Rapport de mini projet

Chaîne de vérification de modèles de processus

Noms des auteurs
JAMES Christopher
FIGUIN Cyril

Table des matières

1 Introduction

L'objectif de ce mini projet est de produire une **chaîne de vérification de modèles de processus** de type simplePDL dans le but de vérifier leurs cohérences, en particulier pour savoir si le processus décrit peut se terminer ou non.
Pour ce faire, nous allons transformer les modèles de type simplePDL en métamodèles plus simples : **les réseaux de pétri**.

2 Tâches à réaliser pour ce projet

- **T1** Compléter le métamodèle SimplePDL pour prendre en compte les ressources.
- **T2** Définir le métamodèle PetriNet.
- **T3** Développer un éditeur graphique SimplePDL pour saisir graphiquement un modèle de processus, y compris les ressources.
- **T4** Définir les contraintes OCL pour capturer les contraintes qui n'ont pu l'être par les métamo- dèles (SimplePDL et PetriNet).
- **T5** Donner une syntaxe concrète textuelle de SimplePDL avec Xtext.
- **T6** Définir une transformation SimplePDL vers PetriNet en utilisant EMF/Java.
- **T7** Définir une transformation SimplePDL vers PetriNet en utilisant ATL.
- **T8** Valider la transformation SimplePDL vers PetriNet en faisant des tests.
- **T9** Définir une transformation PetriNet vers Tina en utilisant Acceleo.
- **T10** Engendrer les propriétés LTL permettant de vérifier la terminaison d'un processus et les appliquer sur différents modèles de processus.
- **T11** Engendrer les propriétés LTL correspondant aux invariants de SimplePDL pour valider la transformation écrite (voir section 1.2).

3 Tâche T1

Prise en compte des ressources (ajout d'une case **ressource** et **besoin**) telles que :

- la case **ressource** permette de nommer les ressources et de les énumérer.
- la case **besoin** permette de former le lien (une activité peut avoir plus ressources) et possibilité d'ajouter des fils.

