Mini-projet IDM

Nesrine Harbaoui
Mohamed El Yessefi
Yahya Younes

Département Sciences du Numérique - Parcours Réseaux
2022-2023

Table des matières

Table des figures

1 Introduction

Le présent rapport porte sur le mini-projet en Ingénierie Dirigée par les Modèles (IDM) réalisé dans le cadre de notre formation. L'objectif de ce projet était de mettre en pratique les concepts fondamentaux de l'IDM en utilisant différents outils tels qu'Ecore, OCL, Xtext, Sirius, Acceleo et Tina pour modéliser, représenter et transformer des modèles.

Dans ce projet, nous avons commencé par visualiser le méta-modèle Ecore et définir des contraintes OCL pour le modèle SimplePDL. Ensuite, nous avons utilisé l'outil Xtext pour créer une syntaxe textuelle du modèle SimplePDL et l'outil Sirius pour créer une syntaxe graphique pour ce même modèle.

Par la suite, nous avons écrit du code Java pour transformer le modèle SimplePDL en un réseau de Petri et avons utilisé Acceleo pour générer un texte interprétable en TINA pour le modèle de réseau de Petri.

Ce rapport présentera donc une description détaillée de chaque étape du projet, y compris les méthodes, les outils et les résultats obtenus.

2 SimplePDL

2.1 Métamodélisation

SimplePDL est un langage de description de processus de développement logiciel. Il permet de modéliser un processus qui est composé d'un ensemble d'activités définies par une WorkDefinition, et d'une relation entre ces activités représentée par WorkSequence, qui est de type WorkSequenceType. Des remarques sur les activités peuvent être fournies grâce à l'utilisation de Guidance, tandis que les activités peuvent être réalisées à l'aide de Resource, qui peut être partagé par plusieurs activités. Le méta-modèle décrivant SimplePDL est présenté dans le fichier SimplePDL.ecore, dont le diagramme est illustré dans la figure 1.

