



# Rapport Mini projet : IDM

EL ALOUT Ismail  
LAHMOUZ Zakaria

Département Sciences du Numérique - Deuxième année- Parcours HPCBD  
2022-2023

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Les métamodèles SimplePDL et PetriNet</b>	<b>3</b>
2.1	Le métamodèle SimplePDL . . . . .	3
2.2	Le métamodèle PetriNet . . . . .	4
2.3	Un exemple de PetriNet . . . . .	5
2.4	Les contraintes OCL associées aux métamodèles . . . . .	5
2.4.1	Contraintes OCL portant sur SimplePDL . . . . .	6
2.4.2	Contraintes OCL du reseau de Petri . . . . .	6
<b>3</b>	<b>L'éditeur graphique SimplePDL</b>	<b>7</b>
<b>4</b>	<b>Définition d'une syntaxe concrète textuelle de SimplePDL avec Xtext.</b>	<b>8</b>
<b>5</b>	<b>La transformation modèle à modèle</b>	<b>9</b>
5.1	La transformation ATL(Atlas Constraint Language) . . . . .	9
5.2	La transformation Java . . . . .	10
5.3	Comparaison entre la transformation ATL et Java . . . . .	11
<b>6</b>	<b>La transformation d'un réseau de Petri en Tina en utilisant Acceleo</b>	<b>12</b>
<b>7</b>	<b>Les propriétés LTL</b>	<b>13</b>
<b>8</b>	<b>Conclusion</b>	<b>13</b>

# List of Figures

1	Le métamodèle SimplePDL . . . . .	3
2	Le métamodèle PetriNet . . . . .	4
3	Un exemple de Petrinet visualisé avec l'outil Net Draw . . . . .	5
4	Le modèle ouvert avec l'éditeur graphique défini . . . . .	7
5	La syntaxe générée pour un exemple de Process . . . . .	8
6	Résultat de la transformation ATL d'un modèle SimplePDL . . . . .	10
7	Résultat de la transformation Java d'un modèle SimplePDL . . . . .	11
8	Le modèle obtenu à l'aide de Tina . . . . .	12
9	La verification de terminaison du processus à l'aide de l'outil Selt . . . . .	13

# 1 Introduction

L'objectif du mini projet est produire une chaine de vérification de modèles de processus SimplePDL dans le but de vérifier leur cohérence, en particulier pour savoir si le processus peut se terminer ou non. Ainsi ,on suit plusieurs étapes : D'abord, la construction des métamodèles grâce à Ecore. Ensuite, On définit les contraintes des métamodèles avec OCL(Object Constraint Language), puis on aborde transformations du modèle de processus en reseaux de petri à l'aide de ATL et Java. Enfin, on utilise la boîte à outils Tina pour le Model cheking

## 2 Les métamodèles SimplePDL et PetriNet

### 2.1 Le métamodèle SimplePDL

Le métamodèle SimplePDL sert à définir des modèles de processus. Notre métamodèle SimplePDL est représenté dans la figure ci-dessous.

Ce métamodèle est constitué :

- Des activités (WorkDefinition) qui sont les taches que le processus doit réaliser.
- Des dépendances (WorkSequence) qui relient les activités.
- Des ressources (Ressource) : la réalisation d'une tache peut nécessiter des ressources.

