

Université Mohammed V de Rabat -  
Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes

## RAPPORT DE PROJET IDM

*3<sup>ème</sup> année*

Option : Génie Logiciel

---

### **Transformer un backlog (User story) en BPMN.**

---

Réalisé par :  
Moufid Ibtissam  
Nasrallah Meriem  
SABOUR Ilham  
Wardi Hafssa

*Sous la direction de :*  
Pr. El Hamlaoui Mahmoud

# Table des figures

1	Schéma de la tranformation M2M . . . . .	3
2	EMF logo . . . . .	3
3	Sirius logo . . . . .	4
4	Xtext logo . . . . .	4
5	Henshin logo . . . . .	4
6	User story DSL . . . . .	5
7	User story EMF . . . . .	6
8	BPMN Diagram . . . . .	7
9	BPMN EMF . . . . .	8
10	Henshin transformation rules 1 . . . . .	9
11	Henshin transformation rules 2 . . . . .	10
12	Henshin transformation units . . . . .	11
13	generate XMI from US code . . . . .	12
14	us test model . . . . .	13
15	user story en entrée . . . . .	13
16	BPMN généré . . . . .	14

Le présent projet se situe dans le cadre de l'ingénierie dirigée par les modèles IDM. Il consiste à réaliser un système qui prend en entrée un ensemble de user stories et les transforme en un modèle BPMN.

## Quelques définitions

Ci-dessous quelques définitions qui s'avèrent utiles pour se situer dans le projet.

1. **Ingénierie Dirigée par les Modèles IDM : *Model Driven Engineering*** C'est un ensemble de pratiques fondées sur le concept de modèle de domaine. Ces pratiques ont pour but d'automatiser la production, la maintenance ou l'utilisation de systèmes logiciels. L'objectif de cette approche est de concentrer les efforts sur le domaine d'application du logiciel plutôt que sur son implémentation.
2. **Modèle de Domaine : *Domain Model ou MetaModel*** C'est le modèle conceptuel d'un domaine. Il comprend tant le comportement que les données. En UML, il est représenté par un diagramme de classe.
3. **Langage dédié : *Domain Specific Language DSL*** Il fournit une notation orientée à un domaine d'application. Il est basé sur les concepts et les fonctionnalités du domaine en question. En tant que tel, un langage dédié est un moyen efficace de décrire et de générer des programmes dans un domaine spécifique.
4. **Transformation Model to Model M2M** : Il s'agit d'une démarche d'analyse syntaxique et sémantique d'un modèle donnée afin de générer un nouveau modèle défini à son tour par un méta-modèle.

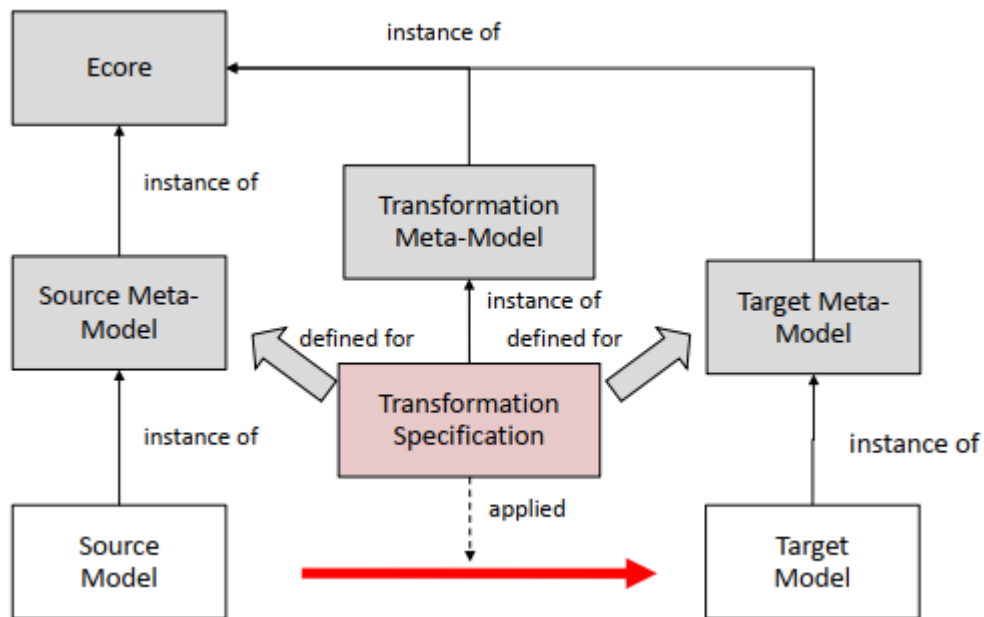


FIGURE 1 – Schéma de la tranformation M2M

## Technologies utilisés

### Eclipse Modeling Framework



FIGURE 2 – EMF logo

C' est un framework de modélisation, une infrastructure de génération de code et des applications basées sur des modèles de données structurées. Partant d'une spécification décrite généralement sous la forme d'un modèle en XMI, EMF fournit des outils permettant de produire des classes Java représentant le modèle avec un ensemble de classes pour adapter les éléments du modèle afin de pouvoir les visualiser, les éditer avec un système de commandes et les manipuler dans un éditeur.