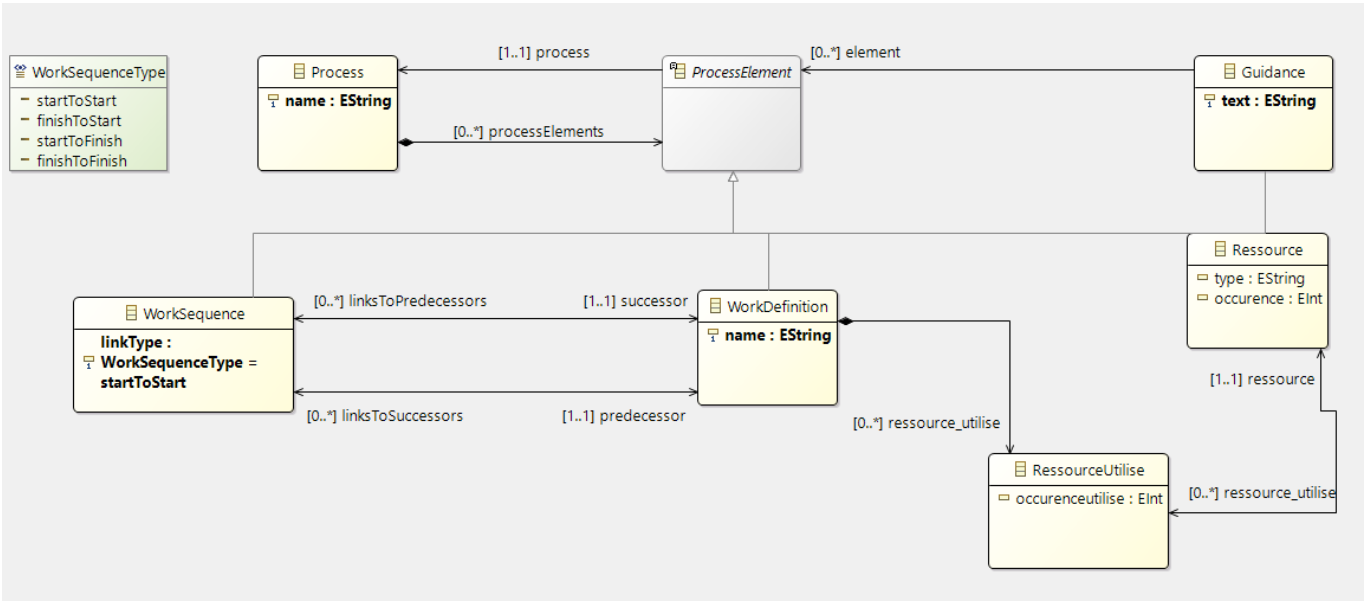

Figure 1 – Métamodéle SimplePDL

2.2 Contraintes OCL

Il est recommandé de renforcer le métamodèle décrit dans simpleP DL.ecore avec davantage de contraintes OCL pour créer des modèles plus robustes et plus utiles.Le fichier SimpleP DL.ocl contient des contraintes OCL qui permettent d'assurer des contraintes qui ne peuvent pas être assurées par le métamodèle lui-même.

2.3 Exemples

Nous avons réussi à tracer un editeur Nous avons réussi à engendré un editeur arborescent de modéle, le lancer et l'utiliser en saisissant un modèle de processus. Le fichier My.SimplePDL est dans la fenêtre principale. Nous avons crée 4 workDefinition : Conception, Developpement, RedactionDocs et RedeactionTests, 5 WorkSequence : 2 de type startToStart, 2 de type finishToFinish et 1 de type finishToStart et 5 Ressource : concepteur, developpeur, redacteur, testeur et machine. L'editeur arborescent que nous avons crée est illustré dans la figure 2.