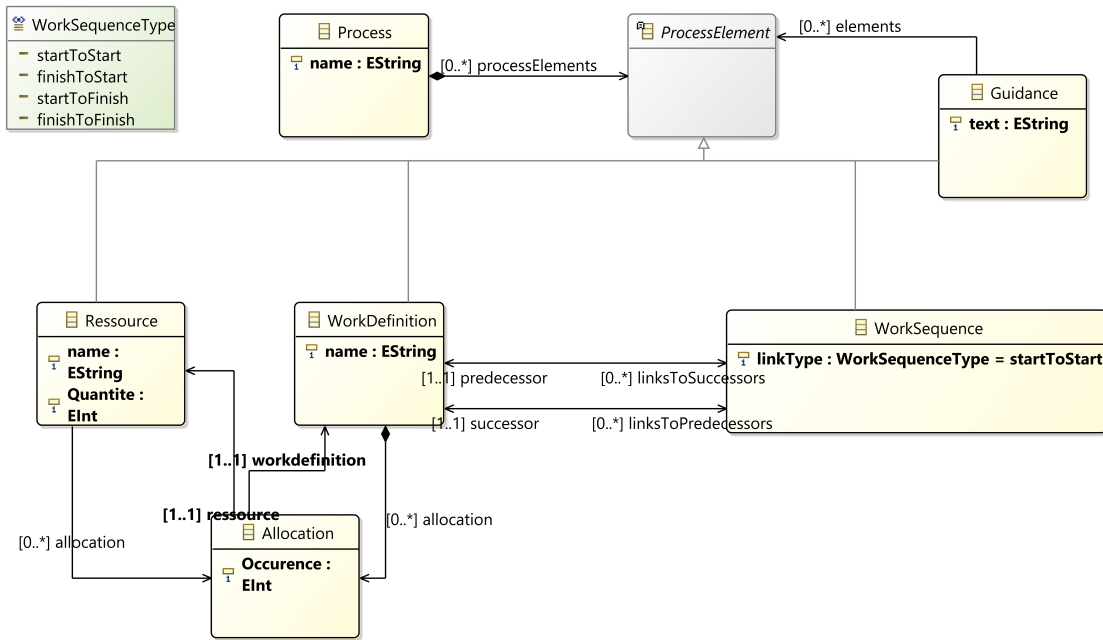


Figure 1: Le métamodèle SimplePDL

## 2.2 Le métamodèle PetriNet

Un réseau de Petri est un tuple  $(S, T, F, M0, W)$  où :

- $S$  définit une ou plusieurs places.
- $T$  définit une ou plusieurs transitions.
- $F$  définit un ou plusieurs arcs (flèches).
- Un arc ne peut pas être connecté entre 2 places ou 2 transitions ; plus formellement :  $F \subset (S \times T) \cup (T \times S)$ .
- $M0 : S \rightarrow \mathbb{N}$  appelé marquage initial (ou place initiale) où, pour chaque place  $s \in S$ , il y a  $n \in \mathbb{N}$  jetons.
- $W : F \rightarrow \mathbb{N}$  appelé ensemble d'arcs primaires, assignés à chaque arc  $f \in F$  un entier positif  $n \in \mathbb{N}$  qui indique combien de jetons sont consommés depuis une place vers une transition, ou sinon, combien de jetons sont produits par une transition et arrivent pour chaque place.

Notre métamodèle PetriNet est donné sur la figure ci-dessous.

Ce métamodèle est composé :

- Des noeuds (Noeud) qui sont soit des places ou des transitions.
- Des arcs (Arc) reliant les noeuds; on a 2 types d'arcs regroupés dans l'énumération ArcType : normal (entre une place et une transition) et read\_arc (entre une place d'entrée et une transition). Ce deuxième type consiste à vérifier que la place a bien au moins le nombre de jetons indiqué sur cet arc. Si c'est le cas, la transition est exécutable.

