# JBPM: TUTORIAL PARTE 2

Autor: Guzmán Llambías
Última actualización: 1 Septiembre 2006

## Introducción

El objetivo de este tutorial es comprender por medio de algunos ejemplos, el uso de constructores avanzados que provee jBPM. Una vez finalizado este tutorial el usuario será capaz de crear procesos en jBPM incluyendo Tareas, Swimlanes, Asignación de Tareas a usuarios y Persistencia del proceso en una Base de datos.

## Prerrequisitos

Se asume que el usuario realizó el '*Tutorial sobre jBPM parte 1*' o ya tiene conocimientos básicos de jBPM. Además, se asume que el usuario está familiarizado con el uso de bases de datos en PostgreSQL.

Se recomienda fuertemente leer los capítulos 9, 11 de la guía del usuario que provee jBPM, versión 3.1.

## Requerimientos del software

Es necesaria la instalación del siguiente software:

- Eclipse IDE versión 3.2 http://www.eclipse.org/downloads/
- JBoss IDE plugin http://www.jboss.com/products/jbpm/downloads
- JBPM starter kit 3.1.2 http://www.jboss.com/products/jbpm/downloads
- PostgreSQL 8.1.4 DBMS http://www.postgresql.org/ftp/source/v8.1.4

## Referencias

- Guia del usuario versión 3.1, capítulos 5, 6, 7, 8, 9, 10, 11
  http://docs.jboss.com/jbpm/v3/userguide/index.html
- New JBPM Getting Started
  http://www.jboss.com/index.html?module=bb&op=viewtopic&t=80287
- Foro JBoss jBPM
  http://www.jboss.com/index.html?module=bb&op=viewforum&f=217

## Contenido

- Creación de un proceso en jBPM
- Deploy de un proceso en una base de datos PostgreSQL
- Levantar un proceso desde la base de datos PostgreSQL
- Asignación de tareas mediante uso de swimlanes
- Asignación de tareas mediante uso de handlers
- Asignación de tareas mediante uso de Pooled Actors
- Crear un driver para testear el proceso
- Testear el proceso

## Creación de un proceso en JBPM

1. Crear un proyecto jBPM con el nombre *Tutorial_avanzado*.
2. Crear una carpeta en dicho proyecto y llamarla *procesos*.
3. Crear el proceso que se muestra en la Figura 1 mediante el uso de tareas y transiciones. Llamarlo *proceso_avanzado* y colocarlo en la carpeta *procesos*.

