BE IDM

Amandine Gravier, Alan Fresco, Louis Basset et Raphaël Giudice

Department Science du numérique
2023-2024

Contents

1 Introduction

Ce mini-projet consiste à produire une chaîne de vérification de modèles de processus Sim- plePDL dans le but de vérifier leur cohérence, en particulier pour savoir si le processus décrit peut se terminer ou non. Pour répondre à cette question, nous utilisons les outils de model-checking définis sur les réseaux de Petri au travers de la boîte à outils Tina. Il nous faudra donc traduire un modèle de processus en un réseau de Petri.

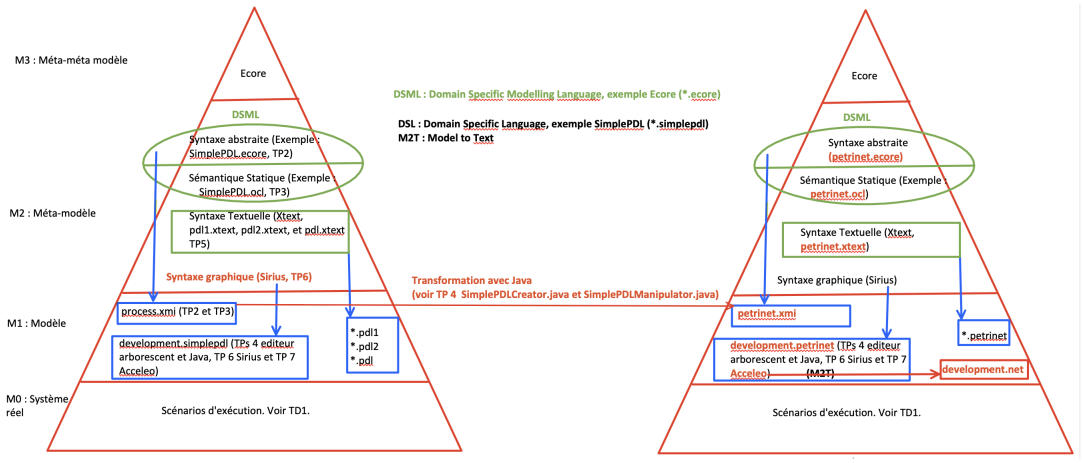

Figure 1: Modèle général

2 Les méta-modèles de SimplePLD et de Petrinet

2.1 Les méta-modèles ecore

Nous allons tout d'abord devoir définir les méta-modèles de SimplePLD et de Petrinet, qui seront la base de notre travail.

Pour SimplePDL, nous avons déjà une base, nous devons seulement ajouter l'utilisation des ressources. Pour cela, nous avons décidé d'ajouter une classe ressource et une classe DemandeRessource. La classe Ressource, comme les autres classes de ce diagramme, est un sous-type de ProcessElement. Cette classe a pour but de représenter les différentes ressources disponibles, en donnant leur nom et le nombre de ressources disponibles. La classe DemandeRessource est également un sous-type de ProcessElement. Cette classe représente le lien entre la ressource et la tâche qui demande cette ressource. Pour cela, elle possède un successeur (la tâche de type WorkDefinition) et un prédécsseur (la ressource de type Ressource). Ce lien possède également une valeur représentant le nombre de ressources nécessaires pour la tâche en question.