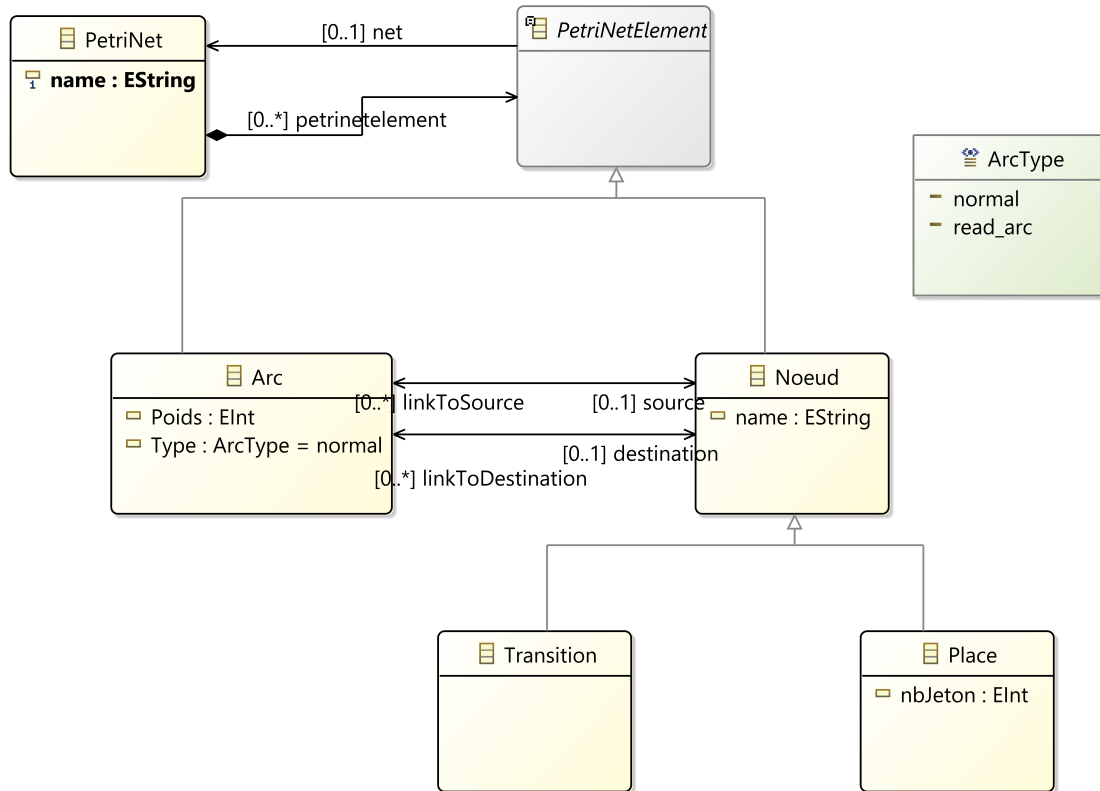


Figure 2: Le métamodèle PetriNet

### 2.3 Un exemple de PetriNet

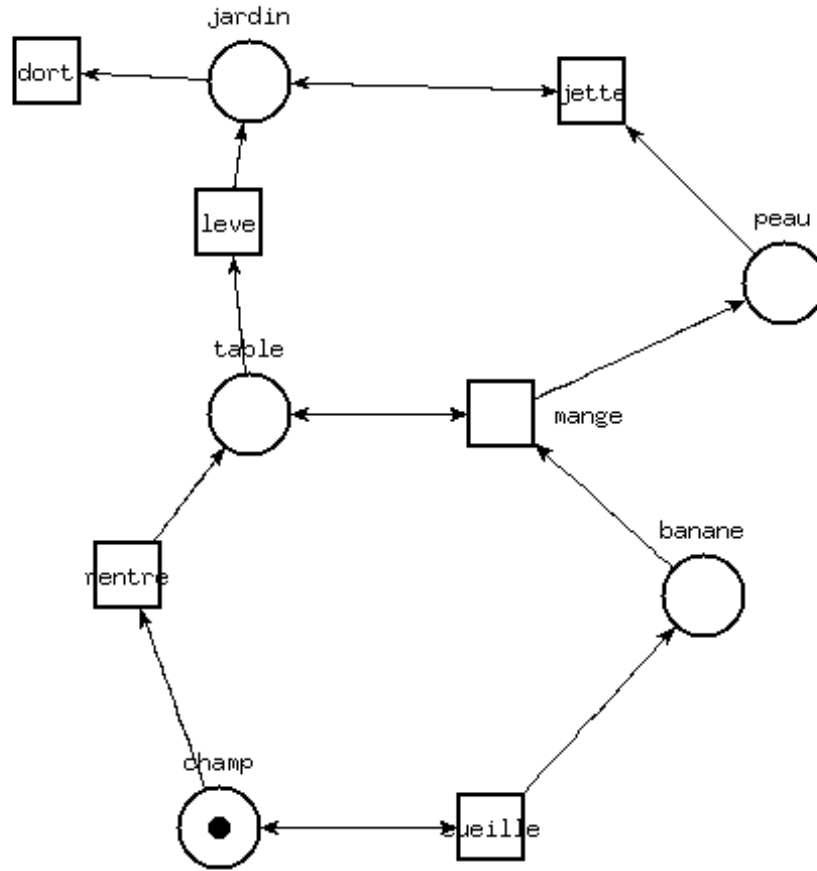


Figure 3: Un exemple de Petrinet visualisé avec l'outil Net Draw

### 2.4 Les contraintes OCL associées aux métamodèles

Puisque Ecore ne permet pas d'exprimer toutes les contraintes des modèles de processus alors on définit les contraintes avec le langage OCL(Object Constraint Language). Ainsi, le métamodèle Ecore et les contraintes OCL définissent la syntaxe abstraite du langage de modélisation considéré.