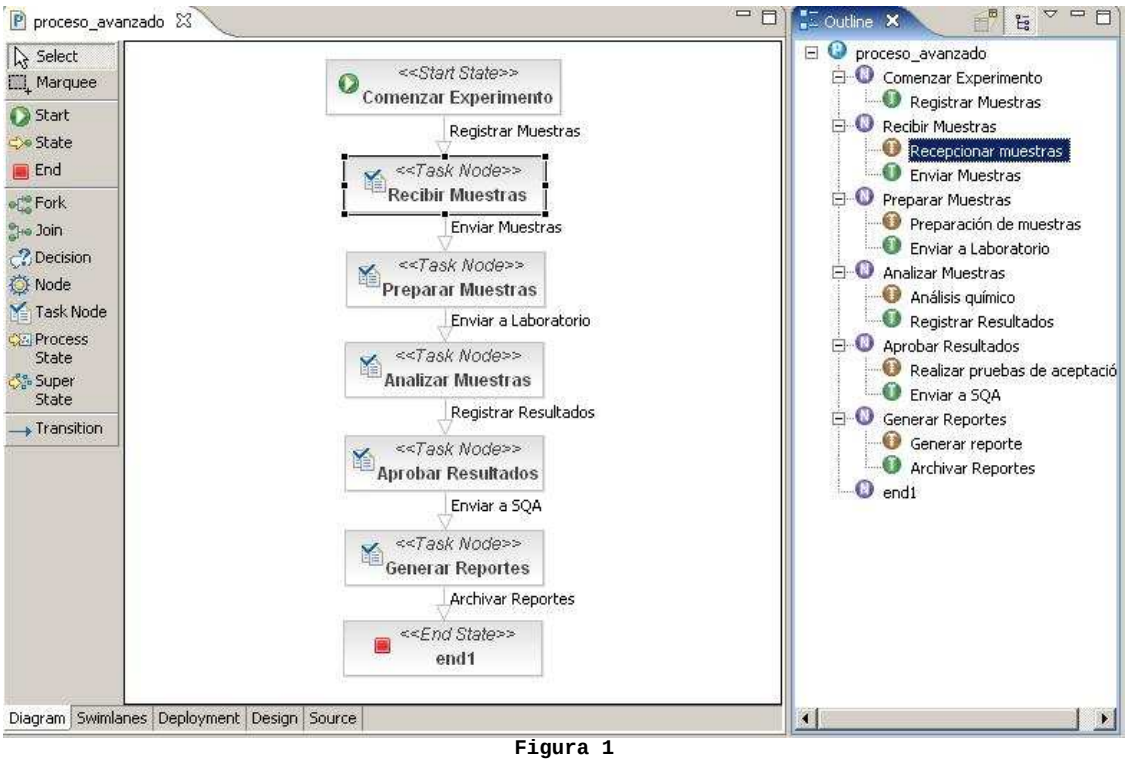


**Figura 1**

4. Agregar al nodo *Recibir Muestras*, la tarea *Recepcionar muestras*.
5. Agregar al nodo *Preparar Muestras*, la tarea *Preparación de muestras*.
6. Agregar al nodo *Analizar Muestras*, la tarea *Análisis químico*.
7. Agregar al nodo *Aprobar Resultados*, la tarea *Realizar pruebas de aceptación*.
8. Agregar al nodo *Generar Reportes*, la tarea *Generar reporte*.

## Deploy de un proceso en una base de datos Postgres

1. Crear la clase *TestJBPM* subclase de *TestCase* en el package *uy.edu.fing.example.jbpm.test*.
2. Agregarle el atributo *jbpmConfiguration* instancia de la clase *JbpmConfiguration*.
3. Sustituir el código del constructor de la clase, por el siguiente:

```
super();
FileInputStream fis = new FileInputStream("src/config.files/jbpm.cfg.xml");
              jbpmConfiguration = JbpmConfiguration.parseInputStream(fis);
```

4. Crear el método *testDeployDefincionProceso* y agregarle el siguiente código:

```
// Ver la configuración de la persistencia más arriba
JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();

try {
        FileInputStream fis = new

        FileInputStream("procesos/proceso_avanzado/processdefinition.xml");
        ProcessDefinition definition = ProcessDefinition.parseXmlInputStream(fis);

        // Deploy del proceso en la base
        jbpmContext.deployProcessDefinition(definition);

} finally {
        // Terminar con el contexto
        // Se hace un flush de los insert que hacen el deploy del proceso en la base
        jbpmContext.close();
}
```

5. Agregar el siguiente código al archivo de configuración *jbpm.cfg.xml*, ubicado en la carpeta *src/config.files*:

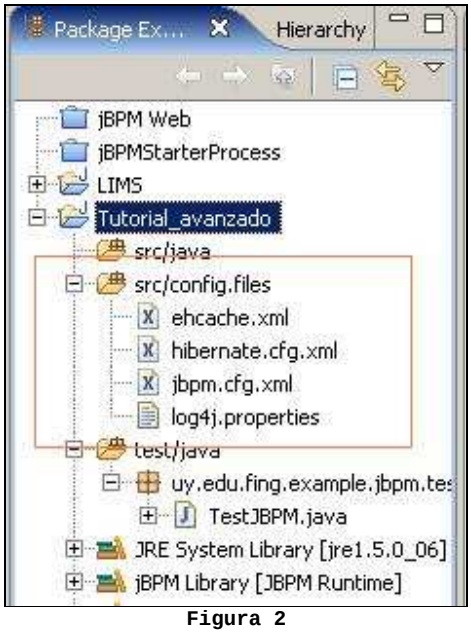


**Figura 2**

```
<jbpm-context>
       <service name='persistence'
            factory='org.jbpm.persistence.db.DbPersistenceServiceFactory'/>
</jbpm-context>

<string name='resource.hibernate.cfg.xml' value='hibernate.cfg.xml' />
<string name='resource.business.calendar'
       value='org/jbpm/calendar/jbpm.business.calendar.properties' />
<string name='resource.default.modules'
       value='org/jbpm/graph/def/jbpm.default.modules.properties' />
<string name='resource.converter'
       value='org/jbpm/db/hibernate/jbpm.converter.properties'/>
<string name='resource.action.types' value='org/jbpm/graph/action/action.types.xml' />
<string name='resource.node.types' value='org/jbpm/graph/node/node.types.xml' />
<string name='resource.varmapping' value='org/jbpm/context/exe/jbpm.varmapping.xml' />
```