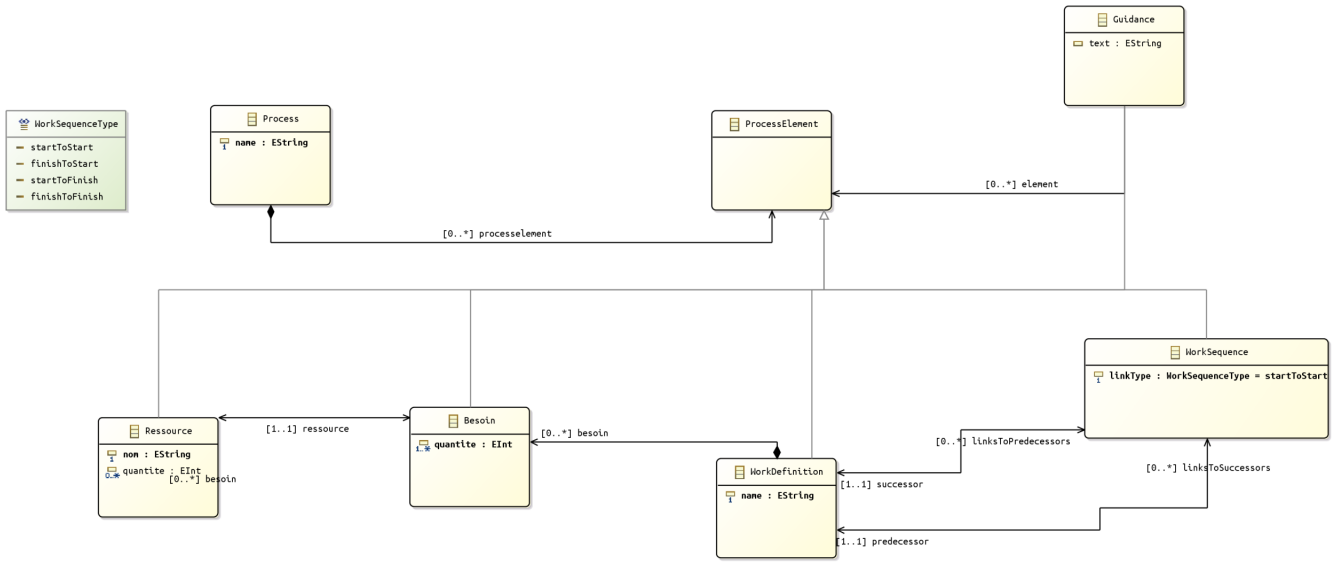

FIGURE 1 – Métamodèle SimplePDL

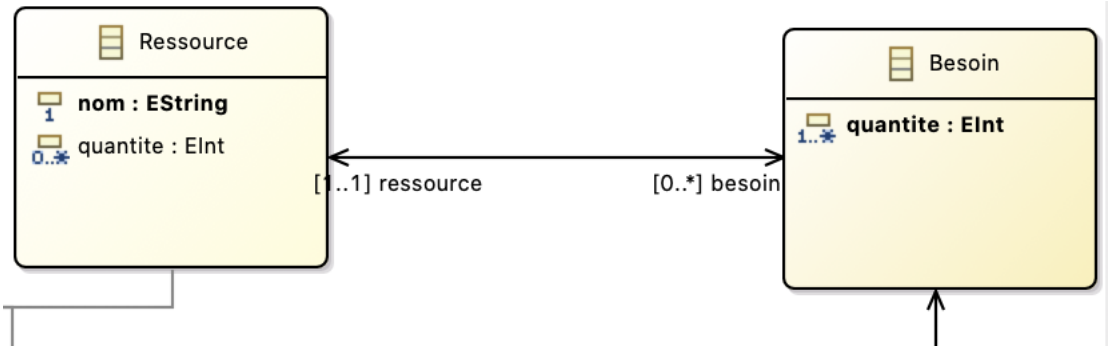

FIGURE 2 – Cases ressource et besoin

4 Tâche T2

Modèle Petrinet

Nous avons défini ce métamodèle "Petrinet" permettant d'identifier les réseaux de Petri. Nous nous sommes tout d'abord basé sur le modèle SimplePDL en créant une classe **PetriNetElement**. Ensuite, nous avons créé une classe **Arc** et une classe **Node** qui héritent de PetriNetElement. Enfin, deux classes héritent de Node : la classe **Transition** et la classe **Place**.