#### **2.4.1 Contraintes OCL portant sur SimplePDL**

Les contraintes liées à SimplePDL sont données dans le fichier SimplePDL.ocl :

1. Une dépendance ne peut pas être réflexive.
2. Deux sous-activités différentes d'un même processus ne peuvent pas avoir le même nom.
3. Le nom d'une activité doit être composé d'au moins un caractère.
4. Les dépendances du modèle de processus ne provoquent pas de blocage.
5. Une activité ne peut pas demander un nombre d'occurrences d'une ressource supérieur à la quantité disponible pour cette ressource.
6. Deux ressources différentes ne peuvent pas avoir le même nom.

#### **2.4.2 Contraintes OCL du reseau de Petri**

Le fichier petrinet.ocl représente les contraintes liées à PetriNet:

1. Chaque place a obligatoirement un nombre entier naturel de jetons.
2. Le nombre de jetons que possède l'arc est un entier naturel.
3. Un arc ne peut pas lier deux places, en fait, il lie une Place avec une Transition.

### 3 L'éditeur graphique SimplePDL

À l'instar d'une syntaxe concrète textuelle, une syntaxe concrète graphique fournit un moyen de visualiser et/ou éditer plus agréablement et efficacement un modèle. Nous allons utiliser l'outil Sirius développé par les sociétés Obeo et Thales, et basé sur les technologies Eclipse Modeling comme EMF et GMF. Il permet de définir une syntaxe graphique pour un langage de modélisation décrit en Ecore et d'engendrer un éditeur graphique intégré à Eclipse.

La syntaxe graphique est définie sur le fichier SimplePDL.odesign.

Nous avons défini un modèle conforme à SimplePDL afin de tester notre syntaxe graphique. On peut voir le résultat de la vue graphique du modèle sur la figure ci-dessous. Nous avons également défini des objets graphiques; regroupés en section sur la palette pour manipuler notre vue graphique.