6. Crear una base de datos en postgreSQL con nombre *jbpmDB*
7. Correr el script *postgresql.create.sql* en dicha base. Este script, crea las tablas que serán utilizadas por jBPM. Dicho script se encuentra en la carpeta *jbpm-starters-kit-3.1.2\jbpm-db\build\postgresql\scripts*
8. Modificar las siguientes propiedades del archivo *hibernate.cfg.xml*, ubicado en la carpeta *src/config.files*:
   - hibernate.dialect: org.hibernate.dialect.PostgreSQLDialect
   - hibernate.connection.driver_class: org.postgresql.Driver
   - hibernate.connection.url: jdbc:postgresql://*host_name*:*port/jbpmDB*
   - hibernate.connection.username: *user_name*
   - hibernate.connection.password: *password*
9. Crear la librería *Hibernate* y agregarla al build path del proyecto. Esta librería debe contener los siguientes archivos jars:
   - hibernate-entitymanager.jar
   - hibernate3.jar
   - hibernate-annotations.jar
   - postgresql-8.0-310.jdbc3.jar
10. Correr el jUnit y verificar el deploy del proceso en la tabla *jbpm_processdefinition* de la base *jbpmDB*.

## Levantar un proceso desde la base de datos Postgres

1. Crear el método *testLevantarProceso*
2. Agregar el siguiente código a dicho método:

```
JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
try {
	GraphSession graphSession = jbpmContext.getGraphSession();
	ProcessDefinition processDefinition =
		graphSession.findLatestProcessDefinition("proceso_avanzado");

	// Con la definición del proceso que obtenemos de la base
	// creamos una ejecución del mismo
	ProcessInstance processInstance =
		new ProcessInstance(processDefinition);

	assertEquals(
		"La instancia se encuentra en el estado inicial",
		processInstance.getRootToken().getNode().getName(), "Comenzar
Experimento");
	//Enviamos la señal para salir del nodo start
	processInstance.signal();

	assertEquals(
		"La instancia se encuentra en el estado Recibir Muestras",
	processInstance.getRootToken().getNode().getName(), "Recibir Muestras");

	// Guardamos la instancia del proceso en la base de datos de forma tal
	// de mantener el estado de ejecución del proceso
	jbpmContext.save(processInstance);

} finally {
	// Finalizar el contexto de jbpm
	jbpmContext.close();
}
```

