Chaîne de vérification de modèles de processus

Ce BE consiste à produire une chaîne de vérification de modèles de processus SimplePDL 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.

Les TP ont présenté les outils qui devront être utilisés pour réaliser ce BE. Chaque TP traite un aspect développé dans le BE. Le TP est une introduction qui devra être complétée en consultant au moins la documentation des outils utilisés.

Le BE correspond à ce qui est fait dans les TP avec une petite extension (les ressources) de manière à vérifier que les TP ont bien été compris et assimilés.

1 Objectifs du BE

Ce BE consiste pour l'essentiel à définir la chaîne de vérification de modèles de processus (leur description est donnée en section 1.1) sur une version simplifiée des modèles de processus. Ce travail a été commencé en TP. Les principales étapes sont les suivantes :

1. Définition des métamodèles avec Ecore.
2. Définition de la sémantique statique avec OCL (Complete OCL).
3. Utilisation de l'infrastructure fournie par EMF pour manipuler les modèles.
4. Définition de transformations modèle à texte (M2T) avec Acceleo, par exemple pour engendrer la syntaxe attendue par Tina à partir d'un modèle de réseau de Petri ou engendrer les propriétés LTL à partir d'un modèle de processus.
5. ~~Définition de syntaxes concrètes textuelles avec Xtext.~~
6. ~~Définition de syntaxes concrètes graphiques avec Sirius.~~
7. Définition d'une transformation de modèle à modèle (M2M) ~~avec Java et~~ avec ATL.

Le travail fait en TP sera complété par ~~la validation de la chaîne de transformation (section 1.2) et~~ l'ajout de ressources au langage de description de processus (section 1.3).

1.1 Description des modèles de processus

Le modèle de procédé utilisé est inspiré de SPEM [1], norme de l'OMG. Nous donnons entre parenthèses le vocabulaire utilisé par cette norme. Dans un premier temps, nous nous intéressons à des processus simples composés seulement d'activités (WorkDefinition) et de dépendances (WorkSequence). La figure 1 donne un exemple de processus qui comprend quatre activités :

1. `http://www.omg.org/spec/SPEM/2.0/`

Conception, RédactionDoc, Développement et RédactionTests. Les activités sont représentées par des ellipses. Les arcs entre activités représentent les dépendances. Une étiquette permet de préciser la nature de la dépendance sous la forme « étatToAction » qui précise l'état qui doit être atteint par l'activité source pour réaliser l'action sur l'activité cible. Par exemple, on ne peut commencer RédactionTests que si Conception est commencée. On ne peut commencer Développement que si Conception est terminée. On ne peut terminer RédactionTests que si Développement est terminé.

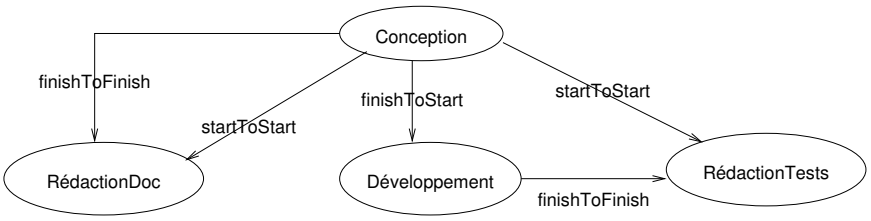

FIGURE 1 – Exemple de modèle de procédé

1.2 Validation de la transformation SimplePDL2PetriNet

Comme pour tout programme écrit, il est important de valider la transformation de modèle. Afin de valider la transformation SimplePDL vers PetriNet, une possibilité est de vérifier que les invariants sur le modèle de processus sont préservés sur le modèle de réseau de Petri correspondant. Ces invariants sont appelés *propriétés de sûreté*. En voici quelques exemples :

— chaque activité est soit non commencée, soit en cours, soit terminée ;

— une activité terminée n'évolue plus.

On peut alors écrire une transformation modèle à texte qui traduit ces propriétés de sûreté sur le modèle de Petri. L'outil *selt* permettra alors de vérifier si elles sont effectivement satisfaites sur le modèle de réseau de Petri. Si ce n'est pas le cas, c'est que la traduction contient une erreur ou que l'invariant n'en est pas un !

