

Chaîne de vérification de modèles de processus

Attention : Des informations complémentaires, en particulier concernant les échéances, les documents à rendre, l'évaluation... seront données sur le site du module.

Ce projet consiste à produire une chaîne de vérification de modèle de processus SimplePDL. Nous nous intéressons à des modèles de processus composés d'activités et de dépendances entre ces activités. Nous souhaitons pouvoir vérifier la cohérence d'un tel modèle de processus, en particulier 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 outil Tina. Il nous faudra donc traduire un modèle de processus en un réseau de Petri.

Les TP seront utilisés pour présenter les outils qui devront être utilisés pour réaliser ce projet. Chaque TP traite un aspect développé dans le projet. Dans la mesure où nous adoptons une approche projet, le TP reste une introduction qui devra être complétée en consultant au moins la documentation des outils utilisés.

La première partie du 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. La deuxième partie correspond à définir une extension plus ambitieuse des modèles de processus et de modifier la chaîne de vérification pour la prendre en compte.

1 Partie I

Cette première partie consiste pour l'essentiel à définir la chaîne de vérification de modèle de processus (leur description est donnée en section 1.1) sur une version simplifiée des modèles de processus. Ce travail sera fait en TP, les principales étapes étant les suivantes :

Les principales étapes (détaillées en TP) sont les suivantes :

1. Définition des métamodèles avec Ecore,
2. Définition de la sémantique statique avec OCL,
3. Utilisation de l'infrastructure fournie par EMF pour manipuler les modèles,
4. Définition de transformation 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 Obeo Designer,
7. Définition de transformation de modèle à modèle (M2M) avec ATL.

Ce 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 SPEM¹, 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 : Conception, RédactionDoc, Développement et RédactionTests. Les activités sont représentées par des ovales. 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.

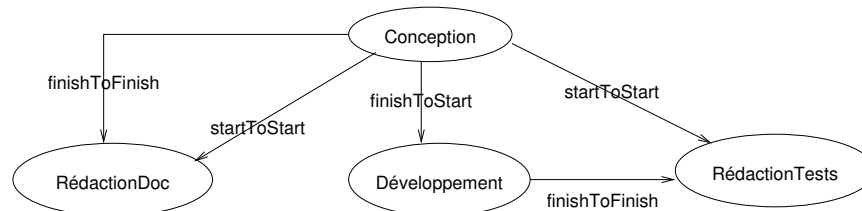


FIGURE 1 – Exemple de modèle de procédé

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 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 Partie II

Dans cette deuxième partie, on propose deux extensions du modèle de processus. Il s'agit d'en choisir une et d'adapter la chaîne de vérification pour la prendre en compte.

2.1 Décomposition d'une activité

Nous envisageons maintenant la possibilité de décomposer une activité. À l'image d'un processus, une activité peut contenir d'autres activités que nous appellerons sous-activités. Bien entendu, une sous-activité peut également être décomposée.

Une sous-activité ne peut utiliser que les ressources réservées par l'activité englobante.

Une activité composée est considérée :

- commencée quand ses ressources ont été allouées ;
- terminée quand toutes ses sous-activités sont terminées (et les ressources libérées).

2.2 Ressources alternatives

Pour réaliser une activité, elle doit disposer d'un certain nombre de ressources. En fait, plusieurs configurations de ressources peuvent permettre de réaliser une activité. Par exemple, rédiger les tests peut nécessiter soit un testeur et deux machines, soit un analyste et un générateur de test et une machine. Si l'un de ces deux ensembles de ressources est disponible, il est utilisé par l'activité qui peut alors commencer.

3 Déroulement du projet

Ce projet est un travail individuel. Les techniques mises en œuvre doivent être celles présentées dans le module de GLS.

3.1 Tâches à réaliser

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

- T₁ Compléter le métamodèle SimplePDL pour prendre en compte les ressources.
- T₂ Définir le métamodèle PetriNet.
- T₃ Développer un éditeur graphique SimplePDL pour saisir graphiquement un modèle de processus, y compris les ressources.
- T₄ Définir les contraintes OCL pour capturer les contraintes qui n'ont pu l'être par les métamodèle (SimplePDL et PetriNet).
- T₅ Donner une syntaxe concrète textuelle de SimplePDL et l'outiller avec Xtext.
- T₆ Définir une transformation SimplePDL vers PetriNet en utilisant ATL.
- T₇ Valider la transformation SimplePDL vers PetriNet.
- T₈ Définir une transformation PetriNet vers Tina en utilisant Acceleo.
- T₉ Engendrer les propriétés LTL permettant de vérifier la terminaison d'un processus et les appliquer sur différents modèles de processus.
- T₁₀ Engendrer les propriétés LTL correspondant aux invariants de SimplePDL pour valider la transformation écrite (voir section 1.2).

3.2 Documents à rendre

Les documents seront rendus sur SVN aux dates indiquées sur le site Web du module. L'activité de votre dépôt SVN pourra être prise en compte dans la notation.

- D₁ Les métamodèles SimplePDL.ecore et PetriNet.ecore.
- D₂ Les fichiers de contraintes OCL associés à ces métamodèles.
- D₃ Le code ATL des transformations modèle à modèle.
- D₄ Le code Acceleo des transformations modèle à texte.
- D₅ Les modèles Obeo Designer décrivant l'éditeur graphique pour SimplePDL.
- D₆ Le modèle Xtext décrivant la syntaxe concrète textuelle de SimplePDL.
- D₇ Des exemples de modèles de processus (en expliquant leur utilité).
- D₈ 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.