Génie des Logiciels et des Systèmes: Modèles de Processus

Thibault MEUNIER Matthieu PERRIER

28 octobre 2016

Introdution

L'objectif de ce BE est de mettre en place une nouvelle représentation de processus appelé simplePDL. L'utilisateur devra pouvoir éditer des processus rapidement et graphiquement, tout en vérifiant que les contraintes soient vérifiées. Pour cette vérification, on utilisera l'outil TINA, qui travaille sur des réseaux de Pétri, d'où le besoin de créer des outils de conversion de modèle à modàle, et de modèle à texte.

Sommaire

1 Les modèles simplePDL et PetriNet et leurs contraintes

1.1 Modèle PetriNet

1.1.1 Présentation

Un réseau de Pétri est un modèle permettant de modéliser et de vérifier le comportement de systèmes à événements discrets. Il est composé de places et de transitions reliées entre elles par de arcs. La Figure 1 représente la modélisation que nous avons implanté.

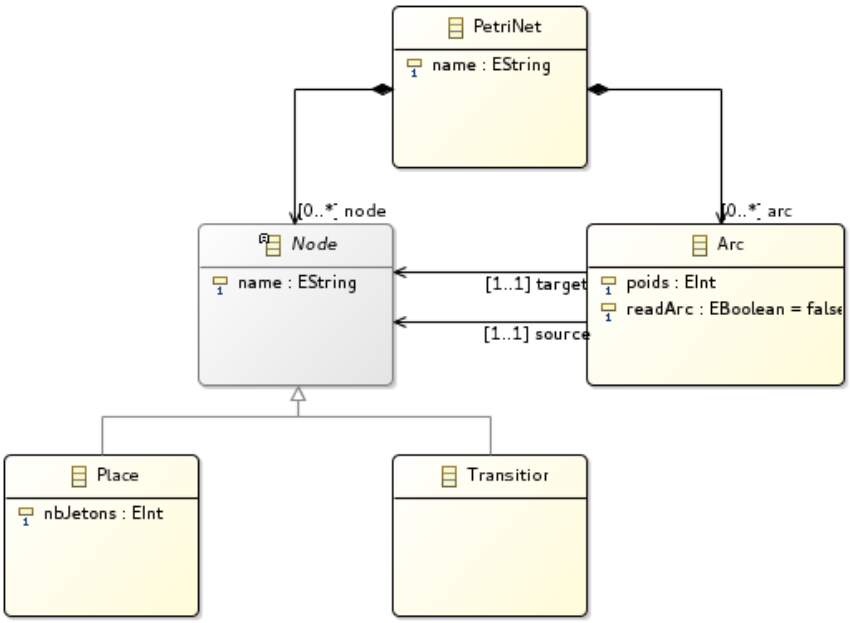

FIGURE 1 – Diagramme du métamodèle PetriNet

1.1.2 Contraintes

Un tel model permet la construction de modèles invcalides. Par exemple, on pourrait lier deux places avec un arc. Pour obtenir un modèle conforme au métamodèle réseau de Pétri, il a fallu le compléter avec les contraintes suivantes en OCL :

Contraintes sur les arcs :

sourceTypeDiffTarget Un arc ne peut pas relier deux nœuds de même type.

positive Un arc a un poid strictement positif

Contraintes sur les nœuds

positive Un nœud contient un nombre positif ou nul de jetons

1.2 Modèle simplePDL

1.2.1 Présentation

SimplePDL est un métamodèle de représentation de processus. Il défini le concept de processus (Process) composé d'activités (WorkDefinition) qui peuvent dépendre les une sur les autres (WorkSequence). Il définit également le concept de ressources (Ressource) et de leur consommations par les activités (RessourceSequence). Pour finir, on peut ajouter un commentaire (Guidance concernant une activité. La Figure 2 représente la modélisation que nous avons implanté.

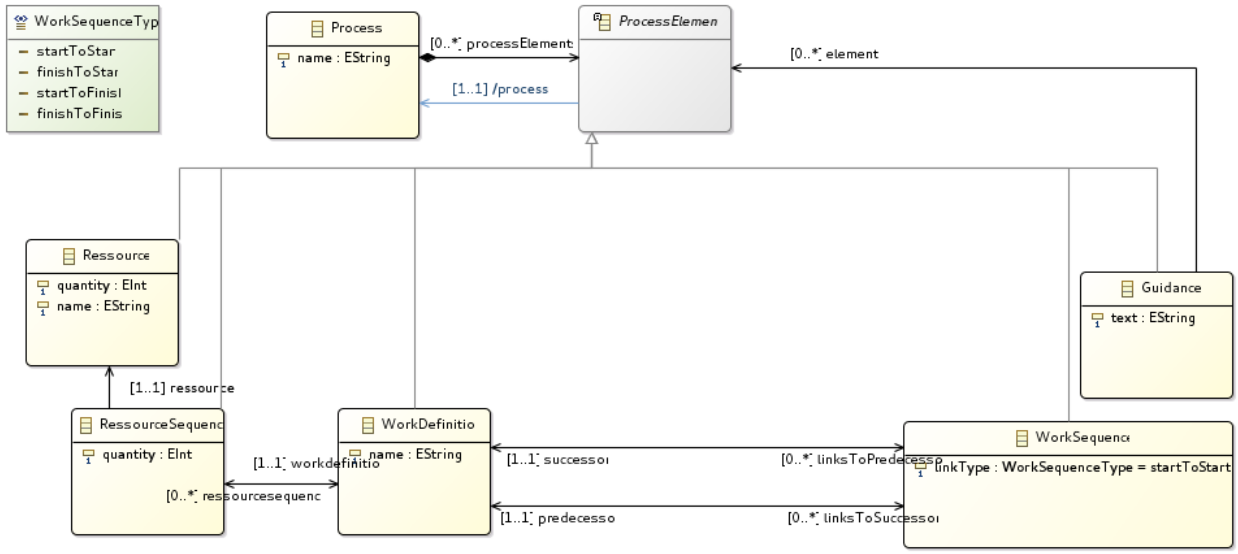

FIGURE 2 – Diagramme du métamodèle simplePDL

1.2.2 Contraintes

Ici encore, il faut rajouter les contraintes OCL suivantes pour obtenir un modèle conforme au métamodèle.