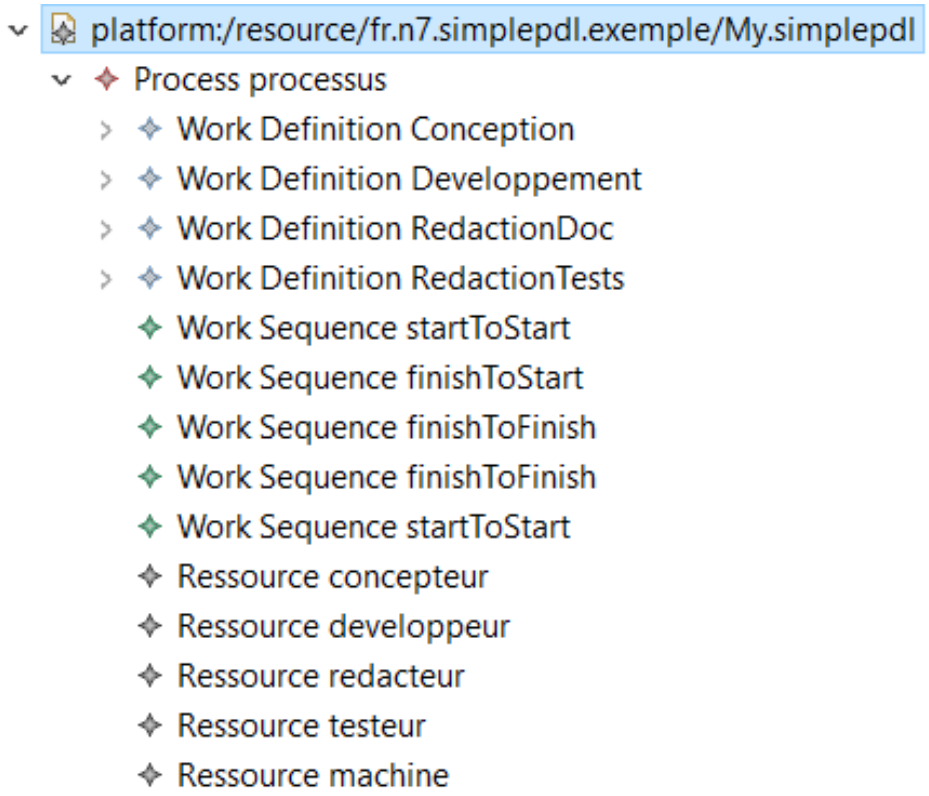

FIGURE 2 – Editeur arborescent du modéle SimplePDL

2.4 L'outil Xtext

L'éditeur arborescent n'est pas suffisant pour bien visualiser le modèle. Il est necessaire de définir des syntaxes concrètes pour assurer une vision meilleure des modèles simplePDL. Une première proposition est de représenter les modèles sous forme textuelle avec le plugin Xtext. Le fichier PDL.xtext propose une syntaxe concrète textuelle des processus.

2.5 L'outil Sirius

La visualisation va encore être plus améliorée avec une sytaxe concrère graphique. Une visulisation graphique des processus est présentée à l'aide du plugin Sirius. Voici le Simplepdl-Viewport qui décrit la vue graphique du modéle.

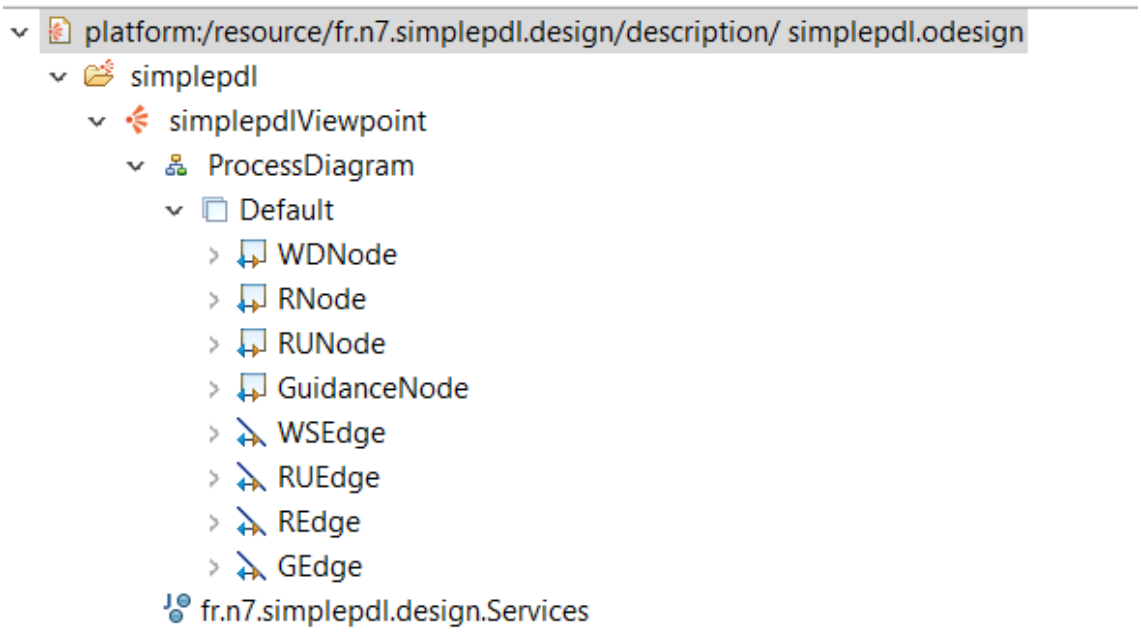

FIGURE 3 – SimplepdlViewport

Et la representation graphique est illustré par la figure 4.