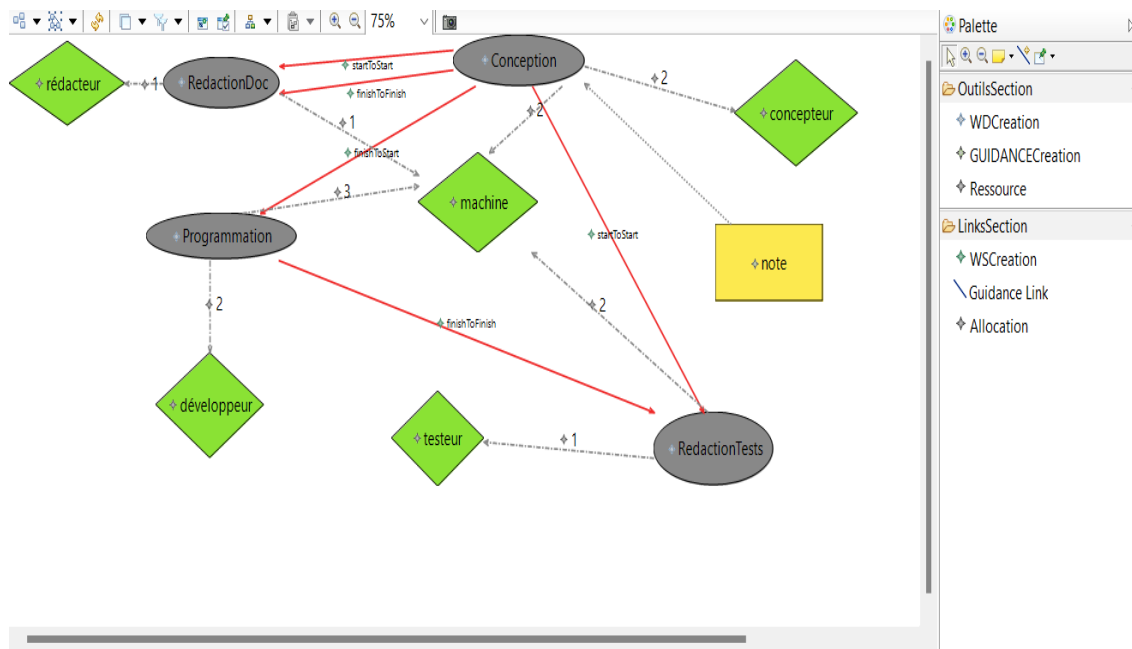


Figure 4: Le modèle ouvert avec l'éditeur graphique défini

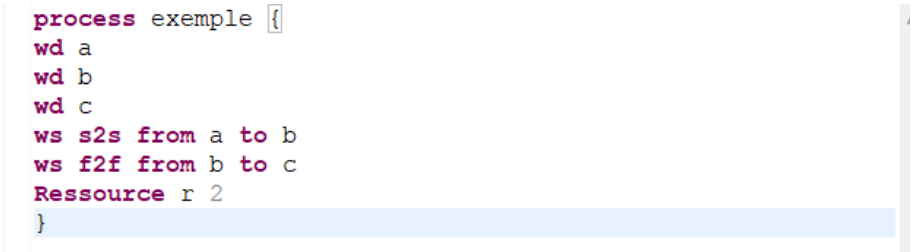
## 4 Définition d'une syntaxe concrète textuelle de SimplePDL avec Xtext.

La syntaxe abstraite d'un DSML (exprimée en Ecore ou un autre langage de métamodélisation) ne peut pas être manipulée directement. Le projet EMF d'Eclipse permet d'engendrer un éditeur arborescent et les classes Java pour manipuler un modèle conforme à un métamodèle. Cependant, ce ne sont pas des outils très pratiques lorsque l'on veut saisir un modèle. Aussi, il est souhaitable d'associer à une syntaxe abstraite une ou plusieurs syntaxes concrètes pour faciliter la construction et la modification des modèles. Ces syntaxes concrètes peuvent être textuelles.

Nous avons utilisé l'outil Xtext qui fait partie du projet TMF (Textual Modeling Framework). Il permet non seulement de définir une syntaxe textuelle pour un DSL mais aussi de disposer, au travers d'Eclipse, d'un environnement de développement pour ce langage, avec en particulier un éditeur syntaxique (coloration, complétion, outline, détection et visualisation des erreurs, etc).

La description Xtext de la syntaxe associée à SimplePDL est donnée dans le fichier PDL.xtext.

Un exemple de la syntaxe engendrée est donné sur la figure ci-dessous.



```
process exemple {  
  wd a  
  wd b  
  wd c  
  ws s2s from a to b  
  ws f2f from b to c  
  Ressource r 2  
}
```