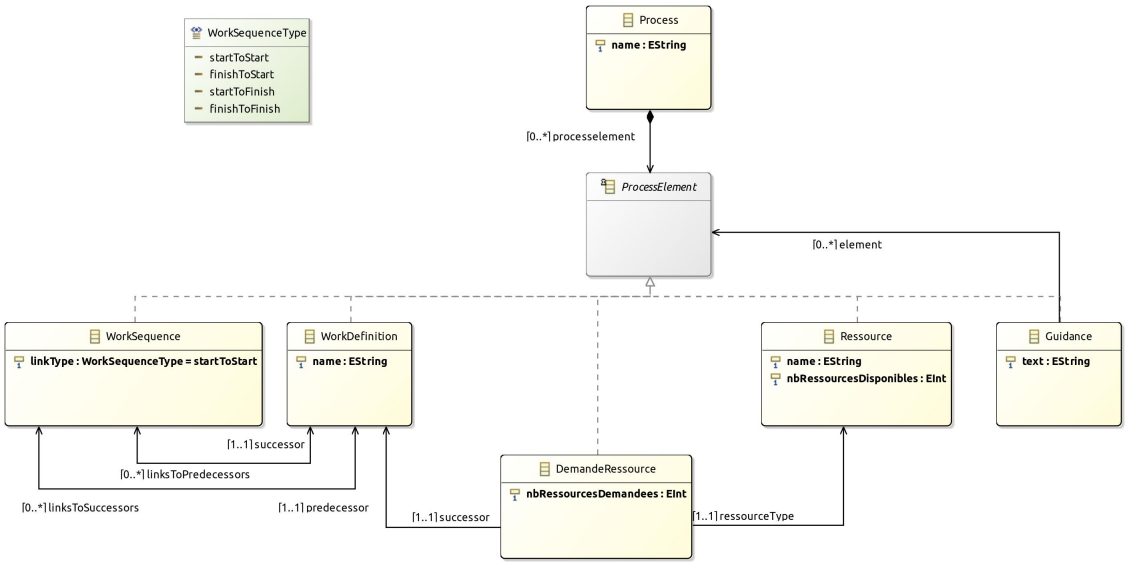

Figure 2: Méta-modèle SimplePDL

Pour PetriNet, nous sommes partis de zéro, en se basant sur un schéma similaire à celui de SimplePDL. Nous avons une classe PetriNet, qui représente le réseau de Pétri dans sa globalité. Ce réseau est composé de PetriNetElement, une autre classe. Ensuite, nous avons les classes Jetons, Arc et PetriArchi qui sont tous des sous-types de PetriNetElement. La classe Jetons permet de représenter les jetons dans le réseau de Pétri, en leur donnant un nom pour pouvoir les différencier plus facilement. La classe Arc permet de représenter les arcs du réseaux. Chaque arc a un poids et un nom, et a un predecesseur et un successeur, les deux de type PetriArchi. La classe PetriArchi, quant à elle, permet de regrouper sous un même type deux autres classes : Place et Transition. La classe Place représente les places, avec un nom et les jetons présents sur cette place. Enfin, la classe Transition représente les transitions, avec leurs noms.

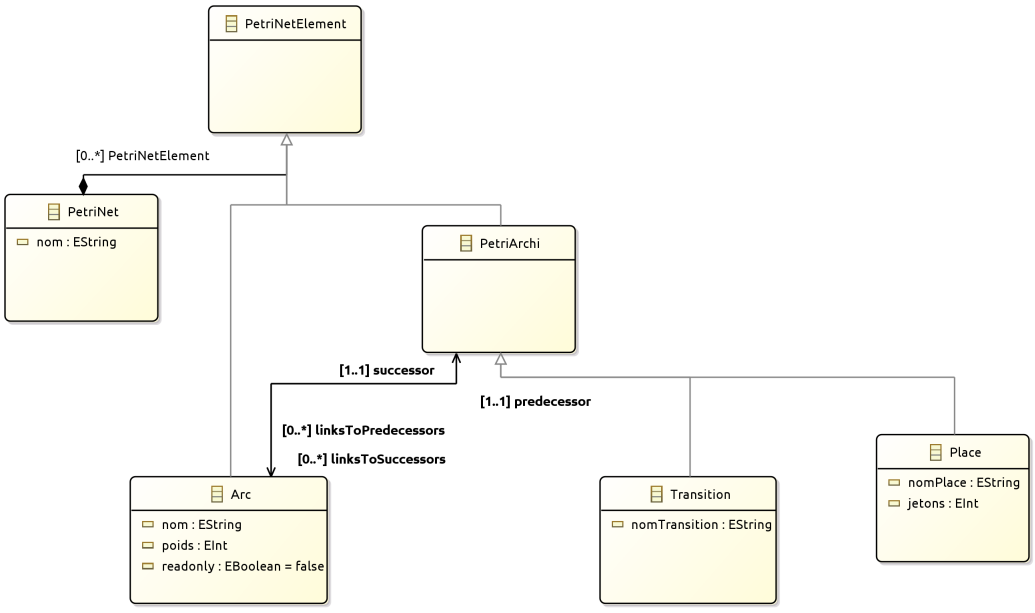

Figure 3: Méta-modèle PetriNet

2.2 Les contraintes OCL

Certaines contraintes ne peuvent pas être respectées et représentées seulement par le modèle Ecore, nous devons les ajouter dans un autre fichier. Ces contraintes seront ajoutées dans un fichier OCL. Pour SimplePDL, voici les contraintes que nous avons ajoutées :

- deux ressources ne peuvent pas avoir le même nom

Pour Petrinet, voici les contraintes que nous avons mises :

- il faut que le réseau ait au moins une place et une transition
- deux places ne peuvent pas avoir le même nom
- une transition doit avoir un prédécesseur et un successeur
- un arc doit relier une transition à une place ou une place à une transition

Pour vérifier que ces contraintes fonctionnent bien, nous avons définis des modèles qui respectent ces contraintes et des modèles qui ne respectent pas ces contraintes. ProcessOK et PetrinetOK valident tous les deux leurs modèles respectifs.