1.3 Ajout des ressources

Pour réaliser une activité, des ressources peuvent être nécessaires. Une ressource peut correspondre à un acteur humain, un outil ou tout autre élément jouant un rôle dans le déroulement de l'activité. Ici, nous nous intéressons simplement aux types de ressources nécessaires et au nombre d'occurrences d'un type de ressource. Par exemple, il peut y avoir deux développeurs, trois machines, un bloc-note, etc. Un type de ressource sera seulement caractérisé par son nom et la quantité d'occurrences de celle-ci.

Pour pouvoir être réalisée, une activité peut nécessiter plusieurs ressources (éventuellement aucune). Par exemple, l'activité *RedactionTest* nécessite un testeur (une occurrence de la ressource de type Testeur) et deux machines (deux occurrences de la ressource Machine). Les occurrences de ressources nécessaires à la réalisation d'une activité sont prises au moment de son démarrage et rendues à la fin de son exécution. Bien entendu, une même occurrence de ressource

ne peut pas être utilisée simultanément par plusieurs activités. Les types de ressource et leur nombre d'occurrences sont définis en même temps que le procédé lui-même.

Voici une affectation possible de ressources pour le processus décrit à la figure 1. Une ligne correspond à un type de ressource. Elle indique la quantité totale de la ressource ainsi que le nombre d'occurrences de cette ressource nécessaire à la réalisation d'une activité.

	Quantité	Conception	RédactionDoc	Développement	RédactionTest
concepteur	3	2			
développeur	2			2	
machine	4	2	1	3	2
rédacteur	1		1		
testeur	2				1

2 Déroulement du BE

Ce BE est un travail en binôme.
Les techniques mises en œuvre doivent être celles présentées dans le module de IDM.

2.1 Tâches à réaliser

Pour chaque partie, voici les tâches à réaliser et les documents à rendre.

T_1 Compléter le métamodèle SimplePDL pour prendre en compte les ressources.

T_2 Définir le métamodèle PetriNet.

T_3 ~~Développer un éditeur graphique SimplePDL pour saisir graphiquement un modèle de processus, y compris les ressources.~~

T_4 Définir les contraintes OCL pour capturer les contraintes qui n'ont pu l'être par les métamodèles (SimplePDL et PetriNet).

T_5 ~~Donner une syntaxe concrète textuelle de SimplePDL avec Xtext.~~

T_6 ~~Définir une transformation SimplePDL vers PetriNet en utilisant EMF/Java.~~

T_7 Définir une transformation SimplePDL vers PetriNet en utilisant ATL.

T_8 Valider la transformation SimplePDL vers PetriNet sur plusieurs exemples.

T_9 Définir une transformation PetriNet vers Tina en utilisant Acceleo. On ne traitera pas les aspects temporels.

T_{10} ~~Engendrer les propriétés LTL permettant de vérifier la terminaison d'un processus et les appliquer sur différents modèles de processus.~~

T_{11} ~~Engendrer les propriétés LTL correspondant aux invariants de SimplePDL pour valider la transformation écrite (voir section 1.2).~~

2.2 Documents à rendre

D_1 Les métamodèles SimplePDL et PetriNet (ainsi que des images de ces métamodèles).

D_2 Les fichiers de contraintes OCL associés à ces métamodèles, avec des exemples (et contre-exemples) qui montrent la pertinence de ces contraintes.

D_3 ~~Le code Java de la transformation modèle à modèle.~~

D_4 Le code ATL de la transformation modèle à modèle.

D_5 Le code Acceleo des transformations modèle à texte.

D_6 ~~Les modèles Sirius décrivant l'éditeur graphique pour SimplePDL.~~

D_7 ~~Le modèle Xtext décrivant la syntaxe concrète textuelle de SimplePDL.~~

D_8 Des exemples de modèles de processus (en expliquant leur intérêt).

D_9 Un document concis (rapport) qui explique le travail réalisé. Attention, c'est ce document qui servira de point d'entrée pour lire les éléments rendus.