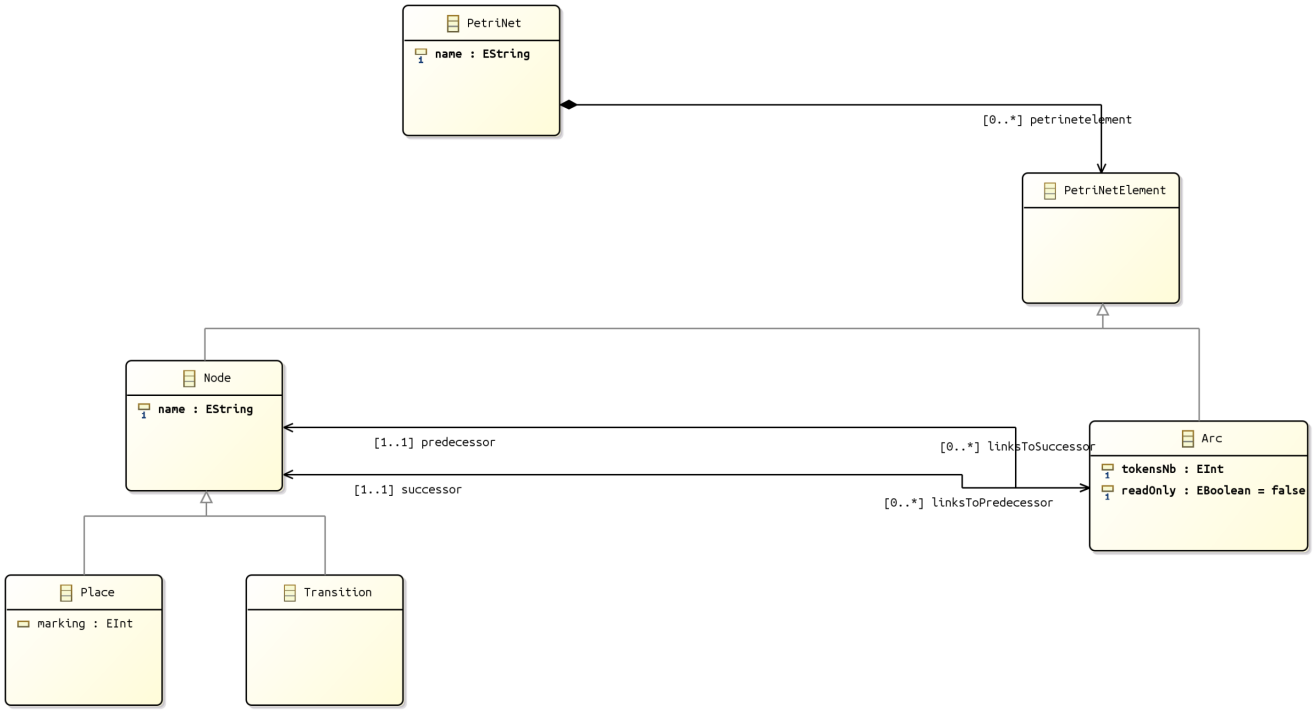

FIGURE 3 – Métamodèle PetriNet

5 Tâche T3

L'éditeur graphique Sirius est un moyen de pouvoir lire, mais également d'éditer un exemple de modèle SimplePDL comprenant les ressources de façon graphique. Un utilisateur peut ainsi, grace à cet éditeur, créer son exemple graphiquement. L'objectif est d'avoir une meilleure comprehension du modèle.
Nous n'avons pas réussi à implanter l'éditeur pour les éléments Guidance.