3. Correr los test de prueba con jUnit.

## Asignación de tareas mediante uso de swimlanes

1. Seleccionar la viñeta *swimlanes* ubicada en la parte inferior del dibujo.

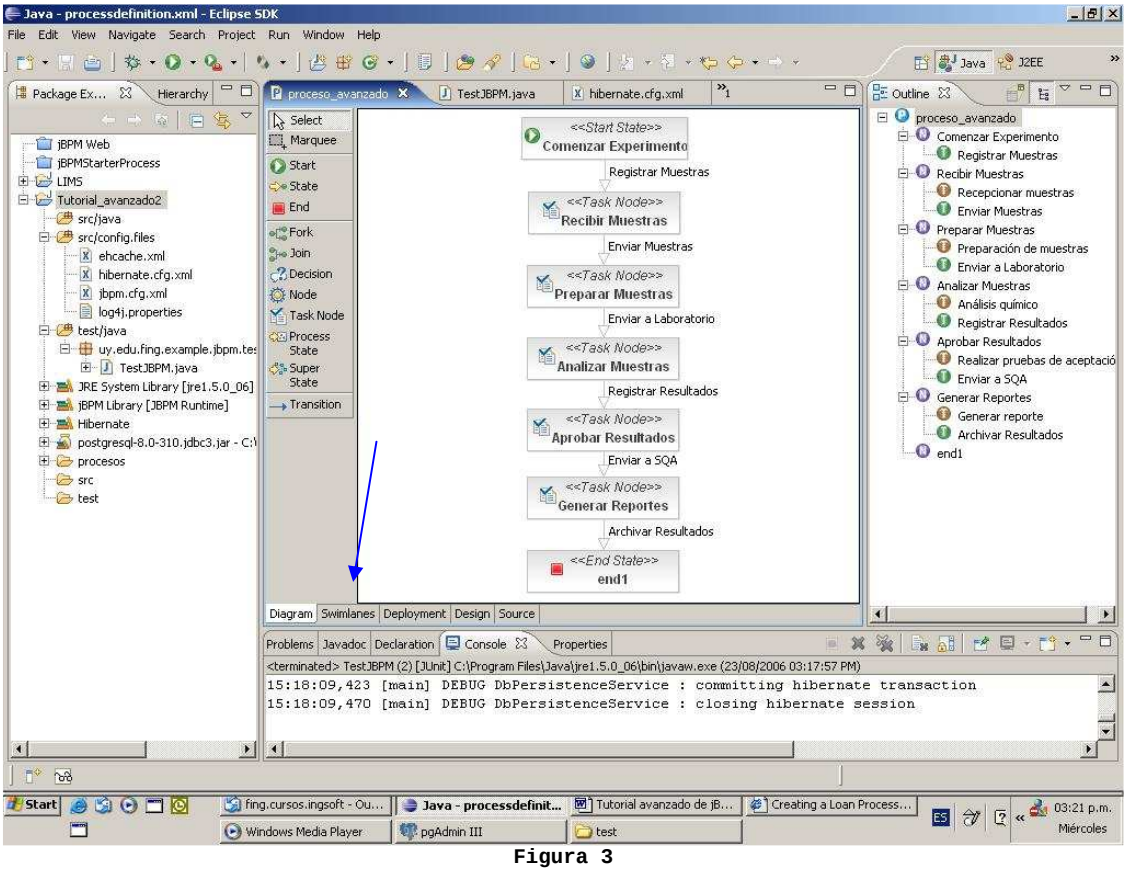


**Figura 3**

2. Presionar el botón *Add*
3. Modificar el nombre por *Operario* y seleccionar *Expression* en *Assignment Type*.
4. En el campo *expression*, ingresar *user(Juan)*.
5. Ingresar en la tabla *jbpm_id_user* los usuarios del proceso corriendo las siguientes transacciones SQL:

```
insert into jbpm_id_user values (10, 'U', 'Pablo', 'pablo@jbpm.org', 'mi_password');
insert into jbpm_id_user values (11, 'U', 'Juan', 'juan@jbpm.org', 'otro_password');
insert into jbpm_id_user values (12, 'U', 'Maria', 'maria@jbpm.org', 'otro_password');
insert into jbpm_id_user values (13, 'U', 'Lucia', 'lucia@jbpm.org', 'otro_password');
```

