TP5 : Persistance

Ce TP permet introduit la mise en place de la persistance dans notre processus précédent. Nous utilisons une base *Postgres* pour stocker les informations relatives à l'exécution du processus

Etapes

1. Création de projet

Utiliser l'assistant pour créer un projet jBPM nommé TP5

2. Modification du processus du précédent TP

Dorénavant, nous voulons associer chaque demande de crédit à une instance de processus. Nous modifions le processus du TP précédent comme suit

- Supprimer la boucle sur le nœud de type RuleTask
- Implémenter un état en attente de signal sur la branche « PetitCredit »

2. Création d'une base postgres

Pour stocker les états des processus et leurs historiques, nous allons utiliser une autre base que la base H2 de démonstration.

Créer une base de données *jbpm*

Créer le fichier persistence.xml correspondant utilisant JTA

Un exemple est donné ci-dessous

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Persistence deployment descriptor for dev profile -->
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
             version="1.0">
<persistence-unit name="tp5db" transaction-type="JTA">
  <provider>org.hibernate.ejb.HibernatePersistence</provider>
  <jta-data-source>jdbc/TP5</jta-data-source>
  <class>org.drools.persistence.info.SessionInfo</class>
  <class>org.jbpm.persistence.processinstance.ProcessInstanceInfo</class>
  <class>org.drools.persistence.info.WorkItemInfo</class>
  <class>org.jbpm.process.audit.ProcessInstanceLog</class>
  <class>org.jbpm.process.audit.NodeInstanceLog</class>
  <class>org.jbpm.process.audit.VariableInstanceLog</class>
 <properties>
    <property name="hibernate.connection.driver_class"
value="org.postgresql.xa.PGXADataSource"/>
```

```
    <property name="hibernate.connection.url"
value="jdbc:postgresql:jbpm"/>
    <property name="hibernate.connection.username" value="postgres"/>
    <property name="hibernate.connection.password" value="postgres"/>
    <property name="hibernate.dialect"
value="org.hibernate.dialect.PostgreSQLDialect"/>
    <property name="hibernate.max_fetch_depth" value="3"/>
    <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
    <property name="hibernate.show_sql" value="false"/>

      <property name="transaction.factory_class"
value="org.hibernate.transaction.JTATransactionFactory"/>
    <property name="hibernate.transaction.manager_lookup_class"
value="org.hibernate.transaction.BTMTransactionManagerLookup"/>
</properties></persistence-unit>
</persistence>
```

Récupérer le driver *jdbc* de Postgres et l'inclure dans le classpath.

3. Intégration du gestionnaire de transaction Bitronix avec JTATransactionManager de Drools

L'intégration avec Bitronix nécessite la création d'un pool de connexion via Bitronix et l'enregistrement dans son serveur JNDI.
Voir méthode *_initDS()* dans TP5_1test
Les lookup JNDI (effectués par *JTATransactionManager* de drools) doivent s'adresser à Bitronix. Pour cela copier le fichier jndi.properties fourni à la racine du classpath.

Exécutez la classe *TP5_1Test* en mode debug et observez les créations de tables dans la base de données

4. Compléter la classe de test

La classe de test va interagir 3 fois avec la base :

1. Création de la session JPA, Association d'un JPAWorkingMemoryDBLogger et démarrage du processus avec une demande de petit crédit
2. Rechargement de la précédente session avec un nouvel environnement, envoi d'un signal permettant de finir la demande de petit crédit précédente.
   Démarrage d'une demande de gros crédit.
3. Rechargement de la précédente session et vérification

Exécuter la classe de test et observer le bon déroulement du processus.