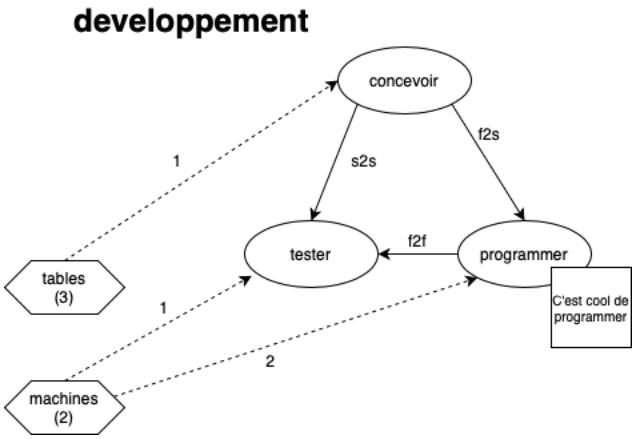

Figure 4: Process OK

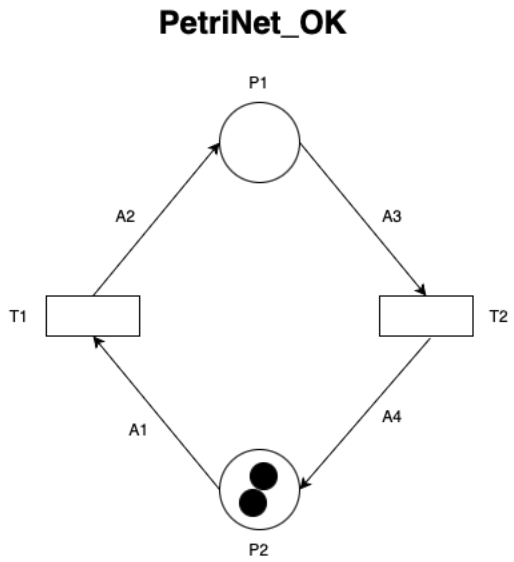

Figure 5: PetriNet OK

ProcessKO ne valide pas les contraintes à cause de :

- une transition est réflexive (notReflexive)
- le nom du process n'est pas bon (ValidName)
- 2 ressources ont le même nom (validRessourceName)
- le nom "te" est trop court (nameIsLongEnough)

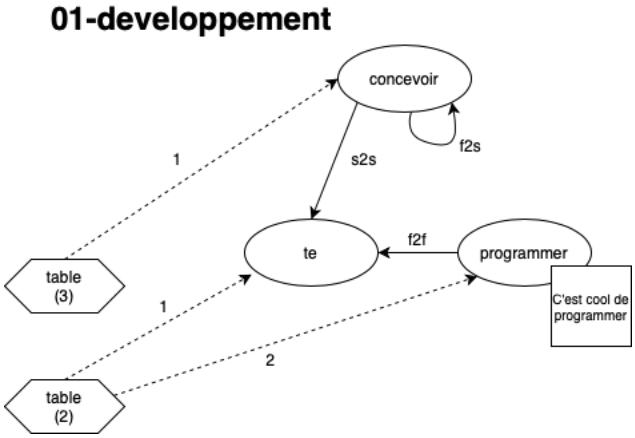

Figure 6: Process KO

PetrinetKO1 ne valide pas les contraintes à cause de :

- deux places ont le même nom (UniquePlaceNames)
- la transition T1 n'a pas de prédecesseur (ValidConnections)
- l'arc A1 n'est lié à rien (ValidArcConnections)

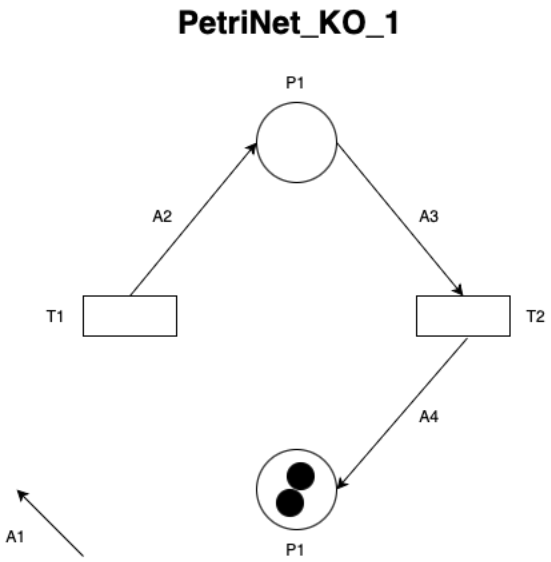

Figure 7: PetriNet KO 1

PetrinetKO2 ne valide pas les contraintes à cause de :

