Chaîne de vérification de modèles de processus

Nous traitons un exemple de langages dédiés suivant l'approche MDE (Model Driven Engineering). Nous nous focalisons sur un langage appelé SimplePDL qui décrit le processus de développement logiciel.

Ce mini-projet 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 présenteront les outils qui devront être utilisés pour réaliser ce mini-projet. Chaque TP traite un aspect développé dans le mini-projet. Le TP est une introduction qui devra être complétée en consultant au moins la documentation des outils utilisés.

Le mini-projet 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 mini-projet

Ce mini-projet consiste pour l'essentiel à définir la chaîne de vérification de modèles de processus (dont la description est donnée en section **??**). Ce travail sera fait en TP, les principales étapes étant les suivantes :

1. Définition des métamodèles avec Ecore. Niveau M2 de la pyramide OMG.
2. Définition de la sémantique statique avec OCL (Complete OCL).
3. Utilisation de l'infrastructure fournie par EMF (Eclipse Modeling Framework) 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 d'une transformation de modèle à modèle (M2M) avec EMF/Java ou ATL.
6. Définition de syntaxes concrètes textuelles avec Xtext et manipulation de la transformation texte à modèle (T2M).
7. Définition de syntaxes concrètes graphiques avec Sirius.

Ce travail fait en TP sera complété par la validation de la chaîne de transformation (section **??**) et l'ajout de ressources au langage (SimplePDL) de description de processus (section **??**).

1.1 Description des modèles de SimplePDL

Le modèle (niveau M1) 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 **??** donne un exemple de processus qui comprend quatre activités : 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.

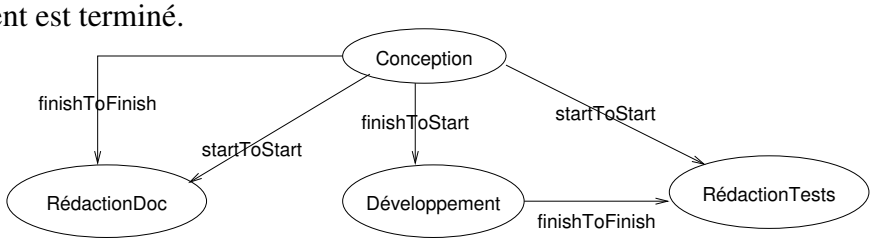

FIGURE 1 – Exemple de modèle de procédé

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.

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

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 **??**. 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 mini-projet

Ce mini-projet est un travail réalise par groupe de 4 étudiants.

Les techniques mises en œuvre doivent être celles présentées dans le module Ingénierie Dirigée par les Modèles (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.ecore" pour prendre en compte les "ressources".

T_2 Définir le métamodèle "PetriNet.ecore".

T_3 Développer un éditeur graphique avec l'outil Sirius pour SimplePDL (voir TP06 [2]) pour saisir graphiquement un modèle de processus, y compris les ressources.

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

T_5 Donner une syntaxe concrète textuelle de SimplePDL avec Xtext. Un fichier avec extension ".xtext" doit être fourni (voir TP05).

T_6 Définir une transformation modèle à modèle (M2M) "SimplePDL.xmi" vers PetriNet.xmi en utilisant EMF/Java. Voir TP04 où vous deux classe Java "SimplePDLCreator.java" (elle permet de créer un modèle avec extension ".xmi" conforme avec un méta-modèle) et "SimplePDLManipulator.java" (Elle permet d'afficher un modèle sur la sortie standard). Ces deux classes peuvent vous inspirer.

2. `http://ouederni.perso.enseeiht.fr/teachingN7IDMR.html`

T_7 Valider la transformation SimplePDL vers PetriNet en faisant des tests. Ceci pourrait être fait en validant (clique droit) sur le fichier ".xmi" obtenu en résultat de la tâche 6.

T_8 Définir une transformation PetriNet vers Tina en utilisant Acceleo. Ici un fichier "PetriNet2Tina.mtl" devrait être donné. L'extension "mtl" est "model to langage" et cela traduit la transformation "Model to Text (M2T)".

T_9 Engendrer les propriétés LTL (Linear Tomporal Logic) permettant de vérifier la "terminaison d'un processus" (Voir la correction de TD1 pour des exemples LTL) et les appliquer sur différents modèles de processus. Ici un fichier "toLTL.mtl" devrait être fourni et qui consiste à exprimer les propriétés de terminaison (les états finaux sont atteints, voir correction de TD1).

T_{10} Engendrer les propriétés LTL correspondant aux invariants de SimplePDL pour valider la transformation écrite (voir section **??**).

2.2 Documents à rendre

Les consignes pour rendre les documents suivants seront données sur la page du module (les documents seront rendus via moodle).

La date limite pour rendre ces documents sera affichée sur moodle (mais il est conseillé de les faire et de les déposer au fur et à mesure de l'avancement des TP). Ces documents constituent un complément des TPs.

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

D_2 Les fichiers de contraintes OCL associés à ces métamodèles, avec des exemples valident (cad des fichiers ".xmi" valides en considérant OCL), et contre-exemples (cad des fichiers ".xmi" non valides en considérant OCL) qui montrent la pertinence de ces contraintes.

D_3 Le code Java de la transformation modèle à modèle (Tâche T6).

D_4 Le code Acceleo des transformations modèle à texte (Tâche T8).

D_5 Les modèles Sirius décrivant l'éditeur graphique pour SimplePDL (Tâche T3).

D_6 Le modèle Xtext décrivant la syntaxe concrète textuelle de SimplePDL (Tâche T5).

D_7 Des exemples de modèles de processus (en expliquant leur intérêt) (Tâche T7 en ".xmi" ou bien ".simplePDL" et ".PetriNet").

D_8 SimplePDL-finish.mtl : transformation M2T qui engendre les propriétés LTL qui vérifie la terminaison d ?un processus.

D_9 SimplePDL-invariants.mtl (extension de l'item a- voir ci-dessus) : transformation M2T qui engendre les propriétés LTL pour valider la correction de la transformation (T10 du sujet) des modèles de processus vers les réseaux de Petri.

D_{10} **Optionnel** : Un document concis (rapport) qui explique le travail réalisé. c'est un document qui servira de point d'entrée pour lire les éléments rendus, les difficultés rencontrées et les leçons appris.