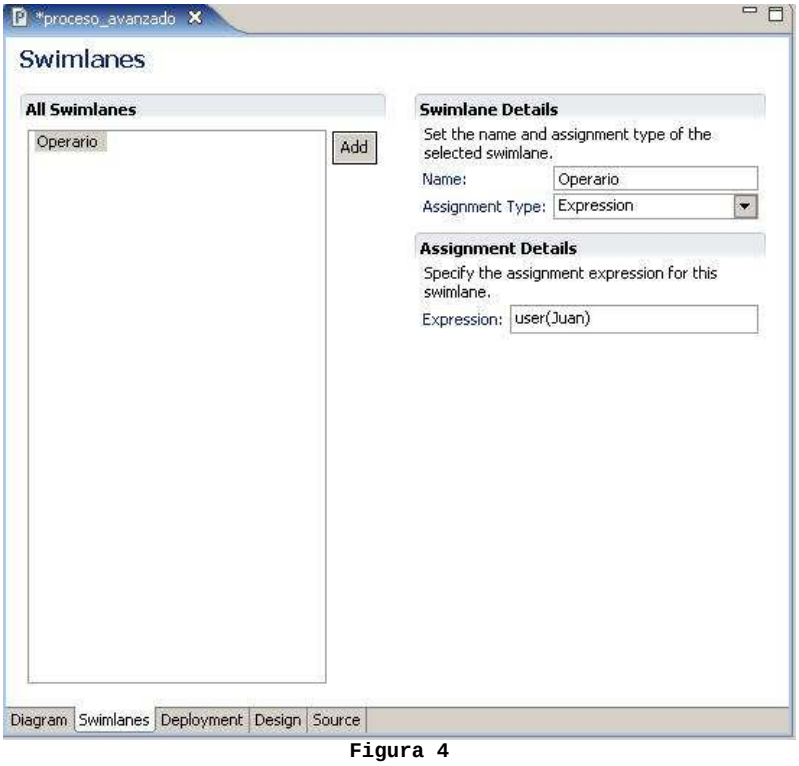


**Figura 4**

6. Crear una nueva *swimlane* de nombre *Laboratorista*, del tipo *Expression* y en el campo *expression* colocar *user(Pablo)*

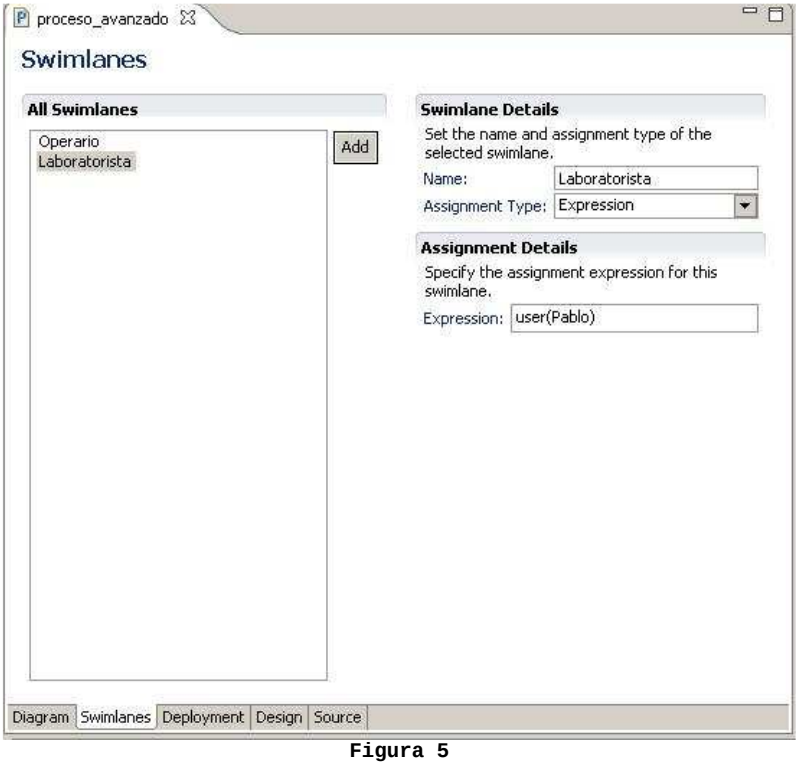


**Figura 5**

7. Hacer clic derecho en la tarea *Recepcionar Muestras* y seleccionar *Propiedades.*
8. Seleccionar *Assignment*.
9. Elegir como tipo de asignación *Swimlane.* Dentro de los *swimlanes* elegir *Operario* como se muestra en la Figura 6 y presionar el botón OK.