- il n'y a pas assez d'éléments (HasElements)

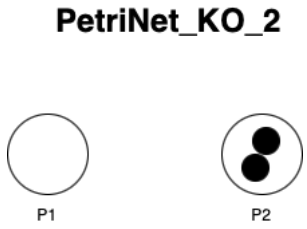

Figure 8: PetriNet KO 2

2.3 Syntaxe textuelle (Xtext)

Pour faire une syntaxe concrète textuelle, nous avons utilisé Xtext. Ceci permet de convertir un diagramme Ecore en un fichier texte, lisible différement.

2.4 Syntaxe graphique (Sirius)

Pour faire une syntaxe graphique, nous avons utilisé Sirius. Après avoir paramétré comme il faut Sirius, nous avons pu tester avec un de nos modèles pour voir le rendu. Voici ci-après le rendu avec le modèle ProcessOK.

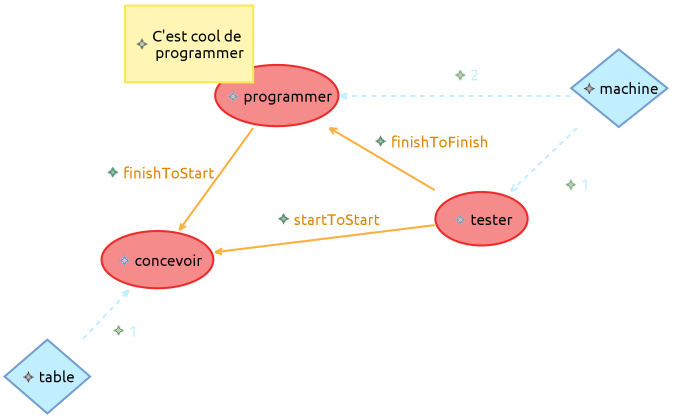

Figure 9: Syntaxe graphique de ProcessOK avec Sirius

3 La transformation M2M en Java

Nous avons réalisé une transformation de modèle à modèle en java. Ce fichier java permet de dire pour chaque élément d'un modèle en SimplePDL quels éléments vont devoir être crées pour créer le modèle en PetriNet équivalent, ainsi que préciser avec quels paramètres ces éléments doivent être crées.

Après avoir réussi à implanter correctement la transformation en java, nous avons pu tester notre code sur des exemples concrets, et s'assurer ainsi que notre code était correct et effectuait bien la transformation attendue.

4 Vérifier la terminaison des processus

L'objectif final étant de pouvoir vérifier la terminaison ou non d'un processus SimplePDL. Après avoir converti notre processus SimplePDL en réseaux de pétri PetriNet, il faut à présent que l'on puisse vérifier que ce réseau de pétri peut se terminer. Pour cela, nous allons utiliser l'outil Tina.

4.1 De PetriNet à Tina

Nous allons donc devoir convertir notre modèle PetriNet pour qu'il soit utilisable par l'outil Tina. Nous allons pour cela utiliser Acceleo. Ce fichier nous permet de convertir un réseau de Pétri Petrinet en un fichier lisible par l'outil Tina.

4.2 Les propriétés LTL

Nous n'avons pas eu le temps d'aborder cette dernière partie à cause d'un manque de temps.

5 Les difficultés rencontrées

Nous avons quelques difficultés durant ce projet, la principale étant qu'en avancant tout au long de notre projet, nous avons remarqué que nos modèles Ecore n'était pas parfaitement adapté. Ainsi, à plusieurs reprises, nous avons du modifié les fichiers SimplePLD.ecore et PetriNet.ecore. Le problème majeur ici est qu'à chaque que nous modifions ces fichiers, nous devions recommencer une grande partie des manipulations à zéro mais avec les nouveaux fichiers. Ces modifications nous ont donc fait perdre beaucoup de temps, nous empechant d'arriver au terme du projet.

6 Conclusion

Pour conclure, nous pouvons dire que nous avons réussi à mener ce projet à bien, en faisant toutes les étapes de la vérification de la terminaison d'un processus. Nous avons pu, pour mener à bien ce projet, manier différents outils tels que Eclipse ou Tina, ainsi qu'approfondir et mieux comprendre les concepts de modèles et méta-modèles vu en cours.