## Sirius



FIGURE 3 – Sirius logo

Sirius est un projet Open Source de la Fondation Eclipse. Cette technologie permet de concevoir un atelier de modélisation graphique sur-mesure en s'appuyant sur les technologies Eclipse Modeling, en particulier EMF (Eclipse Modeling Framework) et GMF (Graphical Modeling Framework). L'atelier de modélisation créé est composé d'un ensemble d'éditeurs Eclipse (diagrammes, tables et arbres) qui permettent aux utilisateurs de créer, éditer et visualiser des modèles EMF.

## Xtext



FIGURE 4 – Xtext logo

Xtext est un framework pour le développement de langages de programmation et de DSL. Il s'appuie sur une grammaire générée ANTLR ainsi que sur le framework de modélisation EMF. Xtext couvre tous les aspects d'un IDE moderne : parseur, compilateur, interpréteur et intégration complète dans l'environnement de développement Eclipse. À ce titre, il permet de couvrir les aspects et les fonctionnalités principales d'un IDE classique.

## Henshin



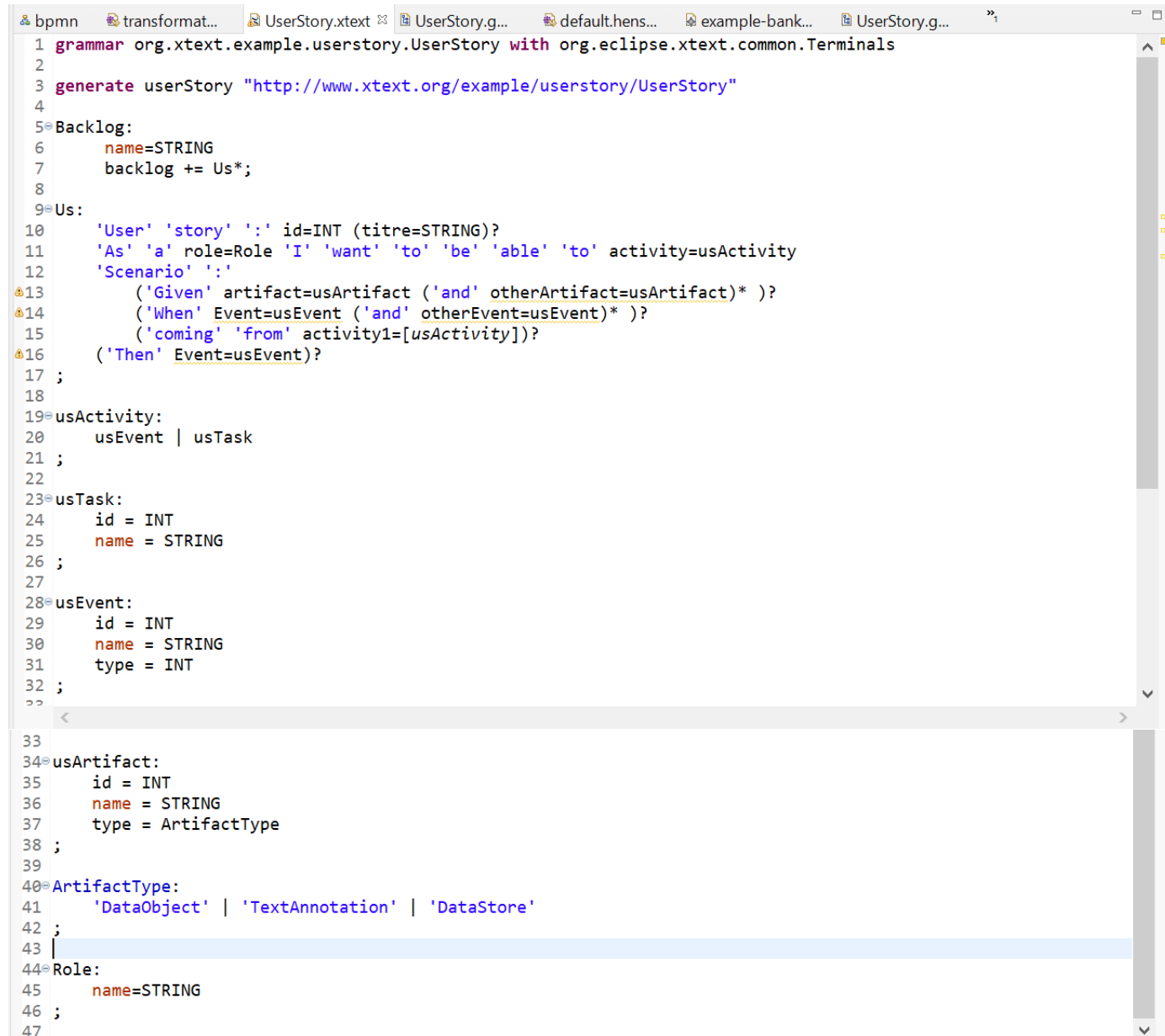
FIGURE 5 – Henshin logo

Le projet Henshin fournit un langage de transformation de modèle de pointe pour le cadre de modélisation Eclipse. Henshin supporte à la fois les transformations directes d'instances de modèle unique EMF (transformations endogènes), et la traduction d'instances de modèle source dans un langage cible (transformations exogènes).