Figure 5: La syntaxe générée pour un exemple de Process



## 5 La transformation modèle à modèle

### 5.1 La transformation ATL(Atlas Constraint Language)

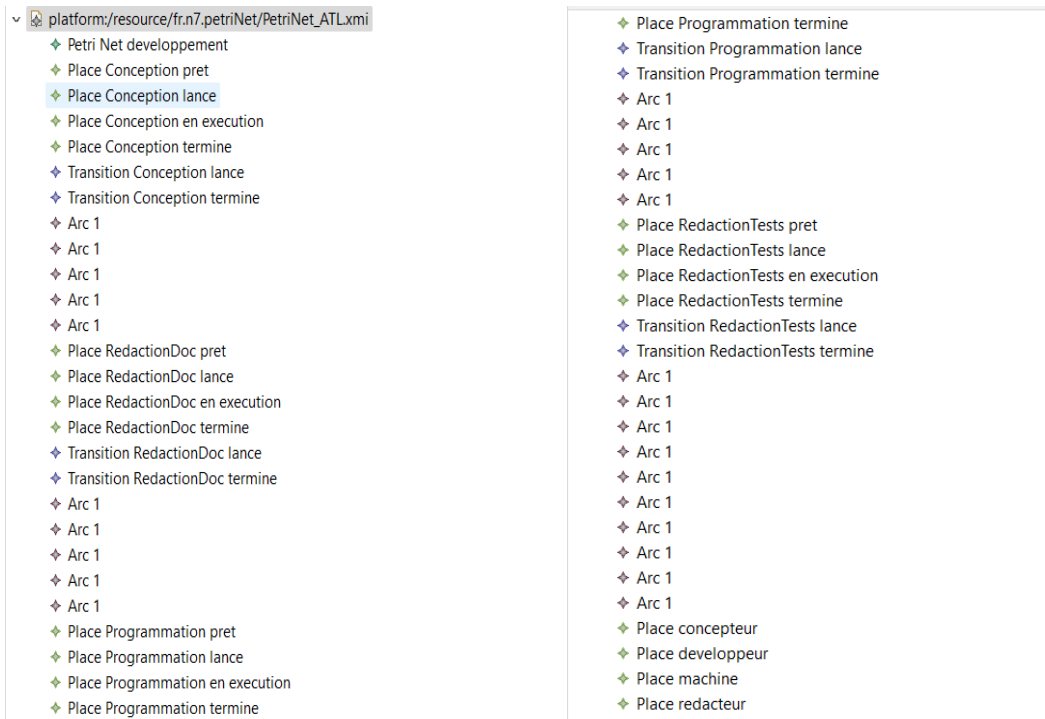
ATL permet de transformer un modèle SimplePdl en un modèle PetriNet en suivant les étapes suivantes:

1. Traduire un Process en un PetriNet de meme nom
2. Convertir toutes les WorkSequences en un arc qu'on relie aux WorkDefinitions appropriées.

Ajouter l'arc au réseau.

3. Convertir toutes les workDefinitions en réseaux de pétri : 5 arcs, 2 transitions et 4 places. Relier le tout correctement et l'ajouter au réseau de Pétri.

4. Convertir toutes les ressources en places et les relier convenablement au reste du réseau. Ajouter tous les éléments créés au réseau.



```

    ◆ Arc 1
    ◆ Arc 1
    ◆ Arc 1
    ◆ Arc 1
    ◆ Arc 1
    ◆ Place concepteur
    ◆ Place developpeur
    ◆ Place machine
    ◆ Place redacteur
    ◆ Place testeur
    ◆ Arc 2
    ◆ Arc 2
    ◆ Arc 2
    ◆ Arc 2
    ◆ Arc 1
    ◆ Arc 1
    ◆ Arc 1
    ◆ Arc 1
    ◆ Arc 2
    ◆ Arc 2
    ◆ Arc 3
    ◆ Arc 3
    ◆ Arc 2
    ◆ Arc 2
    ◆ Arc 1
    ◆ Arc 1
    > platform:/resource/fr.n7.petriNet/PetriNet.genmodel
    > platform:/resource/fr.n7.petriNet/PetriNet.ecore

```

---

Figure 6: Résultat de la transformation ATL d'un modèle SimplePDL

## 5.2 La transformation Java

Le fichier processusToPetri.java représente la transformation java du modèle SimplePdl au modèle PetriNet en suivant ces étapes principales :

1. Charger les Packages SimplePdl et PetriNet et les enregistrer dans le package d'eclipse
2. Configurer l'entrée qu'on fournit au programme java et la sortie rendue par le programme.
3. On récupère le premier élément du modèle process(élément à la racine), ensuite instancier la fabrique.
4. On commence par construire le PetriNet, en convertissant les WorkDefinitions et Resources en Places, puis WorkSequences en Arcs.