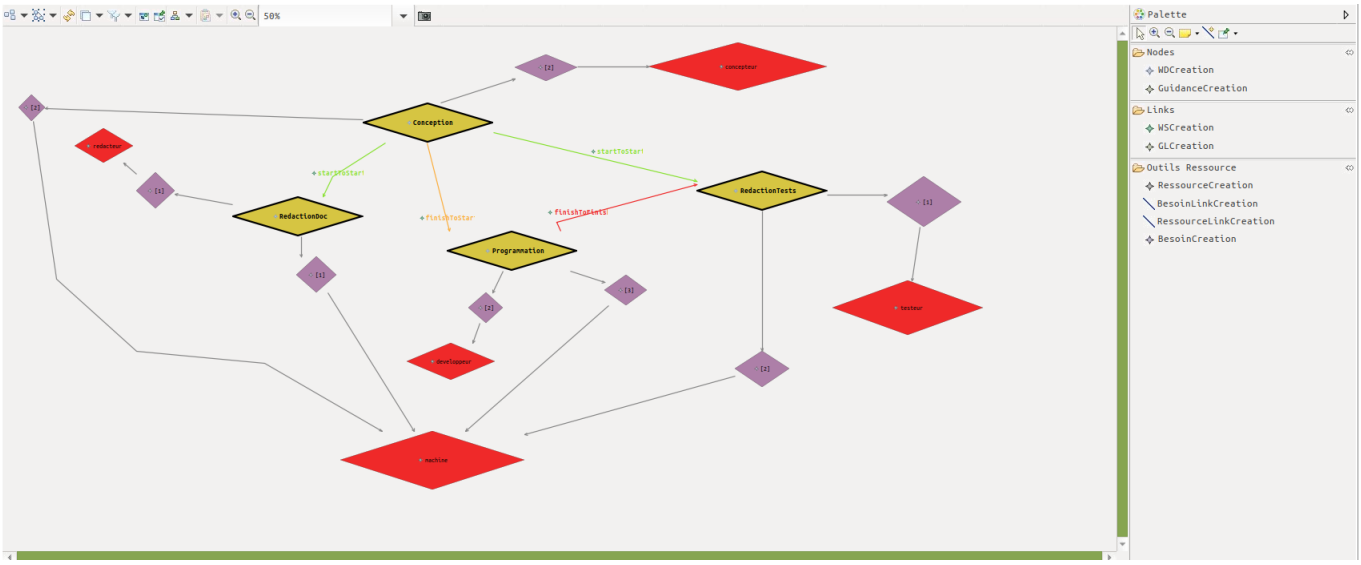

FIGURE 4 – Editeur graphique pour la saisie des modèles de processus

6 Tâche T4

6.1 Contraintes sur SimplePDL :

Contraintes sur chaque **ressources** et sur chaque **workDefinition**

— **nomValide** qui permet de vérifier la taille du nom (> 2 caractères).
— **compositionNom** pour vérifier qu'un nom est bien composé.
— **nomUnique** pour vérifier que deux processus n'ont pas les mêmes noms.
— **nonReflexif** pour la non refléxivité de deux dépendances.

6.2 Contraintes sur PetriNet :

— **nomValide** qui permet de vérifier la taille du nom (> 2 caractères).
— **compositionNom** pour vérifier que le nom d'une activité est bien composée.
— **nomUnique** pour qu'un nom soit unique.
— **nomValide** pour la contrainte sur les node.
— **positiveToken** sur les arcs (nombre de jetons positif).
— **nonReflexif** pour la non réflexivité des arcs.
— **positiveMarking** sur les place.

7 Tâche T5

Le principe de la création de modèles avec Xtext est de pouvoir qualifier de manière complète un métamodèle de type SimplePDL de manière simple.
Ainsi ; nous avons généré une grammaire à partir de ce qui nous était proposé, afin qu'elle soit plus simple et plus compréhensible pour un utilisateur lambda.

Voici la description écrite Xtext pour la syntaxe SimplePDL :

```
// automatically generated by Xtext
grammar fr.n7.txt.SimplePDL with org.eclipse.xtext.common.Terminals

import "http://simplepdl"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Process : 'process' name=EString'{'
        processelement+=ProcessElement*
    '}';

ProcessElement : WorkDefinition | WorkSequence | Guidance | Besoin | Ressource;


EString returns ecore::EString:
    STRING | ID;


WorkDefinition : 'wd' name = ID ;

WorkSequence :'ws' linkType=WorkSequenceType
        'from' predecessor=[WorkDefinition]
        'to' successor=[WorkDefinition] ;

Guidance :
    {Guidance}
    'Guidance' 'text' text=EString
    'element' '(' element+=[ProcessElement|EString] ( "," element+=[ProcessElement|EString])* ')' ?
    ;

Ressource :
    'Ressource' nom=EString 'quantite'  quantite+=EInt
    ;

Besoin returns Besoin:
    'Besoin' 'quantite' quantite+=EInt 'ressource' ressource=[Ressource|EString]
    ;

enum WorkSequenceType :
    startToStart = 's2s' | finishToStart = 'f2s' | startToFinish = 's2f' | finishToFinish = 'f2f';


EInt returns ecore::EInt:
    '-'? INT;

```

FIGURE 5 – Description textuelle de la syntaxe SimplePDL

Voici un exemple de syntaxe concrète textuelle conforme à SimplePDL avec Xtext :

```
process Test {
    wd start
    wd finish
    ws s2f from start to finish
}
```

FIGURE 6 – Exemple de syntaxe concrète textuelle

8 Tâche T6

Nous n'avons pas réussi à implanter SimplePDL2PetriNet.java.

9 Tâche T7

Nous avons écrit, grâce à ATL, une transformation de SimplePDL vers PetriNet. Pour se faire, il nous a fallu faire des transformations pour les objets suivants :

- **Process** : Un Process devient un PetriNet dans cette transformation.
- **WorkDefinition** : Une WorkDefinition devient 4 places notStarted, started, in_progress et finished et deux transitions start et finish.
- **WorkSequence** : Une WorkSequence devient un read arc entre une place de l'activité précédente (started ou finished) et une transition de l'activité cible (start ou finish).

Notre transformation prend également en compte les différentes ressources allouées.