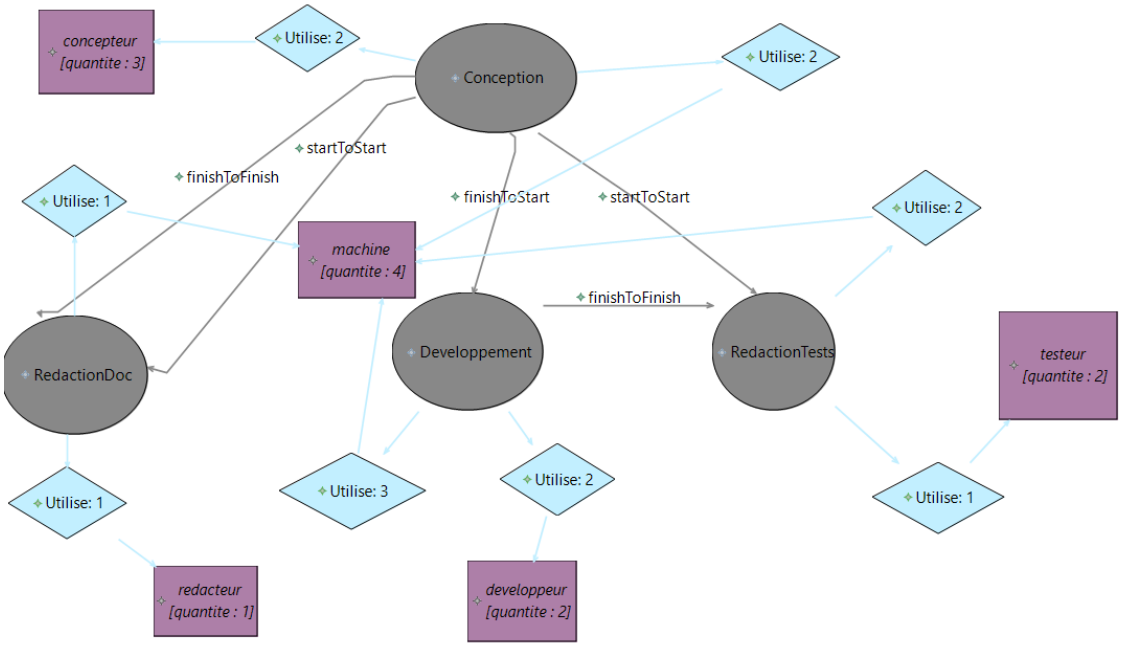

FIGURE 4 – Representation graphique

2.6 L'outil Acceleo

Le principe d'Acceleo est de s'appuyer sur des templates des fichiers à engendrer. qui est dans notre cas toPDL.mtl. Le fichier processus.pdl1 contient le syntaxe concrète textuelle engendré par Acceleo à partir d'un modèle SimplePDL. La figure 5 presente ce contenu.