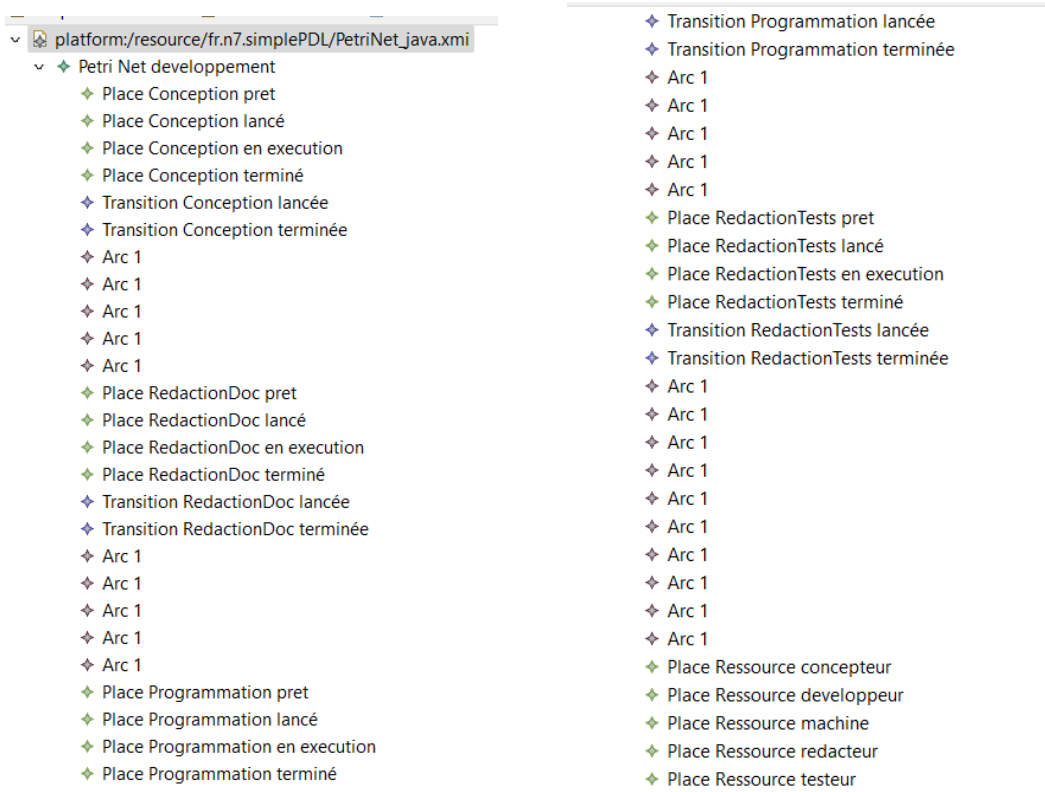


Figure 7: Résultat de la transformation Java d'un modèle SimplePDL

### 5.3 Comparaison entre la transformation ATL et Java

La transformation ATL est plus efficace que celle de Java. En effet, l'implémentation avec ATL est simple et claire : Les différentes règles, "rule" en ATL, sont facilement distinguées. De plus, la fonctionnalité "resolveTemp" permet d'attribuer facilement les éléments du modèle SimplePDL au modèle PetriNet.

## 6 La transformation d'un réseau de Petri en Tina en utilisant Aceleo

Dans cette partie, on s'intéresse aux transformations modèle vers texte (M2T). Nous avons utilisé l'outil Aceleo de la société Obeo.

La syntaxe textuelle désirée est la syntaxe en extension .net utilisée par les outils de Tina. Cette transformation va nous permettre par la suite de tester et visualiser notre modèle graphiquement avec l'outil NetDraw.

Le fichier correspondant à la Template Aceleo de la transformation d'un réseau de Petri en Tina : toTina.mtl.

Nous avons appliqué la transformation modèle à modèle (AT ou Java) au modèle de simplePDL du sujet (fichier Process2.xmi). Ensuite, nous avons appliqué la transformation PetriNet to Tina sur le modèle obtenu. La figure ci-dessous représente une vue graphique du modèle avec l'outil NetDraw.