10 Tâche T8

Nous avons effectué différents tests afin de vérifier que la chaîne est complète :

- pdl-sujet.simplepdl (obtenu avec XText) > pdl-sujet.petrinet (transformation ATL) > pdl-sujet.net (Acceleo PetriNet2Tina) > Validation sur Tina
- modelisation Sirius > pdl-sujet.simplepdl > pdl-sujet.petrinet > pdl-sujet.net > Validation sur Tina

11 Tâche T9

Voici le résultat de la transformation du modèle Petrinet vers Tina avec Acceleo :

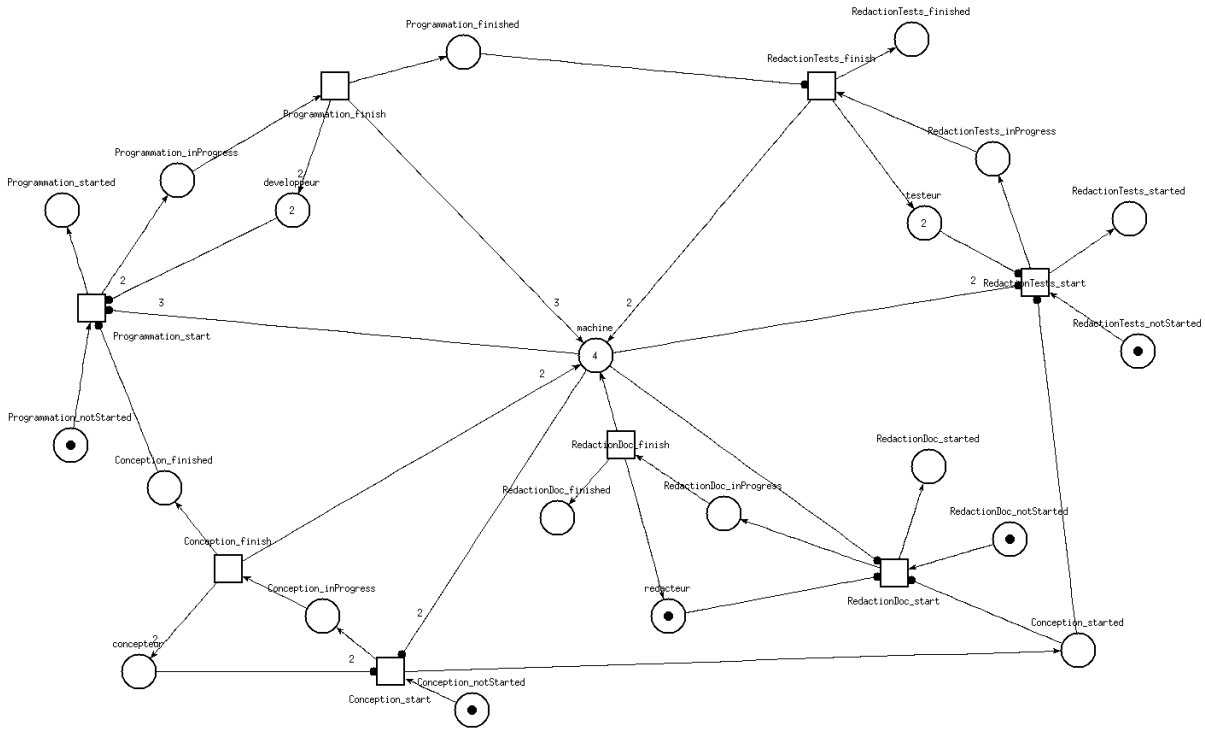

FIGURE 7 – Transformation modèle Petrinet vers Tina

Et un exemple avec le modèle des saisons :

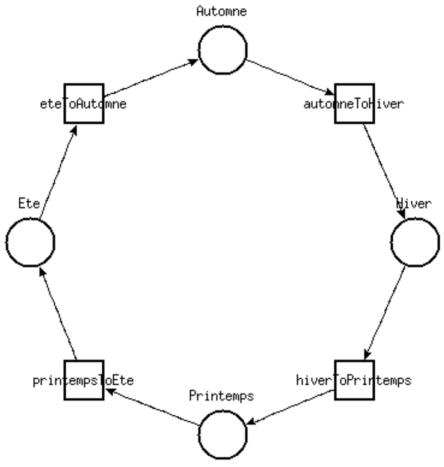

FIGURE 8 – Exemple du modèle des saisons transformé en Tina

12 Tâche T10

Voici le fichier LTL associé à notre réseau de pétri utilisé pour vérifier la propriété souhaitée (le processus se termine bien) :

```
1 op finished = Conception_finished /\ RedactionDoc_finished /\ Programmation_finished /\ RedactionTests_finished;
2
3 [] (finished => dead);
4 [] <> dead ;
5 [] (dead => finished);
6 <> finished;
```