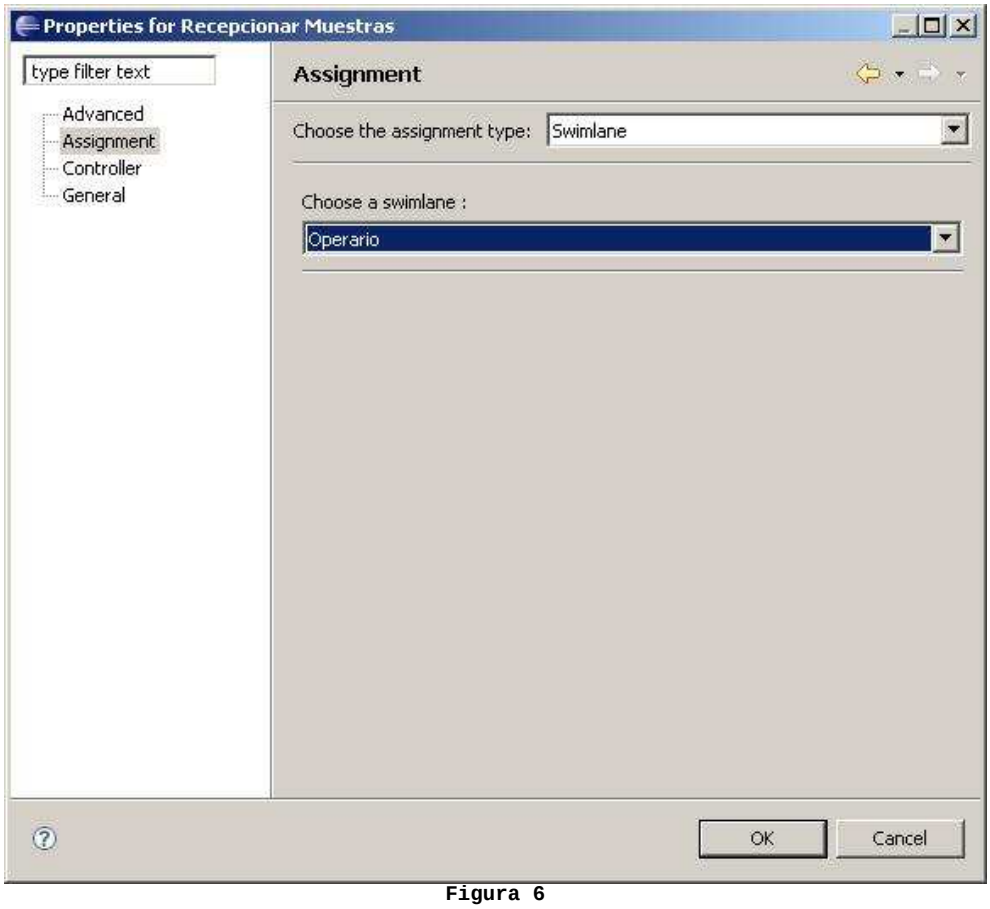


**Figura 6**

10. De la misma forma asignar a la tarea *Preparación de muestras* el swimlane *Operario*.
11. Asignar a la tarea *Análisis químico* el swimlane *Laboratorista*.

## Asignación de tareas mediante uso de handlers

1. Crear el package *uy.edu.fing.example.jbpm*
2. Crear la clase *AssignmentExample* que implementa la interfaz *AssignmentHandler* como se muestra en la Figura 7.


**Figura 7**

3. Asignar el siguiente código al método *assign*:

```
assignable.setActorId("Maria");
```

4. Ir a las propiedades de la tarea *Generar Reporte*, elegir la opción *Assignment* y luego seleccionar como *assignment type* a *Handler*.
5. Ingresar la clase *AssignmentExample* como handler, ya sea seleccionándolo con el botón *Browse…* o ingresándolo directamente en el campo de texto.

