Ateliers

Formation jBPM7

Pré-requis :
Poste développeur avec accès réseau Internet libre
Linux, Windows 10, MacOS
Pré-installation de :

- JDK8
- Ant
- Git

Solutions :

- `https://github.com/dthibau/jbpm-solutions.git`

Atelier 1: Mise en place l'IDE

Différentes façons d'installer IDE, les différents plugins les différents modeleur BPMN.

Le plus simple (permettant une compatibilité avec les versions d'Eclipse) est d'utiliser la distribution *jbpm-full-installer* qui :

- Télécharge le serveur applicatif WildFly
- Configure et déploie un KieServer
- Configure et déploie Business Central
- Configure et déploie l'application exemple case management
- Télécharge Eclipse
- Installe le plugin Eclipse Drools et jBPM
- Installe le modeleur Eclipse BPMN 2.0

Etapes

1. Installation

Dézipper la distribution de l'installeur: *jbpm-installer-full-<version>.Final.zip*

Dans le répertoire <JBPM_INSTALLER_HOME> :
`ant install.demo.eclipse`
Prendre un café !

Lorsque l'installation est terminée :
Démarrer l'IDE avec
`ant start.demo.eclipse`

2. Premier projet

Démarrer l'assistant pour créer un projet jBPM et choisir :

- Créer un projet avec les exemples
- Donner un nom
- Choisir Maven
  - *GroupId* : ***org.formation***
  - *artifcatId* : ***tp1***
- Sélectionner également :
  - Ajouter un processus simple
  - Ajouter un test unitaire

Observer dans le projet:

- Ses dépendances
- *kmodule.xml*
- Le processus
- Le cas de Test

Exécuter le cas de test et la classe principale

Atelier 2 : Test d’un processus et API ksession

Cet atelier implémente un test unitaire pour un processus.
La base de connaissance est chargée à partir du classpath.

Etapes

1. Création de projet

Nouveau projet Maven sans source

- *GroupId* : ***org.formation***
- *artifcatId* : ***tp2***

2. Test du processus

Récupérer les sources fournis : *Lab2.bpmn* et *ProcessTest.java* et les copier dans le projet

Compléter les méthodes `testClasspathProcess et _doTest` afin de tester l’exécution du processus.
Inspirer vous de l’exemple du TP précédent

Visualiser le fichier de traces produit et l’ouvrir dans l’*Audit View*

Atelier 3 : Modélisation

Cet atelier permet de se familiariser avec les nœuds : *Script*, *Timer*, *Subprocess*, les gateways *diverge* et *converge*.

Etapes

1. Création de projet Project

Projet Maven sans code source

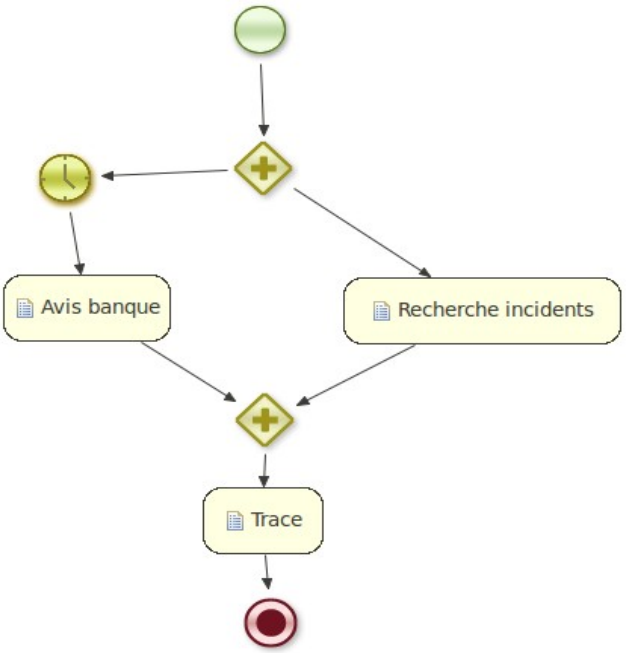
- *GroupId* : ***org.formation***
- *artifcatId* : ***tp3***

2. Branches parallèles

Créer une définition de processus dans le dossier ressources :
org/formation/tp3/subprocess.bpmn

Ce processus modélise une demande de prêt bancaire .
Il est composé de 2 branches parallèles :

- La première branche consistera à terme en une tâche utilisateur (un banquier qui étudie la demande de crédit).
  Pour le moment, nous allons modéliser cette branche avec juste un nœud ***timer*** et un nœud ***script***.
- La deuxième branche modélisera une activité automatique censée vérifier dans l'historique du client les éventuels incidents bancaires.
  Pour l'instant, nous allons modéliser cette branche par un simple nœud de ***script***.