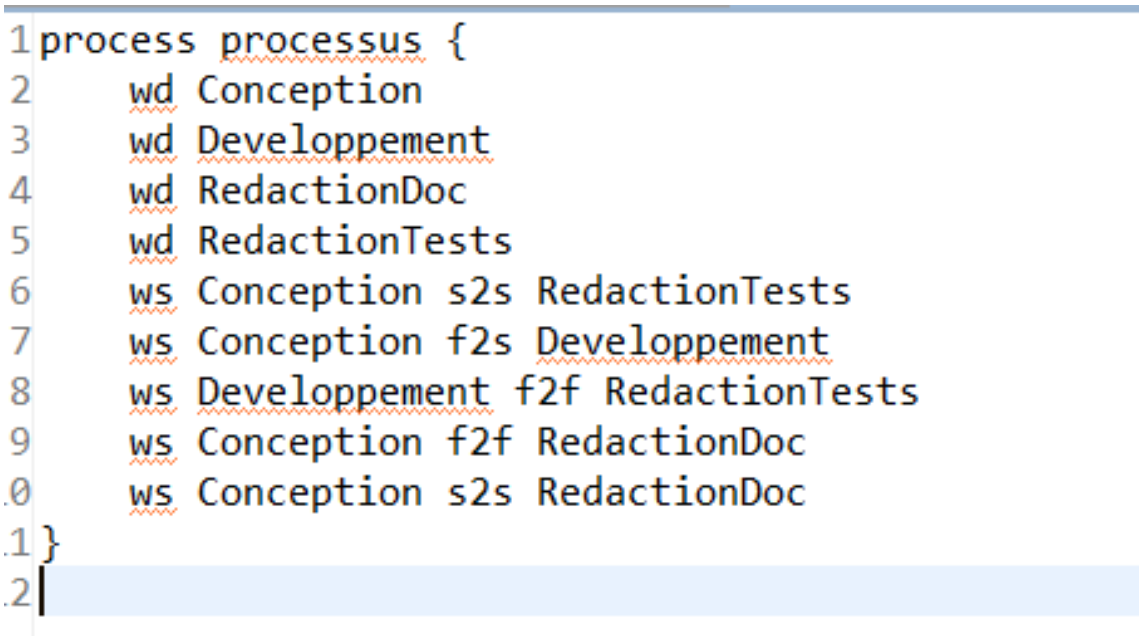

FIGURE 5 – Representation textuelle avec Acceleo

3 PetriNet

3.1 Métamodélisation

Pour générer un réseau de pétri interprétable par la boîte à outils Tina, il est nécessaire de créer un métamodèle de réseau de pétri, qui sera ensuite transformé. Les réseaux de pétri sont composés de places, de transitions et d'arcs, et présentent des caractéristiques telles que la présence de jetons dans les places et des poids pour les arcs. Le fichier PetriNet.ecore décrit le métamodèle de réseau de pétri, représenté dans la figure 6, et doit respecter certaines contraintes définies dans la section suivante ou grâce à des contraintes OCL.

