subprocess in the same way of the status of its main process.