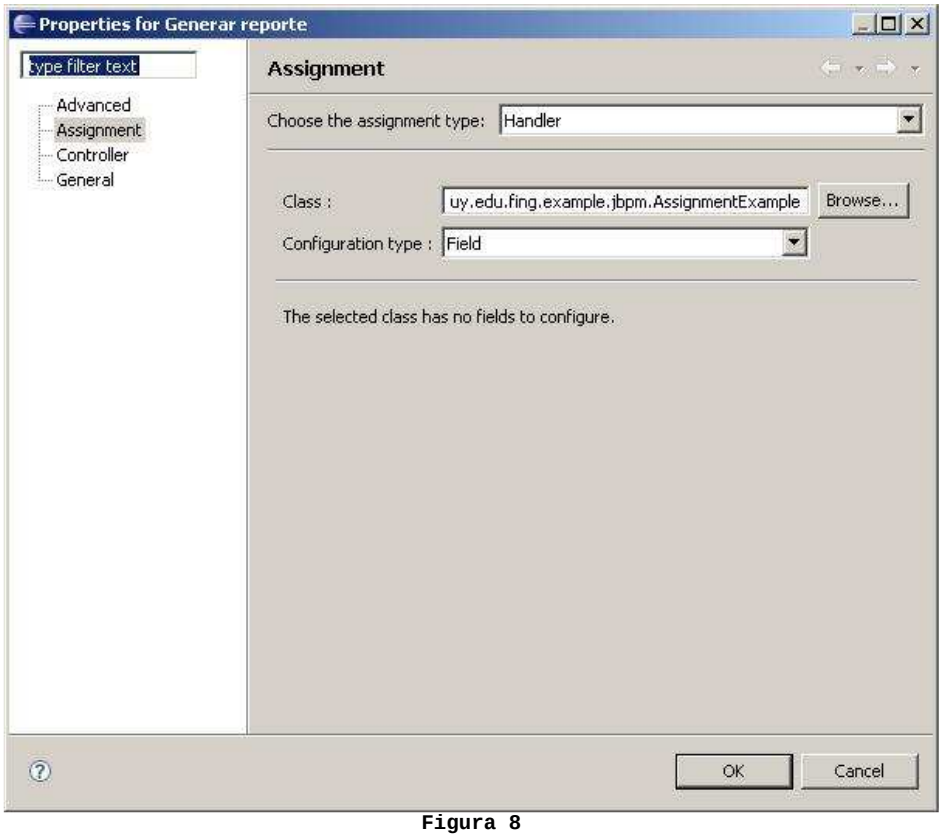
Properties for Generar reporte
type filter text
Advanced
Assignment
Controller
General
Assignment
Choose the assignment type:
Handler
Class :
uy.edu.fing.example.jbpm.AssignmentExample
Browse...
Configuration type :
Field
The selected class has no fields to configure.
OK
Cancel

**Figura 8**

## Asignación de tareas mediante uso de Pooled Actors

Los *Pooled Actors* son actores candidatos a ser asignados para realizar una tarea.

Si una tarea no se encuentra asignada, cualquier actor dentro de los Pooled Actors de la tarea puede reclamarla como suya y asignársela a sí mismo.

En este tutorial se asignarán dichos actores mediante el uso de un handlers de asignación, pero es posible asignarlos directamente en la base de datos. Para asignar los Pooled Actors mediante un Handler, se deben seguir los siguientes pasos:

1. Crear la clase *AssignmentPooledActorsExample* que implementa la interfaz *AssignmentHandler,* en el package *uy.edu.fing.example.jbpm*.
2. Agregar el siguiente código al método *assign*:

```
String[] actors = {"Juan", "Pablo", "Lucia"};
assignable.setPooledActors(actors);
```

3. Ir a las propiedades de la tarea *Realizar pruebas de aceptación* y seleccionar como handler a la clase *AssignmentPooledActorsExample*.