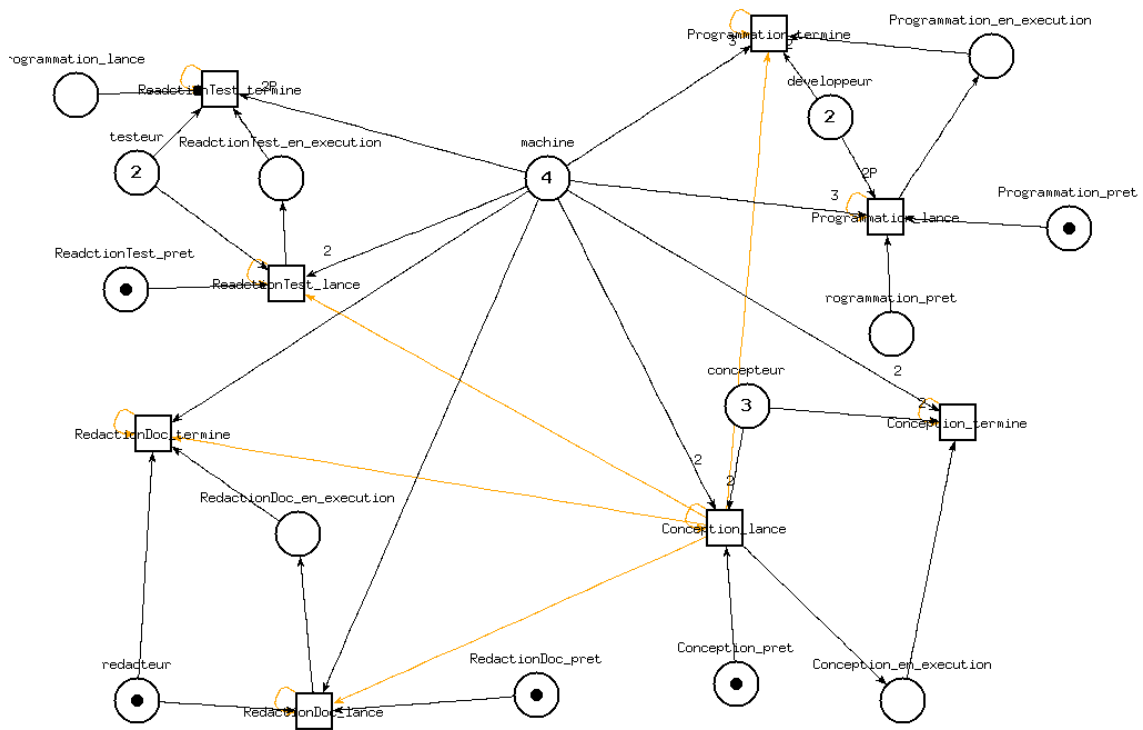


Figure 8: Le modèle obtenu à l'aide de Tina

## 7 Les propriétés LTL

Dans cette partie, nous avons défini une transformation modèle vers texte (SimplePDLtoLLTL) en utilisant l'outil Acceleo pour engendrer le fichier LTL permettant de vérifier la terminaison du Process.

Le fichier correspondant à la Template Acceleo de cette transformation : toLTL.mtl. Le but d'utiliser la propriété - <> finished plutôt que <> finished est d'avoir la réponse False et d'avoir le chemin à suivre pour avoir une terminaison, au lieu d'avoir un résultat True et sans aucun chemin à suivre. Les résultats obtenus après exécution avec l'outil Selt de Tina sont donnés sur la figure ci-dessous.

```
- bye
PS C:\Users\33665\Desktop\tina-3.7.0\bin> ./tina dnet.net d.ktz

# net dnet, 16 places, 8 transitions, 25 arcs
# bounded, not live, not reversible
# abstraction      count      props      psets      dead      live #
#   states         26         16         26         1         1 #
#   transitions    47         8         8         0         0 #
PS C:\Users\33665\Desktop\tina-3.7.0\bin> ./selt -p -S d.scn d.ktz -prelude d.ltl
Selt version 3.7.0 -- 06/05/22 -- LAAS/CNRS
ktz loaded, 26 states, 47 transitions
0.000s

- source d.ltl:
operator finished : prop
operator running  : prop
operator started  : prop
operator ready    : prop
TRUE
TRUE
TRUE
FALSE
state 0: l_scn25 n_concnpionready n_programmationready n_redactionDecready n_testready
```

Figure 9: La verification de terminaison du processus à l'aide de l'outil Selt

## 8 Conclusion

En guise de conclusion, ce projet nous a permis d'appliquer les notions vues en cours/TD/TP. Nous trouvons aussi que le projet est riche d'applications et reflète l'importance de la modélisation dans un projet ainsi que la possibilité du model-checking à travers Tina. Nous remercions chaleureusement toutes les personnes qui ont donné des remarques précieuses pour réussir ce projet.