FIGURE 9 – Fichier developpement.ltl

Ce fichier a été généré grace à Acceleo. Voici le résultat après vérification avec l'outil selt de tina :

```
MacBook-Pro-de-Chris:bin chris$ selt -p -S developpement.scn developpement.ktz -prelude developpement.ltl
Selt version 3.6.0 -- 07/06/20 -- LAAS/CNRS
ktz loaded, 26 states, 47 transitions
0.002s

- source developpement.ltl;
operator finished : prop
TRUE
TRUE
TRUE
TRUE
0.003s
```

FIGURE 10 – Résultats de la vérification des propriétés de terminaisons

Les propriétés sont bien vérifiées, le processus se termine.

13 Tâche T11

Afin de vérifier les invariants sur les modèles de processus (chaque activité est soit non commencée, soit en cours, soit terminée), (une activité terminée ne peut plus évoluer), les propriétés suivantes ont été testées par l'outil selt :

```
[] (Conception_notStarted \/ Conception_started);
[] (RedactionDoc_notStarted \/ RedactionDoc_started);
[] (Programmation_notStarted \/ Programmation_started);
[] (RedactionTests_notStarted \/ RedactionTests_started);

[] (Conception_notStarted \/ Conception_inProgress \/ Conception_finished);
[] (RedactionDoc_notStarted \/ RedactionDoc_inProgress \/ RedactionDoc_finished);
[] (Programmation_notStarted \/ Programmation_inProgress \/ Programmation_finished);
[] (RedactionTests_notStarted \/ RedactionTests_inProgress \/ RedactionTests_finished);

Conception_finished => [] Conception_finished;
RedactionDoc_finished => [] RedactionDoc_finished;
Programmation_finished => [] Programmation_finished;
RedactionTests_finished => [] RedactionTests_finished;

Conception_notStarted => () (Conception_notStarted \/ (Conception_started /\ Conception_inProgress));
RedactionDoc_notStarted => () (RedactionDoc_notStarted \/ (RedactionDoc_started /\ RedactionDoc_inProgress));
Programmation_notStarted => () (Programmation_notStarted \/ (Programmation_started /\ Programmation_inProgress));
RedactionTests_notStarted => () (RedactionTests_notStarted \/ (RedactionTests_started /\ RedactionTests_inProgress));

Conception_inProgress => () (Conception_finished \/ Conception_inProgress );

RedactionDoc_inProgress => () (RedactionDoc_finished \/ RedactionDoc_inProgress );

Programmation_inProgress => () (Programmation_finished \/ Programmation_inProgress );

RedactionTests_inProgress => () (RedactionTests_finished \/ RedactionTests_inProgress );


```

FIGURE 11 – Propriété testée sur les invariants des processus

Elle a été vérifiée par selt :

```
MacBook-Pro-de-Chris:bin chris$ selt -p -S developpement.scn developpement.ktz -prelude developpement_invariants.ltl
Selt version 3.6.0 -- 07/06/20 -- LAAS/CNRS
ktz loaded, 26 states, 47 transitions
0.001s

- source developpement_invariants.ltl;
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
0.014s
```

FIGURE 12 – Résultats obtenus avec l'outil selt

14 Conclusion

Après de nombreuses semaines de travail, nous avons réussi a réaliser la majorité des transformations et ainsi d'obtenir un réseau de Petri qu'on peut executer sur le logiciel Tina. Ce projet nous a ainsi permis de nous familiariser avec de nouveaux outils associés à Eclipse et nous a permis de bien mettre en forme notre cours d'Ingénierie Dirigée par les Modèles.