# Conception et réalisation

## Métamodèle source : User story

Pour créer le métamodèle du user story on utilise le générateur textuel xtext. Premièrement on définit le dsl comme suit :



```
1 grammar org.xtext.example.userstory.UserStory with org.eclipse.xtext.common.Terminals
2
3 generate userStory "http://www.xtext.org/example/userstory/UserStory"
4
5 =Backlog:
6     name=STRING
7     backlog += Us*;
8
9 =Us:
10    'User' 'story' ':' id=INT (titre=STRING)?
11    'As' 'a' role=Role 'I' 'want' 'to' 'be' 'able' 'to' activity=usActivity
12    'Scenario' ':'
13    ('Given' artifact=usArtifact ('and' otherArtifact=usArtifact)* )?
14    ('When' Event=usEvent ('and' otherEvent=usEvent)* )?
15    ('coming' 'from' activity1=[usActivity])?
16    ('Then' Event=usEvent)?
17 ;
18
19 =usActivity:
20     usEvent | usTask
21 ;
22
23 =usTask:
24     id = INT
25     name = STRING
26 ;
27
28 =usEvent:
29     id = INT
30     name = STRING
31     type = INT
32 ;
33
34 =usArtifact:
35     id = INT
36     name = STRING
37     type = ArtifactType
38 ;
39
40 =ArtifactType:
41     'DataObject' | 'TextAnnotation' | 'DataStore'
42 ;
43
44 =Role:
45     name=STRING
46 ;
47
```

FIGURE 6 – User story DSL

Ensuite on génère le métamodèle ecore, représenté comme suit :

