Rapport d'IDM

MEGEMONT Clement, SEGHROUCHNI Anas

Département Sciences du Numérique - Deuxième année
2022-2023

Taches à réaliser

0.1 Compléter le métamodèle SimplePDL pour prendre en compte les ressources.

Afin de prendre en compte les ressources nous avons créer deux nouvelles classes : **Ressource** et **RessourceUtile**.
La première est une sous classe de **ProcessElement** et possède 2 attributs : un *name* et une *quantité*.
La seconde nous permet de lier **Workdefinition** avec **Ressource**, elle représente la quantité d'une ressource nécessaire à la réalisation d'une activité. Elle est liée par composition à **Workdefinition** car si l'activité n'existe pas alors la ressource utile non plus. Elle est aussi liée à **Ressource** afin de pouvoir savoir de quelle ressource il s'agit.

0.2 Définir le métamodèle PetriNet

En s'inspirant du métamodèle des processus, on a décidé de créer une classe "générale" nommée **ReseauPetri** ayant qu'un attribut : *name*. Un réseau de Petri est composé soit de noeuds soit d'arcs, c'est pour cela que nous avons crée deux classes : **Noeud** et **Arc** possédant respectivement l'attribue *name* et *Poids*. Chaque Arc possède une unique cible et une source de la classe **Noeud** et chaque noeud peuvent avoir plusieurs ou aucun arcs sortants et entrants. Les classes **Place** et **Transition** sont toute deux des réalisation de **Noeud**. La classe **Place** possède un attribut en plus : *nbJeton*. Pour finir on a **ReadArc** qui est une réalisation d'**Arc**.

0.3 Développer un éditeur graphique SimplePDL pour saisir graphiquement un modèle de processus, y compris les ressources.

Afin de développer l'éditeur graphique nous sommes parti du travail réalisé en séance de TP, nous avons définie des composants représentant la classe **Ressource** et des liens de **Ressource** vers les activités qui en ont besoin. Sur chaque lien, on a affiché la quantité utilisé par l'activité (**RessourceUtile**). Nous n'avons cependant pas réussi à pouvoir créer un lien entre un commentaire et une activité à l'aide de la palette. Cependant, il est possible de créer une activité représenter par une ellipse rose, des ressources (rectangle gris) et de lier les ressources aux activités.

0.4 Définir les contraintes OCL pour capturer les contraintes qui n'ont pu l'être par les métamodèles (SimplePDL et PetriNet).

De nombreuses contraintes OCL sont similaires pour Réseau de Petri et SimplePDL :

- Nom bien défini : Les noms des réseaux de Petri et celui des Process devaient commencé par une lettre.
- Non Reflexive : Les WorkSequences comme les arcs ne peuvent pas avoir le même elélément en tant source et cible.
- Nom unique : Il ne peut pas y avoir 2 éléments possédant le même nom (Place, Transition,c WorkDefinition).
- Quantité positive : Le nombre de jeton des places et les poids des arc doivent être positifs tout comme la quantité totale de ressource (**Ressource**) et la quantité utilisé par l'activité (**RessourceUtile**).

De plus certaines contraintes sont propres aux réseaux de Pétri comme celle qui indique qu'un arc doit aller d'une transition vers une place ou inversement. Il y a aussi, concernant les processus, une contraintes qui indique que la quantité utilisé par une activité ne peut pas dépasser la quantité totale de la ressource.

0.5 Donner une syntaxe concrète textuelle de SimplePDL avec Xtext.

Nous sommes parti du travail fait en TP, nous avons rajouter **Ressource** au ProcessElements. Ainsi pour créer un *process* il faudra y ajouter les ressources utiles aux activités. Et aussi pour la création d'une **Workdefinition** il sera nésséraire de rajouter entre accolade les ressources utiles. Ces dernières sont créer en indiquant le nombre nécessaire à l'activité et le type de ressource.

0.6 Définir des transformations SimplePDL vers PetriNet

Dans cette section nous devions réaliser deux transformations de SimplePDL vers PetriNet. L'une en Java/EMF et l'autre en utilisant ATL. Le principe de cette transformation reste le même quelque soit le langage. Chaque WorkDefinition était représentant dans le réseau de Petri par (en remplaçant "A1" par le nom de l'activité) :

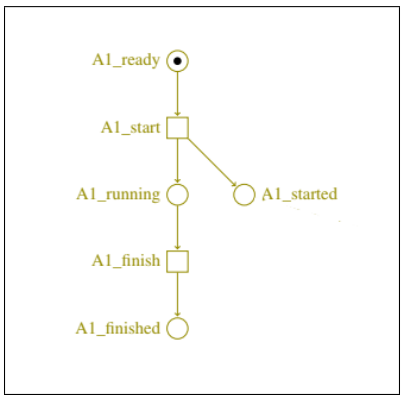

Figure 1: Représentation d'une WorkDefinition (A1) dans un réseau de Petri

Selon le type des WorkSequence (finish2start, start2finish...) les arcs liant les différentes représentations des WorkDefinitions ne sont pas les mêmes. Ils débutaient soit de la place `A1_finished` pour finish2start et start2finish soit de `A1_started` dans les autres cas. Quant à leur cible, c'est soit la transition `A1_start` pour finish2start et start2start soit `A1_finish` pour les autres cas. Voici un exemple pour 2 WorkDefinitions liées par un start2finish.

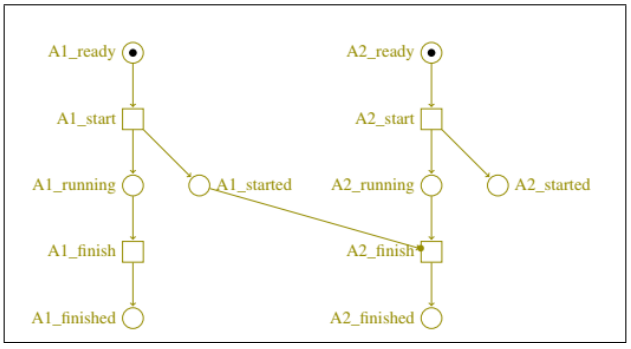

Figure 2: Représentation de 2 WorkDefinitions (A1 et A2) liées par un start2finish dans un réseau de Petri

Pour les ressources, chaque ressource sera représentée par une place dont le nombre de jeton sera la quantité totale de cette ressource. On a aussi crée un arc ayant pour source la place ressource, pour cible la transition `A_start` et pour poids la quantité utile à l'activité. Et bien évidemment

afin de relâcher la ressource à la fin d'une activité, on a créé un arc entre `A_finish` et la place ressource avec le même poids que l'arc précèdent.

0.7 Définir une transformation PetriNet vers Tina en utilisant Acceleo.

Nous avons construit une transformation de modèle à texte, les principales difficultés étaient dans la compréhension d'Acceleo.