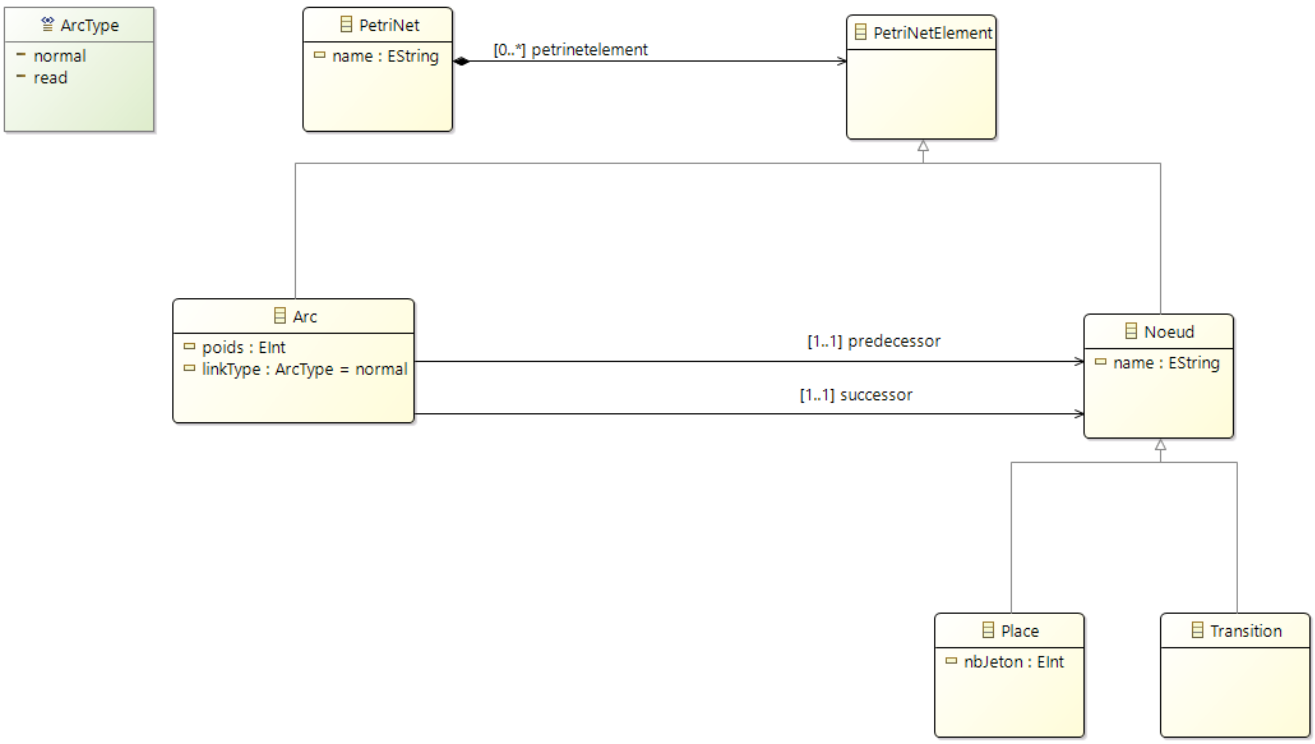

FIGURE 6 – métamodéle PetriNet

3.2 Contraintes OCL

Pour rendre les modèles du réseau de pétri plus robustes et utiles, il est nécessaire de renforcer le métamodèle décrit dans P etriNet.ecore avec des contraintes OCL.

3.3 Transformations M2M

La section Transformations du rapport est sans doute la partie la plus importante et la plus difficile du projet. En effet, elle décrit le processus qui permet de transformer le modèle SimplePDL en modèle de réseau de pétri. Pour y parvenir, nous avons développé un code Java qui lit les éléments du modèle SimplePDL et les utilise pour construire les éléments du modèle de réseau de pétri. Cette transformation a nécessité une compréhension approfondie des deux métamodèles et des règles de transformation entre eux. Nous avons également dû faire face à des défis tels que la gestion des dépendances entre les éléments du modèle et la vérification des contraintes de cohérence. Malgré ces difficultés, nous sommes parvenus à développer un code fonctionnel qui prend le modèle SimplePDL en entrée et génère le modèle de réseau de pétri correspondant en sortie. Le modele de reseau de petri est presenté par le fichier reseaudepetri.xmi et donné par la figure 7.