Exécuter le cas de test fournis : *org.formation.lab3.test.Lab3_Test1.java*

2. Sous-process et traitement des erreurs

Créer un autre processus : *org/formation/tp3/Main.bpmn*

Le processus principal démarre le sous-processus précédent

Mettre à jour le sous-processus afin qu'il lance systématiquement un événement erreur à la fin de son processus.
Cette erreur devra déclencher un gestionnaire d'erreur dans le processus principal

Tester l'exécution du processus avec la classe fournie :
org.formation.lab3.test.TP3_Test2.java

Atelier 4 : Données associées au processus

Cet atelier présente différents cas d'utilisation des données associées aux processus.

Etapes

1. Projet

Le projet précédent peut être réutilisé

2. Branching conditionnel

Nous modifions le processus précédent. Selon le montant d'une demande (demande de crédit), différentes branches doivent être exécutées :

- Si le montant de la demande est inférieur à 1000, le crédit est immédiatement accepté et le processus atteint un état final (non définitif)
- Si le montant de la demande est supérieur ou égal à 1 000, le sous-processus est lancé.

Récupérez la classe Java fournie modélisant une requête.
Modélisez cette branche conditionnelle et implémentez une classe de test pour vous assurer de la bonne exécution du processus

2. Échange de données avec le sous-processus

Dans le cas d'une demande de crédit supérieure à 1000, le sous-processus est lancé.
Celui-ci va gérer 3 variables :

- ***note*** : Représente une évaluation donnée par un agent bancaire
- ***incidents*** : Représente les incidents bancaires du client faisant sa demande
- ***result*** : booléen donnant le résultat de l'enquête. *true* si la note est supérieure aux incidents

Dans le processus principal, une variable booléenne ***result*** représente le résultat de la demande. Si cette variable est vraie, le crédit est accordé sinon il est rejeté.

- Dans le cas d'un petit crédit, la variable *result* est fixée par le processus principal.
- Dans le cas d'un grand crédit, la variable *result* reçoit la valeur de la variable de sous-processus de résultat.

Déclarez les variables nécessaires dans les différents processus et effectuez le mappage approprié entre le processus principal et le sous-processus.

3. Ecrire une classe de test

Mettre en place une classe de test qui contient 2 méthodes qui valident les 2 chemins du processus.

Atelier 5 : Persistance

Cet atelier met en œuvre la persistance de notre processus précédent.
Nous utilisons le support de *JbpmUnitTestCase* pour configurer la persistance et la source de données JTA. Une base de données H2 par défaut est utilisée

Etapes

1. Dépendances Maven et persistence.xml

Ajouter la dépendance suivante :

```
org.jbpm:jbpm-persistence-jpa :${runtime.version}
```

Récupérez le fichier de configuration de l'unité de persistance et placez-le dans *src/main/resources/META-INF*
Il est configuré pour afficher les instructions SQL produites par Hibernate.

2. Mise à jour du sous-processus
Afin de voir la persistance en action, nous insérons un état d'attente dans le processus.
Mettre à jour le sous-processus et modifier la tâche de script « Donner une note » par un nœud en attente d'événement

2. Completer le cas de test fourni

Vous devez :

- Initialiser la persistance grâce au bon constructeur
- Récupérer le service de journal afin d'interroger la base de données.
- Implémenter la méthode ***printProcess*** qui affiche les états des processus dans la session
- Interagir avec le processus

Regardez les instructions SQL produites par l'exécution du processus.

Atelier 6 : Workflow humain

Dans cet atelier , nous modélisons certaines tâches humaines et utilisons l'implémentation de *TaskService* fourni par jbpm

Préparation du projet pour le workflow humain

Ajouter les dépendances suivantes : :

```
<dependency>
  <groupId>org.jbpm</groupId>
  <artifactId>jbpm-human-task-audit</artifactId>
  <version>${runtime.version}</version>
</dependency>
```

Récupérer le fichier ***persistence.xml*** qui définit les requêtes pour le service de tâche
Récupérez ***usergroups.properties*** et mettez-le dans *src/main/resources*, il définit les utilisateur et les rôles

Tests du processus

Récupérer la classe de test fournie compléter la méthode *testGrosDebit()*

Atelier 7 : Service de tâche

Cet atelier permet de mettre en œuvre un nœud d'intégration d'un service spécifique.
Le nœud chargé de trouver le nombre d'incidents d'un client donné est remplacé par un nœud de service. (il était implémenté jusqu'à présent par un nœud de script),

Etapes

1. Implémentation d'un WorkItemHandler

Implémentez un ***WorkItemHandler*** qui récupère un paramètre d'entrée représentant le client.

Faire un appel asynchrone à une méthode en prenant un peu de temps et en calculant le nombre d'incidents.

À la fin de cette méthode, terminez la tâche de service

2. Mise à jour du processus
Modifier le processus et remplacer le nœud de script calculant les incidents par le nœud de service effectuer la correspondance des entrées/sorties

3. Classe de Test
Modifier la classe de test pour enregistrer le gestionnaire précédent afin de gérer la tâche de service *"Calcul d'incident"*.
Tester