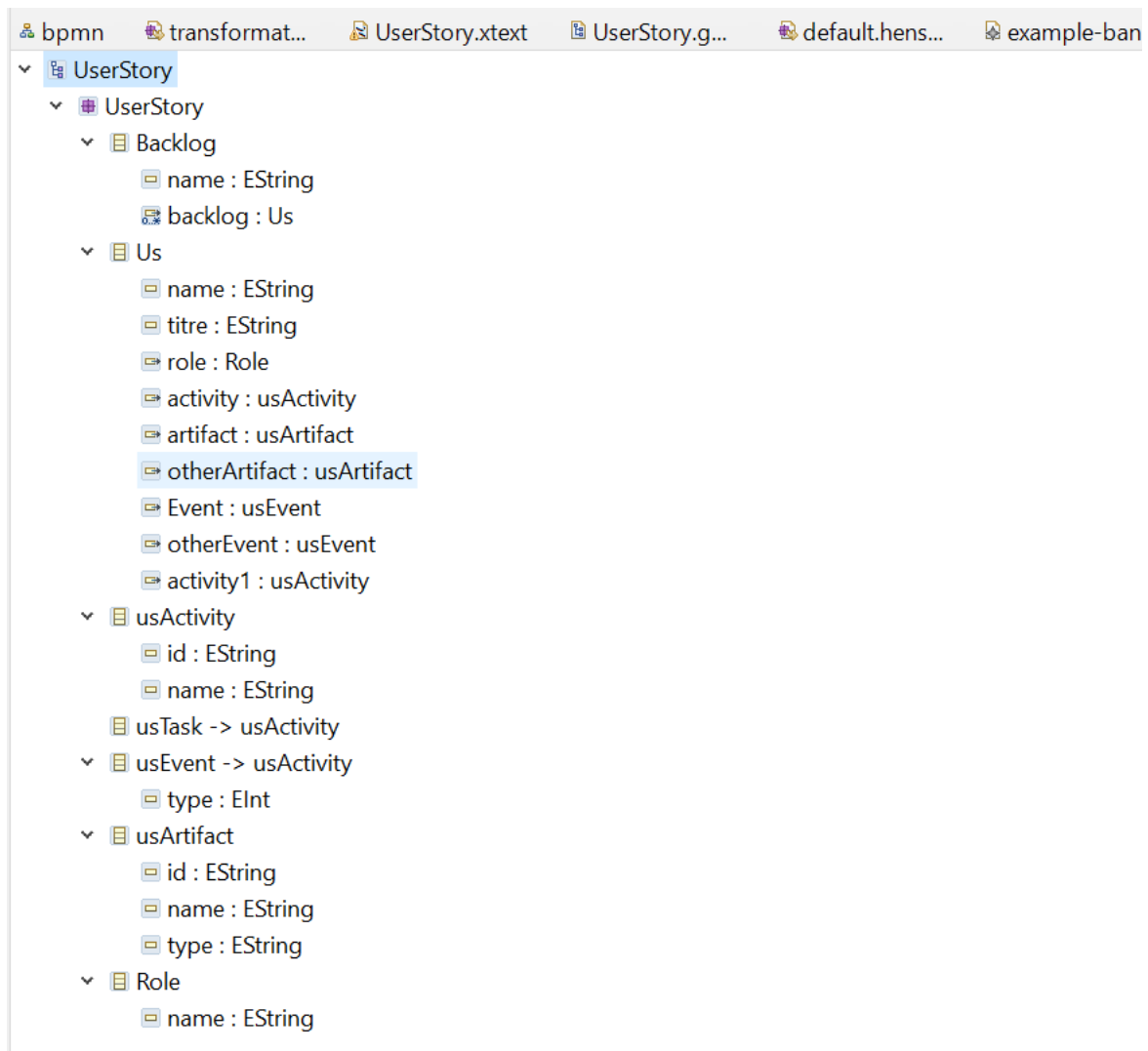


FIGURE 7 – User story EMF

## Métamodèle destination : BPMN

On utilise le générateur graphique Sirius pour générer le métamodèle du BPMN figurant ci-dessous.

