Mappers & Performer assignments within Bonita

This document is a HowTo introducing mappers and performer assignment features in Bonita Workflow System. In the following lines we will present these powerful functionalities and we describe different steps for use them within Bonita.

1. - Mappers feature: automatic filling in of the bonita groups

1.1. - Introduction

Mappers feature gives the possibility to fill in automatically the bonita roles defined into the project model when the project is instantiated.

Three filling in methods are available (3 types of mappers) depending on the way to retrieve the users in the information system

- by getting groups/roles in an LDAP server (*ldap mapper*)
- by calling a java class to request a database (*custum mapper*)
- by getting the initiator of the project instance (*properties mapper*)

Like others definitions of process elements, the access to this functionality is performed throw the bonita API (See the ProjectSessionBean API). It's also accessible within the *graphEditor* application.

This function is particularly interesting for process instantiation usage of Bonita workflow System. The fill in of the groups happens at the first instantiation of the project model (for both the project model and the 1st instance). Then, it happens at each instance creation.

1.2. - Mappers types

1.2.1. - LDAP mapper:

This mapper uses your LDAP directory to retrieve users corresponding with a specific role defined in a Bonita Workflow project. Please refer to the documentation (Bonita LDAP configuration for JOnAS) to apply this type of mapper .

- LDAP mapper specificities:
  - The location of the LDAP groups depends on the attributes: *roleDN* and *roleNameAttribute* .
  - There is no mapping between roles/groups in the LDAP and roles in bonita database (same name for both bases).
  - The attribute name: *uid* has been used to realize the mapping between the actor identifier in the LDAP base and the userName in the bonita base.
  - If the goup does not exit an exception is thrown.

- Users found in the groups must have been deployed before usage of the mapper function. Otherwise an exception is thrown.

- The name of the mapper could be what you want

- Limitations of this version:

  - Groups cannot be recursive. Group's inclusions are ignored.

  - No checking that the distinguished names (dn) for the users found in the groups are compatible with the LDAP tree containing the users defined in the JOnAS LDAP realm configuration.

1.2.2. - Custom mapper

It lets the process developer to request its own user's storage base. When this type of mapper has been added, a call to a java class is performed. The name of this mapper is the name of the called java class (ex.: *hero.mapper.CustomSeachGroup*) located under *BONITA_HOME\src\resources\mappers\hero\mapper*. After retrieving users these must be added to the project instance and also added to the targeted role. The Bonita workflow engine loads and executes these classes at runtime, so, if you would add your custom mapper, please follow the next steps:

- Take a look at sample class above and implements your custom mapper logic in a new java file.

- Add your .java file into this directory and then launch "ant" task from your $BONITA_HOME directory.

- After that, you can start JOnAS application server.

1.2.3. - Properties mapper

At now, this type of mapper fills in the role with the user name of the creator of the instance (based on the authenticated user that initiates the instance). This mapper is very useful for administrative workflow processes in order to assign the role specified in the property to the user which has instantiated the process.

1.3. - Sample API to add mappers

```
.../....
  ProjectSessionHome projectSessionh=ProjectSessionUtil.getHome();
  ProjectSession pss=projectSessionh.create();

   String role1="Admintoto";
   pss.addRole(role1, "role added for activity 1");
   String role2="Admintiti";
   pss.addRole(role2, "role added for activity 2");

   // NODE 1
   pss.addNode("h1",Constants.Nd.AND_JOIN_NODE);
   pss.setNodeRole("h1",role1);
```

```
    // NODE 2
    pss.addNode("h2",Constants.Nd.AND_JOIN_NODE);
    pss.setNodeRole("h2",role2);

    // add MAPPERS
    pss.addRoleMapper(role1,"mapper1",Constants.Mapper.LDAP);
    pss.addRoleMapper(role2,"mapper2",Constants.Mapper.PROPERTIES);

    pss.instantiateProject(projectName);
  ..../....
```

2. - Performer assignments feature: to modify the standard assignment rules for activities

2.1. - Introduction

This new feature allows getting additional assignment rules than in the standard bonita model.
In the std model (oriented cooperative workflow), all the users defined into the group associated to the activity can see and can execute (***toDo List***) this one.
By adding the new functionality, we can:

- **assign the activity to a user of a group** by calling a java class in charge to do the user selection into the user group (*callback performer assignment*)
- **assign dynamically the activity to a user** by using an *activity property* (*properties performer assignment*)

When this functionality is added, the user is notified (mail notification) that the activity is ready to be started.

The users of the groups (role in Bonita) associated to the activity can see the activity but cannot start and terminate it.

This functionality is accessible within the Bonita API (see ProjectSessionBean API) and inside the Bonita *graphEditor* application.

Furthermore, we can assign an activity to the **initiator of the instance.** It needs only the use of a *properties mapper* (as described above).

2.2. - Description of these performer assignments

2.2.1. - callback performer assignment

It lets the process developer writing a request with its own algorithm of user selection. When this type of callback **performer assignment** has been added, a call to a java class is performed.

The name of this callback performer assignment is the name of the called java class (ex.: *hero.performerAssign.CallbackSelectActors*) located under *BONITA_HOME\src\resources\performerAssigns\hero\performerAssign*. As mappers, your callbacks are loaded and executed by Bonita workflow engine. If you would add your own callback, please follow the next steps:

- Take a look at sample class above and implements your performer assignment logic in a new java file.

- Add your .java file into this directory and then launch "ant" task from your $BONITA_HOME directory.

- Start JOnAS application server.

2.2.2. - Properties performer assignment

It allows the process developer to provide at the **properties performer assignment** creation the activity property that is used by the workflow engine to assign the activity. This activity property has to be defined either into a previously sequenced activity with the property propagation or into the targeted activity to be assign.

2.3.- Sample API to add mappers

```
..../....
   // NODE 1
   pss.addNode("h1",Constants.Nd.AND_JOIN_NODE);
   pss.setNodeRole("h1",role1);

   // NODE 2
   pss.addNode("h2",Constants.Nd.AND_JOIN_NODE);
   pss.setNodeRole("h2",role2);

   // NODE 3
   pss.addNode("h3",Constants.Nd.AND_JOIN_NODE);
   pss.setNodeRole("h3",role3);

.../....

 // activity property
 pss.setNodeProperty("h3","acteurH3","gaillarr");
..../....

 // PERFORMER ASSIGN
  pss.addNodePerformerAssign("h2",
"hero.performerAssign.CallbackSelectActors" ,
Constants.Performer.CALLBACK,"");
pss.addNodePerformerAssign("h3",
"hero.performerAssign.PropertySelectActors" ,
Constants.Performer.PROPERTIES ,"acteurH3");
```