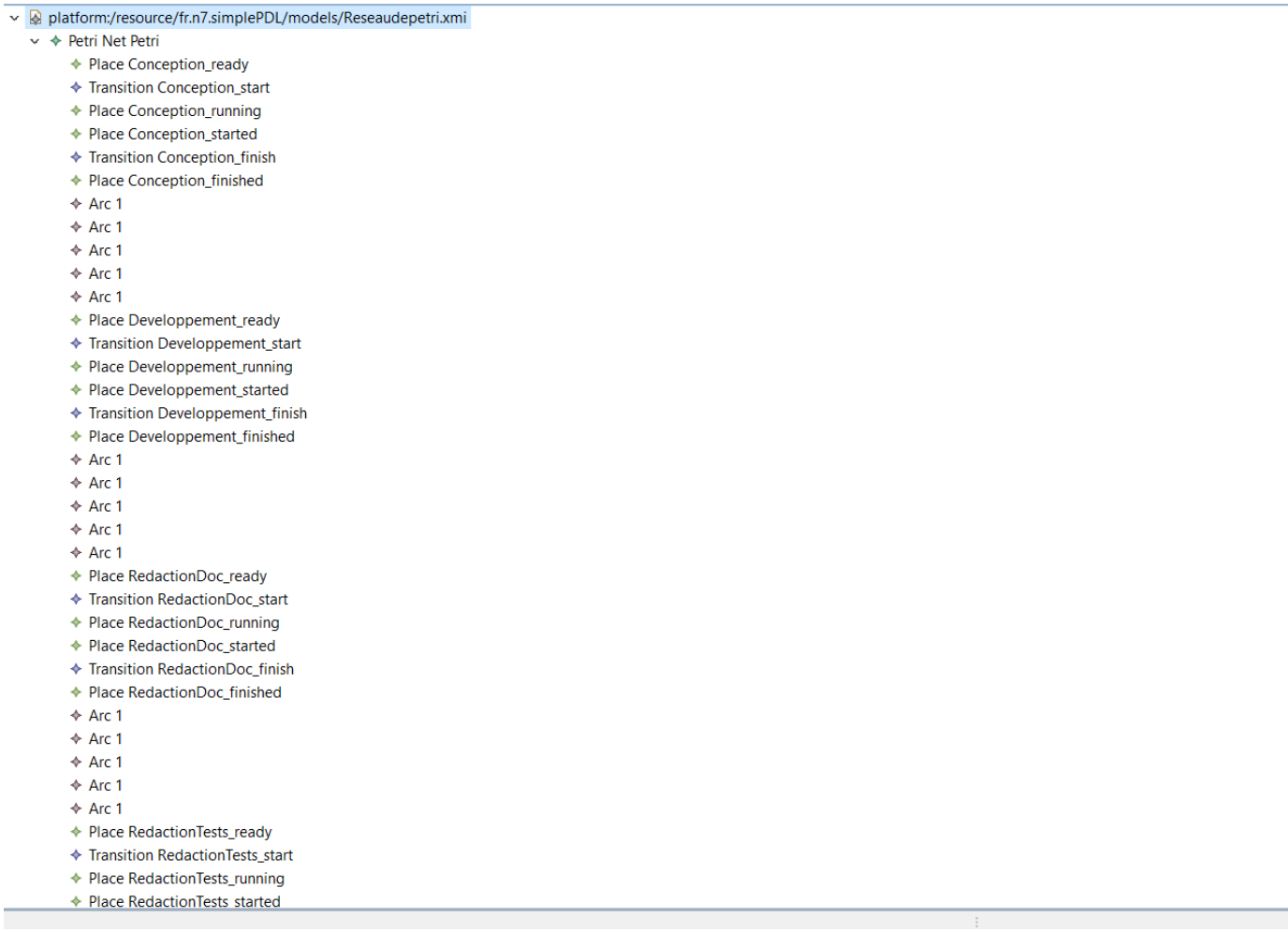

FIGURE 7 – Editeur arborescent du modéle PetriNet

3.4 Transformation M2T

A la sortie du code Java, on obtient un modèle de processus conforme au métamodèle du réseau de petri. Ce modèle va maintenant entrer dans la transformation modèle à texte qui transforme tout modèle conforme à un PetriNet vers un texte conforme à la syntaxe de la boîte à outils Tina. La transformation se fait par le plugin Acceleo. Le fichier Net2Tina.mtl propose cette transformation et voici le contenu de contenu de fichier Petri.net engendré par Acceleo.

```
pl Conception_ready (1)
pl Conception_running (0)
pl Conception_started (0)
pl Conception_finished (0)
pl Developpement_ready (1)
pl Developpement_running (0)
pl Developpement_started (0)
pl Developpement_finished (0)
pl RedactionDoc_ready (1)
pl RedactionDoc_running (0)
pl RedactionDoc_started (0)
pl RedactionDoc_finished (0)
pl RedactionTests_ready (1)
pl RedactionTests_running (0)
pl RedactionTests_started (0)
pl RedactionTests_finished (0)
tr Conception_start Conception_ready*1 -> Conception_started*1 Conception_running*1
tr Conception_finish Conception_started*1 -> Conception_finished*1
tr Developpement_start Developpement_ready*1 Conception_finished?1 -> Developpement_started*1 Dev
tr Developpement_finish Developpement_started*1 -> Developpement_finished*1
tr RedactionDoc_start RedactionDoc_ready*1 Conception_started?1 -> RedactionDoc_started*1 Redacti
tr RedactionDoc_finish RedactionDoc_started*1 Conception_finished?1 -> RedactionDoc_finished*1
tr RedactionTests_start RedactionTests_ready*1 Conception_started?1 -> RedactionTests_started*1 R
tr RedactionTests_finish RedactionTests_started*1 Developpement_finished?1 -> RedactionTests_fini
```

FIGURE 8 – Contenu de Petri.net

Aprés, nous avons utilisé l'outil Tina pour visualisé le reseau de Petri à partie du fichier Petri.net. la figure 9 montre le reseau de petri trouvé.