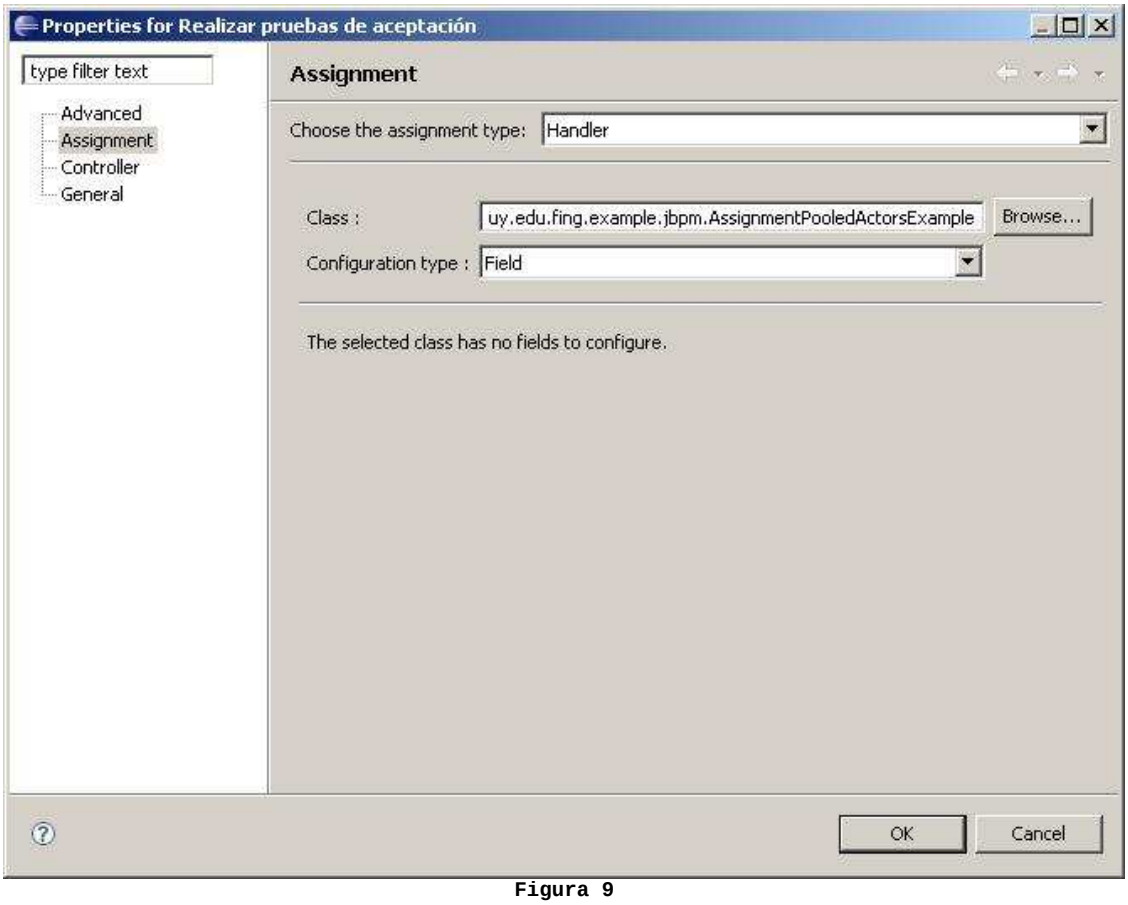


**Figura 9**

## Crear un driver para testear el proceso

1. Crear el método *testGetActorTaskList* y asignarle el siguiente código:

```
JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
try {
	GraphSession graphSession = jbpmContext.getGraphSession();
	//Crear una instancia de la última versión del proceso
	ProcessDefinition processDefinition =
		graphSession.findLatestProcessDefinition("proceso_avanzado");

	// Con la definición del proceso que obtenemos de la base
	// Creamos una ejecución del mismo.
	ProcessInstance processInstance = new ProcessInstance(processDefinition);

	//Moverse del estado incial
	processInstance.signal();

	while (!processInstance.hasEnded()){
	//Por cada tarea sin terminar del nodo imprimir sus actores
	Token token = processInstance.getRootToken();
	Collection tasks =
		processInstance.getTaskMgmtInstance().getUnfinishedTasks(token);
	for (Iterator iter = tasks.iterator(); iter.hasNext();) {
		TaskInstance task = (TaskInstance) iter.next();
		//Imprimir el actor responsable, si lo tiene
		if (task.getActorId()!=null)
System.out.println("\nEl actor encargado de la tarea "+task.getName().toUpperCase()+" es
"+ task.getActorId()+".\n");
		Set pooledActors = task.getPooledActors();

		//Imprimir los actores candidatos, si los tiene.
		if (pooledActors!=null){
System.out.println("\nLa tarea "+task.getName().toUpperCase()+" la pueden realizar los
siguientes actores:");
		for (Iterator iterator  = pooledActors.iterator();iterator.hasNext();) {
			PooledActor pActor = (PooledActor) iterator .next();
			System.out.println("\t-"+pActor.getActorId());
		}
		System.out.println("\n");
		}
		//Terminar la tarea.
		task.end();
	}
	}
} finally {
	// Cerrar el contexto jbpm
	jbpmContext.close();
}
```

## Testear el proceso

1. Hacer clic derecho en la clase *TestjBPM* y seleccionar *Run As->JUnit Test*
2. En la consola se despliega el log de la ejecución así como las también los encargados de realizar cada tarea.