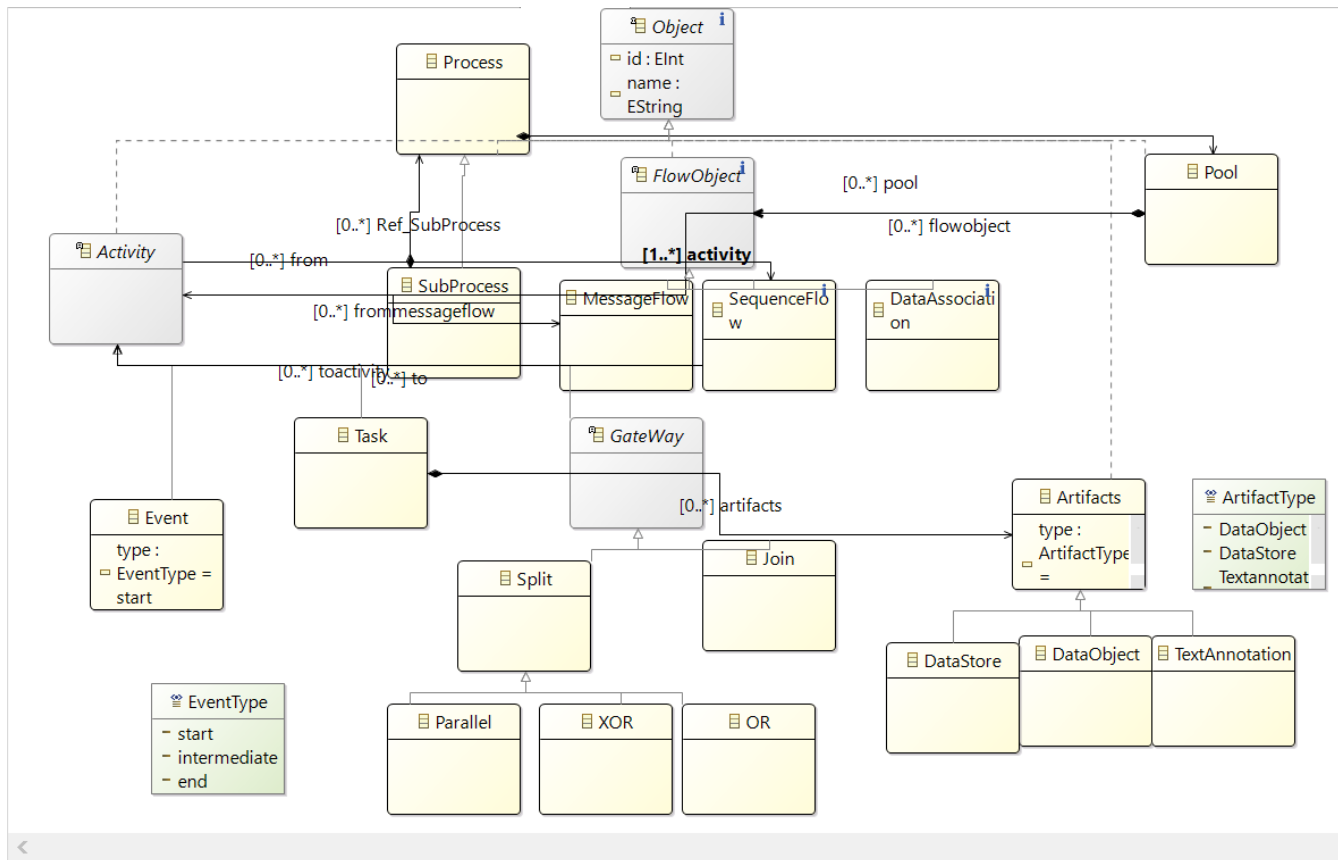


FIGURE 8 – BPMN Diagram



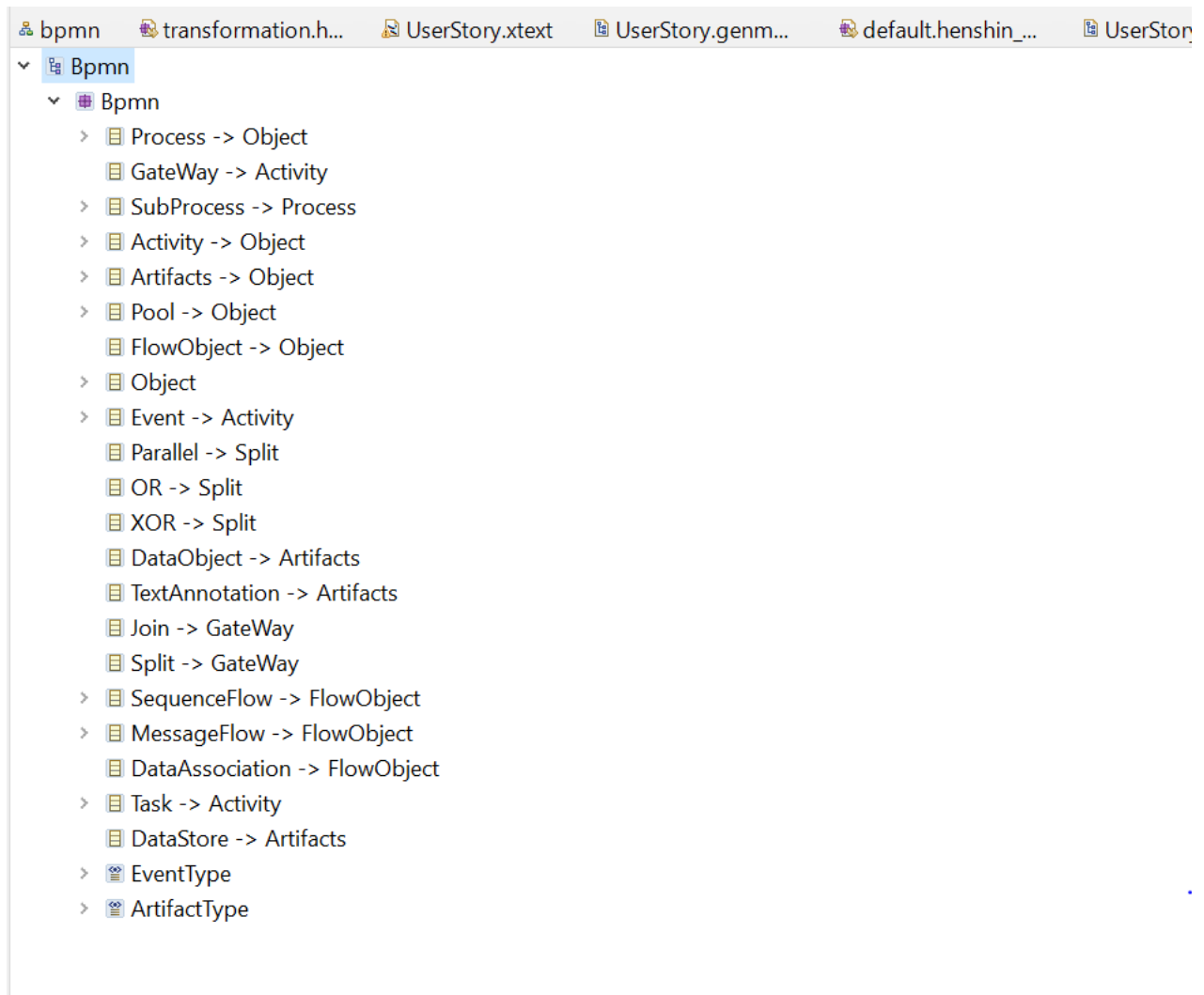


FIGURE 9 – BPMN EMF

## Transformation : Henshin

On définit les règles de transformation associées à chaque objet ou concept du méta-modèle. Les règles de transformation adaptées pour notre modèle sont les suivantes :

1. **CreateProcess** : instancie le process traité par le modèle. Il est constitué des autres objets
2. **CreatePool** : crée un nouveau pool dans le modèle de destination correspondant au rôle issu de la user story et contient plusieurs activités, séquences et artifacts
3. **CreateTask** : crée une nouvelle tâche bpmn (task) correspondant à une tâche user story (usTask)
4. **CreateEvent** : crée un événement bpm correspondant à un événement user story. On définit son type : start, intermediate ou end

5. **CreateSequenceFlow CreateMessageFlow** : crée les transitions entre les activités appartenent au même pool ou entre pool
6. **CreateArtifact** : crée un artifact

En plus de ces règles on ajoute des unités de transformation voire cycliques et séquentielle.