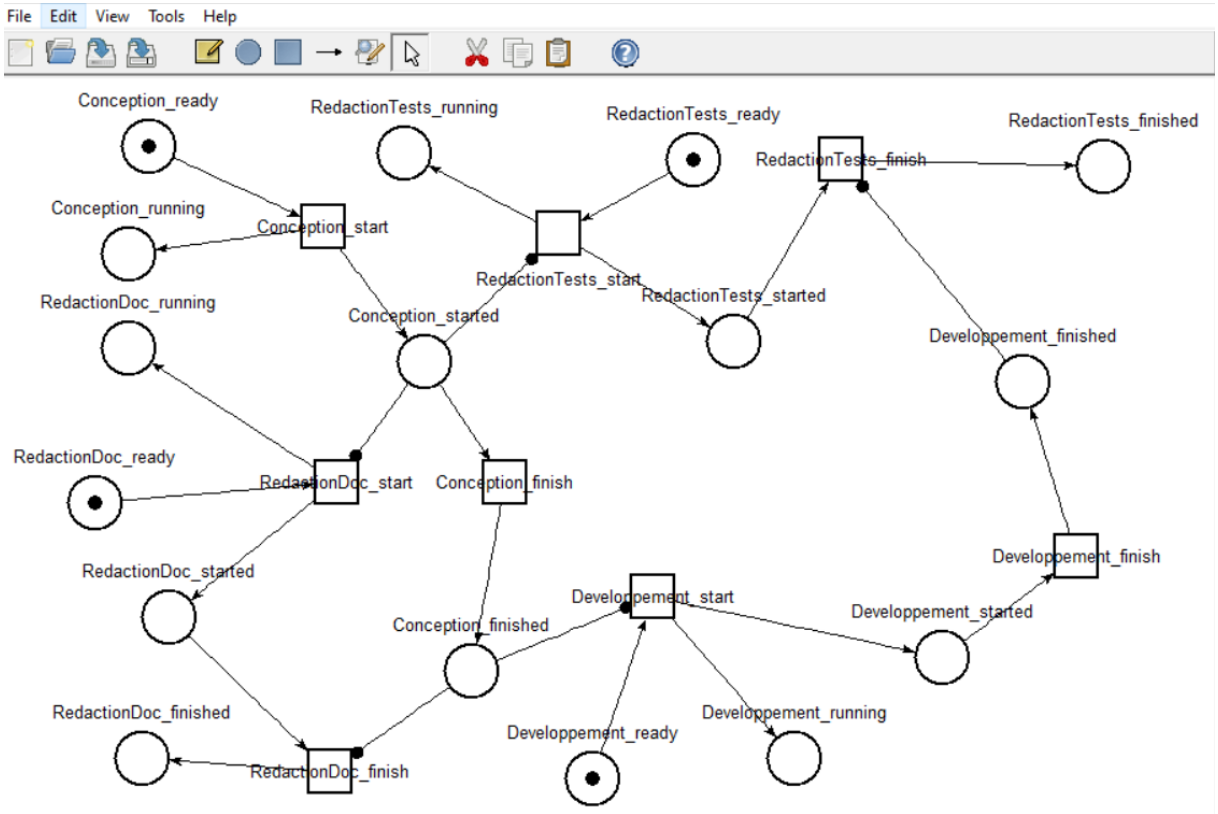

FIGURE 9 – Le reseau de petri avec Tina

Les modèles de processus conformes au métamodèle de réseau de pétri peuvent être utilisés pour générer les propriétés LTL qui vérifient la terminaison du processus. Plutôt que d'écrire ces propriétés manuellement pour chaque modèle, une transformation modèle à texte est effectuée pour engendrer un fichier .ltl contenant la condition de terminaison : toutes les activités doivent atteindre l'état "finished".

4 Conclusion

En conclusion, ce projet nous a permis de travailler sur la modélisation de processus avec le langage SimplePDL et la transformation de ces modèles en réseaux de pétri pour faciliter leur analyse et leur vérification. Bien que ce projet ait présenté des défis, notamment la modélisation des réseaux de pétri et l'écriture de règles OCL, il a été très enrichissant et nous avons pu acquérir de nouvelles compétences en matière de modélisation et de transformation de modèles.