Rapport

Mini-projet
Chaine de vérification de modelés de processus

Briag Rehel
Alexia Huc-Lhuillery

Métamodélisation

Rajout des ressources au métamodèle de SimplePDL:

Pour rajouter les ressources nous avons créé deux nouvelles classes : Ressource et RessourcesNecessaires.
Ressource fait partie des éléments du processus, il hérite donc de processElement. Une ressource a un nom (attribut name: String) et une quantité (attribut quantite: int).
RessourcesNecessaires caractérise la dépendance entre WorkDefinition et Ressource.
RessourcesNecessaires correspond pour Ressource aux quantités de ressources utilisées et pour RessourcesNecessaires Ressource correspond à la ressource utilisée.
Le lien de WorkDefinition vers RessourcesNecessaires est un lien de composition car il n'y a pas de RessourcesNecessaires sans WorkDefinition (car RessourcesNecessaires caractérise les ressources nécessaires à une WorkDefinition). Et une Ressource nécessaire n'est associée qu'à une WorkDefinition.

Métamodèle des réseaux de pétri:

Pour construire le métamodèle des réseaux de pétri (appelé petrinet) on s'est inspiré de celui de SimplePDL.
On crée une première classe ReseauPetri qui correspond à un réseau de pétri. Un réseau de pétri contient des éléments qui lui sont propres. La classe ReseauPetri est donc liée par composition à une classe petriElement. Les éléments d'un réseau de pétri sont soit des arcs, soit des transitions, soit des places. Une transition et une place ont les mêmes propriétés mais une place peut contenir des jetons. On abstrait ceci grâce à une classe abstraite dont place et transition hérite. Cette classe contient un attribut nom car place et attribut ont toutes les deux un attribut nom. De plus comme elles sont liées de la même manière à Arc c'est boite qui abstrait ce lien.
Le lien entre boite et Arc est similaire à celui entre WorkSequence et WorkDefinition.
Enfin on rajoute la classe ReadArc qui est un type particulier d'arc et qui hérite donc de arc.

Contraintes Ocl:

Contraintes portant sur les ressources :

Les quantités de ressources et de ressources nécessaires doivent être positives et les quantités de RessourcesNecessaires doivent être inférieures aux quantités de ressources de la ressource associée.

Contraintes portant sur les réseaux de pétri :

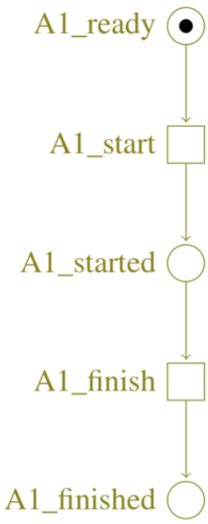
Tous les éléments de même type du réseau de pétri doivent avoir des noms différents. Le nombre de jeton des arcs et des places doivent êtres positifs.

Enfin le métamodèle Ecore ne permet de dire que les arcs vont d'une transition vers une place ou l'inverse mais pas vers deux éléments de même type. On a donc défini une contrainte Ocl pour cette propriété.

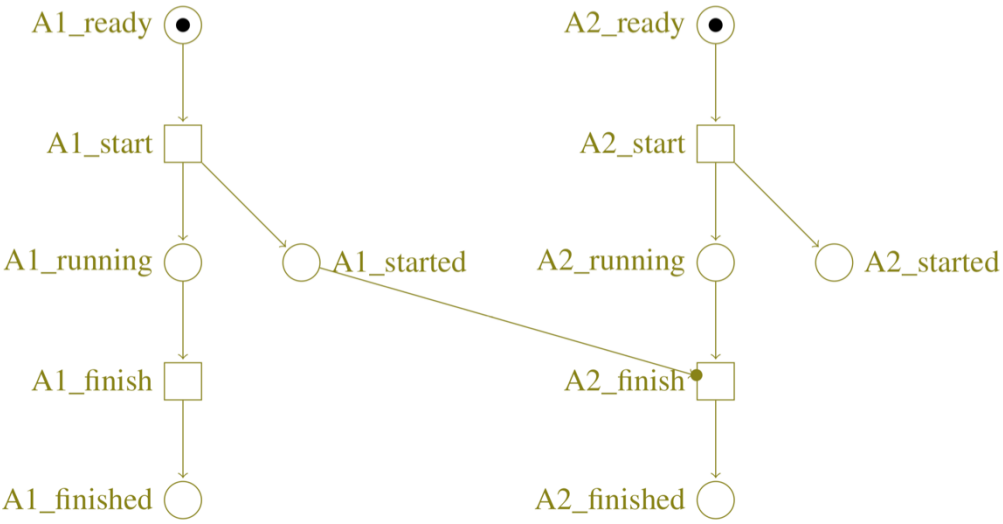
Transformation d'un modèle de processus en réseau de pétri en java.

Choix de réalisation:

Une activité (WorkDefinition) se traduit en réseau pétri par la structure suivante:



Pour ajouter les WorkSequences on doit créer une place supplémentaire car si l'activité change d'état on veut garder en mémoire le fait que l'on a passé l'état souhaité.
On obtient la structure suivante dont la transition de départ dépend de la condition (startToStart, startToFinish ...):



Pour ajouter les ressources on va créer une place ressource pour chaque ressource dont le nombre de jeton dépend de la quantité de cette ressource.

Les ressources nécessaires vont être représentées sous pétri par un arc qui pointe vers la transition Start de l'activité concernée.
Pour redonner les ressources on a besoin d'un arc de la transition finish vers la ressource.
La place ready est à un nombre de jeton de 1.
Enfin on utilise une Hashmap pour faciliter la recherche des éléments du réseau de pétri déjà crée.

Transformation modèle a texte:
On a dû faire une ligne de commande longue pour faire les transitions car Acceleo est sensible au changement de ligne.

Sirius:
On a choisi de représenter les ressources par des ellipses bleues. Pour faire l'outil de création d'une Ressources on procède de la même façon que pour une WorkDefinition.
Pour l'outil de création d'un lien RessourcesNecessaires on procède de la même façon que pour l'outil de création d'une WorkSequence sauf qu'on se place dans le contexte de la source ([source/]) car RessourcesNecessaires est contenue dans WorkDefinition.

Validation avec LTL :
Pour valider la transformation, on définit un invariant pour vérifier que les activités ne peuvent pas être dans deux états différents en même temps, qui peut être appliqué sur le fichier petrinet créé après la transformation d'un fichier simplePDL. La propriété créée est toujours vraie.
De plus pour vérifier la terminaison, on peut vérifier que toutes les activités sont bien dans l'état finished à la fin de l'évolution du processus. La première propriété est vraie, elle signifie que le processus est dans un état final. La dernière est fausse, elle signifie que le processus ne peut jamais finir, ce qui permet d'obtenir via le message d'erreur les transitions à passer afin d'arriver à l'état final du processus.