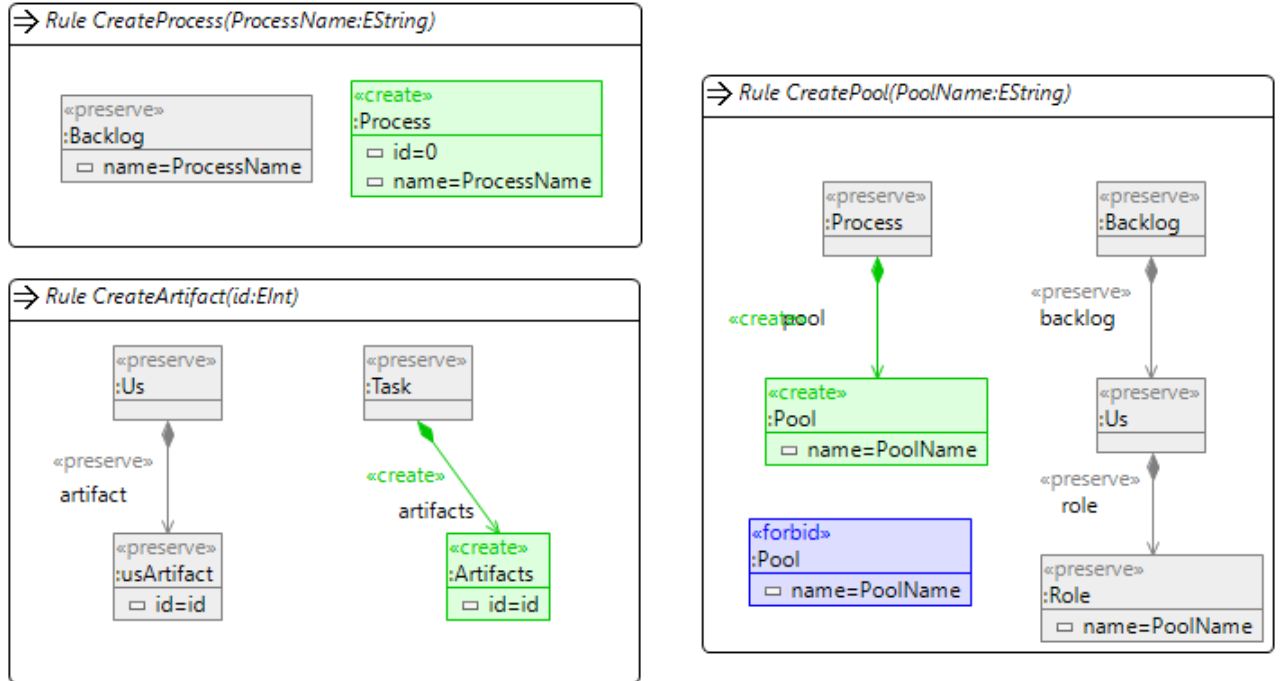


FIGURE 10 – Henshin transformation rules 1

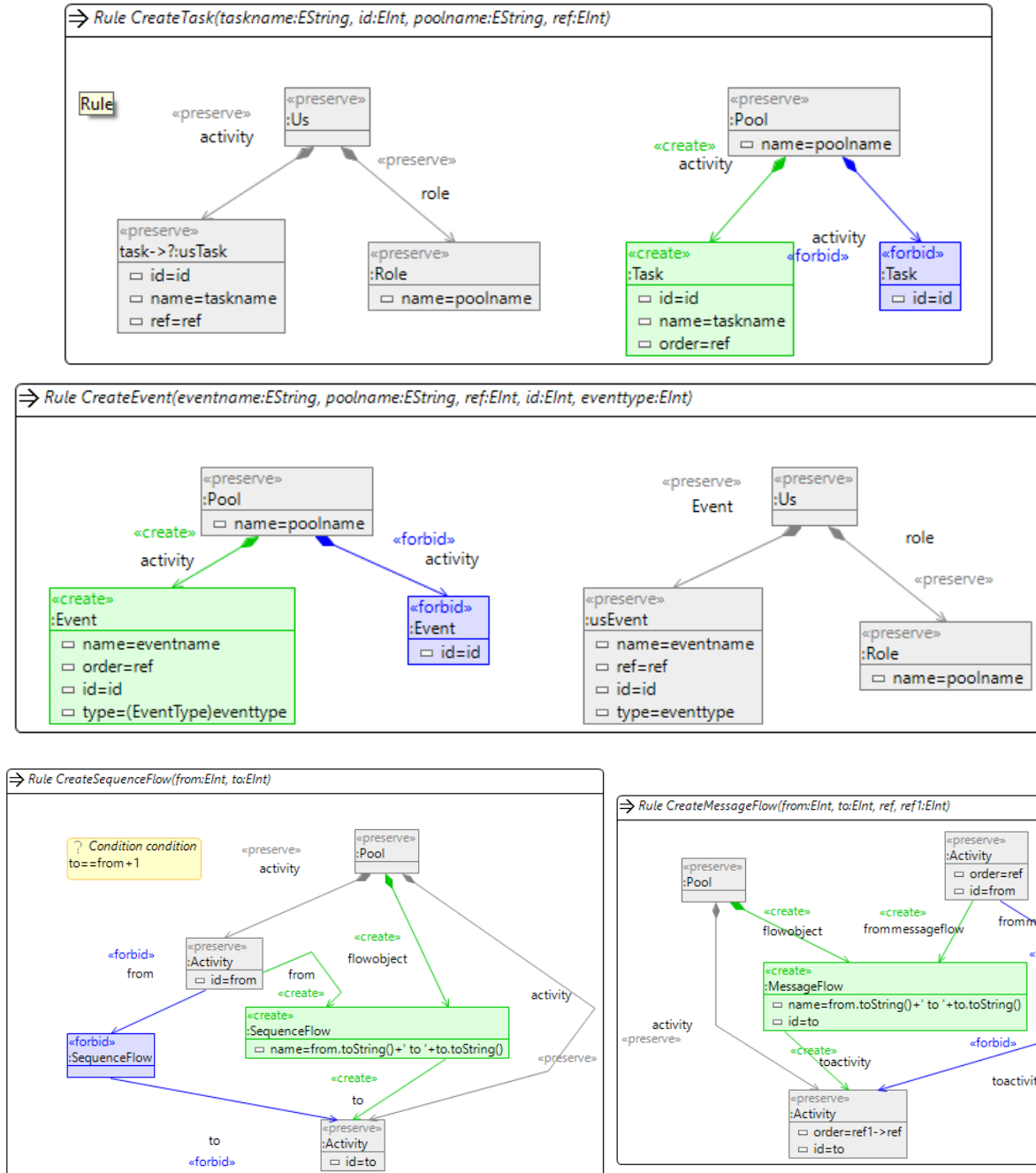


FIGURE 11 – Henshin transformation rules 2

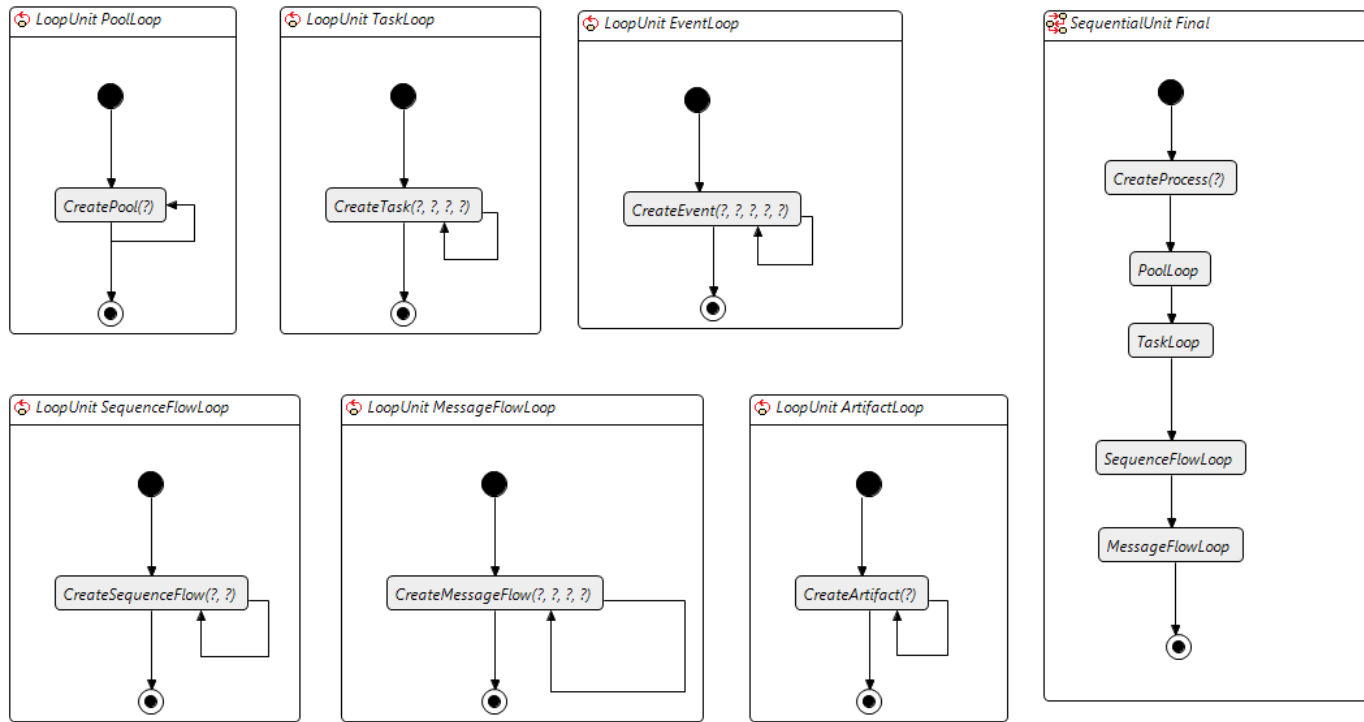


FIGURE 12 – Henshin transformation units

# Instanciation du modèle

On commence d'abord par créer un fichier 'test.us' qui définit un modèle de user story. Ensuite, afin de générer une instance dynamique 'result.XMI' à partir de ce fichier on crée une classe java qui permet de faire cette transformation. Elle est comme suit :

```
1
2 import java.io.IOException;
3 import org.eclipse.emf.common.util.URI;
4 import org.eclipse.emf.ecore.resource.Resource;
5 import org.eclipse.emf.ecore.resource.ResourceSet;
6 import org.eclipse.emf.ecore.util.EcoreUtil;
7 import org.xtext.example.userstory.UserStoryStandaloneSetup;
8