Contraintes sur Process :

nameForbidden Un Processus ne peut pas s'appeler 'Process'.

differentNames Deux processus ne peuvent pas avoir le même nom.

Contraintes sur WorkDefinition :

nameNotEmpty Le nom d'un WorkDefinition n'est pas vide.

Contraintes sur WorkSequence :

previousWDinSameProcess La Work sequence et sa WorkDefinition source sont dans le même processus.

nextWDinSameProcess Idem avec la WorkDefinition cible.

notReflexive Une WorkSequence ne boucle pas sur la même WorkDefinition.

Contraintes sur Ressource :

nameNotEmpty Le nom d'un WorkDefinition n'est pas vide.

quantityPositive Il y a au moins une instance de cette ressource.

Contraintes sur RessourceSequence :

quantityPositive Il faut au moins une instance de la ressource.

quantityAvailable Il existe assez d'instance de la ressource.

2 Transformations de modèle

2.1 Transformation de modèle à modèle

2.1.1 SimplePDL to PetriNet

Un Process peut être transformé en un PetriNet. Pour cela, chaque WorkDefinition peut être modélisée en un ensemble de places et transition comme schématisé sur la Figure 3. Puis, chaque WorkSequence correspond à un read-arc entre les nœuds appropriés. Enfin, une Ressource est une place et une RessourceSequence correspond à deux arcs : un pour la consommation et un pour la restitution des ressources.

Cette transformation à été implanté en Java, puis en ATL.

2.2 Transformation de modèle à texte

Quatre transformation de modèle à texte ont été implantés :
— PetriNet to dot
— PetriNet to TINA
— SimplePDL to dot
— SimplePDL to LTL

Ces transformations ont été implémentées en utilisant le language ATL. La mixité avec OCL n'a pas rendu cette transformation aisée, la syntaxe obtenue n'étant pas très lisible.

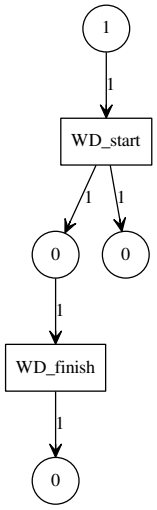

FIGURE 3 – Diagramme du métamodèle PetriNet

2.3 Transformation de texte à modèle

Il est nécessaire de définir une manière textelle de modélistaion des languages, afin de pouvoir rapidemment vérifier et importer une modélisation. Ainsi, le language textuel défini se doit concis et lisible. XText fourni des outils d'analyse textuels puissants intégrés à Eclipse. La principale difficultée a été la modélisation du language et les différents liens dynamiques entre les éléments.

3 L'éditeur graphique Sirius

Nous avons utilisé l'éditeur graphique Sirius pour pouvoir modifier des processus modélisés en SimplePDL graphiquement. Il y a la possibilité d'ajouter dynamiquement tous les éléments modélisés. La principale difficultées rencontrée lors de cette étape a été d'implanter une coloration conditionelle des fleches représentant les WorkSequences en fonction de leur type. En effet, la syntaxe n'est pas évidente et non décrite dans les TPs. La Figure 4 montre la vue obtenue pour le process Developpement.

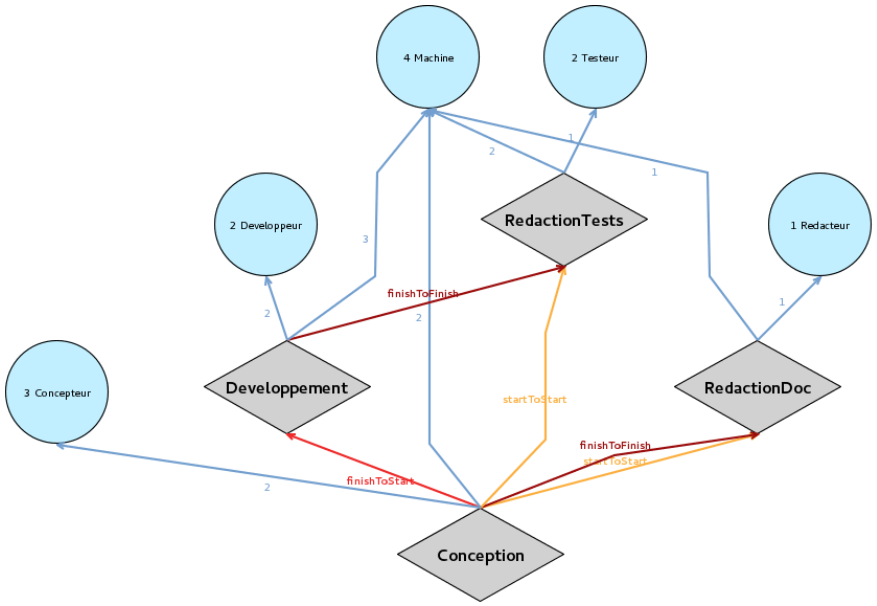

FIGURE 4 – Représentation du process Develeppement dans l'éditeur graphique

4 Exemples et tests

Nous avons essayer de tester tous les invariants de nos modèles séparémment, puis nous avons tester le tout avec des exemple correct le plus concrets et distincts possible.