9 import com.google.inject.Injector;
10
11 public class Main {
12
13     public static void main(String[] args) throws IOException {
14         Injector injector = new UserStoryStandaloneSetup().createInjectorAndDoEMFRegistration();
15         ResourceSet resourceSet = injector.getInstance(ResourceSet.class);
16         Resource resource = resourceSet.getResource(URI.createURI("./model/test.us"), true);
17         resource.load(null);
18         EcoreUtil.resolveAll(resourceSet);
19         Resource xmiResource = resourceSet.createResource(URI.createURI("./model/result.xmi"));
20         xmiResource.getContents().add(resource.getContents().get(0));
21         xmiResource.save(null);
22     }
23 }
24
25
```

FIGURE 13 – generate XMI from US code

Le modèle 'test.us' présente un backlog comportant deux user stories, chacune avec un rôle et une activité

```

test.us
'back'
User story : 1 As a 'customer' I want to be able to 1 'start' 0
User story : 2 As a 'customer' I want to be able to 2 'selectTravel'
User story : 3 As a 'customer' I want to be able to 3 'confirmInterest'
User story : 4 As a 'customer' I want to be able to 4 'end' 2
User story : 5 As a 'organization' I want to be able to 1 'start' Scenario: coming from selectTravel
User story : 6 As a 'organization' I want to be able to 2 'indicate price'
User story : 7 As a 'organization' I want to be able to 3 'end' 2

```

FIGURE 14 – us test model

L'instance dynamique générée est :

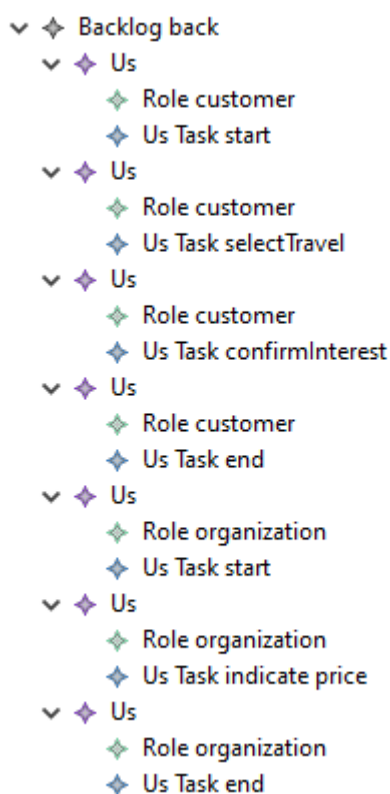


FIGURE 15 – user story en entrée

Après application de la transformation on obtient l'instance suivante :

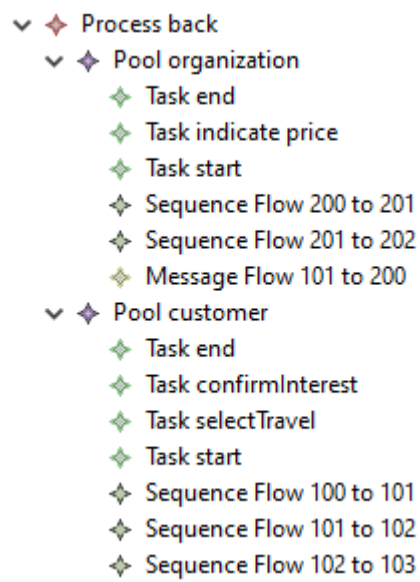


FIGURE 16 – BPMN généré