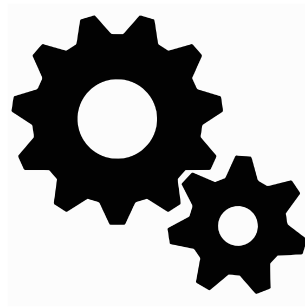


ENSEEIHT : GÉNIE DU LOGICIEL ET DES SYSTÈMES (GLS)

RAPPORT DE PROJET

GLS : Rapport de projet



Matthieu Pizenberg

20 décembre 2013

Table des matières

1	Les métamodèles SimplePDL et PetriNet	2
1.1	SimplePDL	2
1.2	PetriNet	4
1.3	Exemples de modèles	5
2	Transformations modèle à modèle (M2M)	6
2.1	L'outil ATL	6
2.2	De SimplePDL à PetriNet	7
2.2.1	WorkDefinition WD	7
2.2.2	NeedSet	8
3	Transformations modèles à textes (M2T)	9
3.1	L'outil Acceleo	9
3.2	SimplePDL	9
3.3	PetriNet	9
4	Transformations textes à modèles (T2M)	11
4.1	L'outil Xtext	11
4.2	SimplePDL	11
4.3	PetriNet	11

Chapitre 1

Les métamodèles SimplePDL et PetriNet

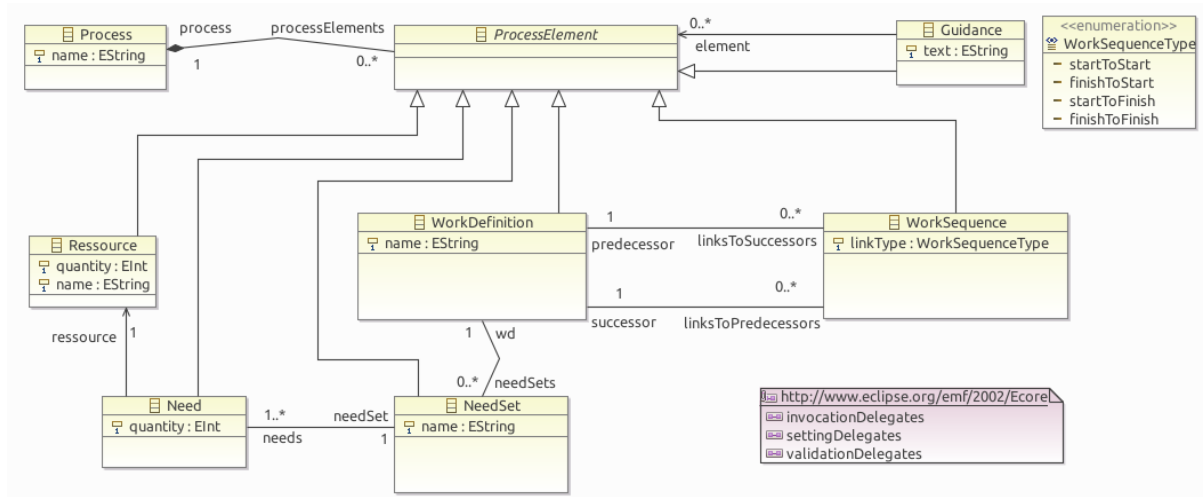
Pour l'ensemble de la chaîne de transformation on aura besoin des métamodèles SimplePDL et PetriNet, on définit donc ces deux métamodèles ainsi que des exemples de modèles qui serviront à faire les tests.

1.1 SimplePDL

On a ajouté au modèle SimplePDL la gestion des ressources. Pour cela, on a introduit :

- Ressource : une ressource ayant un nom et étant présente en une certaine quantité
- Need : un besoin représentant un nombre de ressources d'un certain type
- NeedSet : un ensemble de besoins nécessaires à la réalisation d'une WD

Voilà un diagramme représentant le métamodèle SimplePDL que j'ai utilisé :



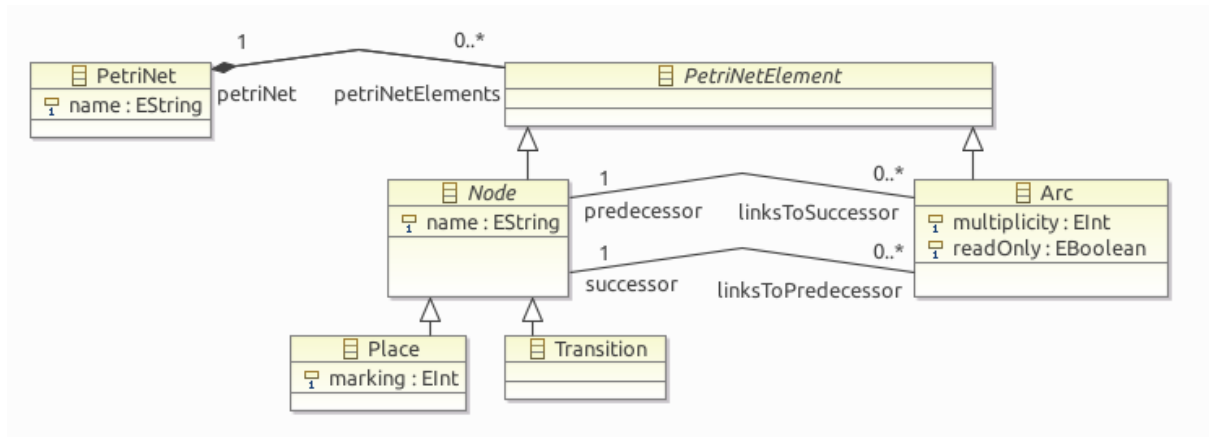
Ce modèle n'est pas suffisant pour décrire entièrement l'ensemble des contraintes. Pour cela on a donc ajouté des contraintes OCL avec OCLinEcore.

On a ajouté les règles suivantes :

- class Process
 - sameWDName : les WD doivent avoir des noms différents
 - sameRessourcesName : les ressources doivent avoir des noms différents
 - nameForbidden : un processus ne peut pas s'appeler : "Process"
 - sameNeedSetsName :
- class WorkDefinition
 - voidName : nom vide interdit
- class WorkSequence
 - previousWDinSameProcess : la WD précédente doit être dans le même processus
 - reflexivity : le prédécesseur et le successeur doivent être différents
 - nextWDinSameProcess : la WD suivante doit être dans le même processus
- class Ressource
 - voidName : nom vide interdit
 - positiveQuantity : la quantité de ressources doit être ≥ 0
- class Need
 - positiveQuantity : la quantité de ressources d'un besoin doit être > 0
- class NeedSet
 - voidName : nom vide interdit

1.2 PetriNet

Le modèle PetriNet que j'ai utilisé est représenté par le diagramme suivant :



Ce modèle n'est pas suffisant pour décrire entièrement l'ensemble des contraintes. Pour cela on a donc ajouté des contraintes OCL avec OCLinEcore.

On a ajouté les règles suivantes :

- class **PetriNet**
 - `voidPetriName` : nom vide interdit
 - `sameNodeName` : les noms des noeuds doivent être différents
- abstract class **Node**
 - `voidNodeName` : nom vide interdit
- class **Arc**
 - `positiveMultiplicity` : la multiplicité doit être > 0
 - `sameTypeOfPredecessorAndSuccessor` : Prédecesseur et successeur doivent être différents
 - `nextNodeNotInSamePetriNet` : les noeuds reliés sont dans le même réseau de pétri
 - `previousNodeNotInSamePetriNet` : les noeuds reliés sont dans le même réseau de pétri
- class **Place**
 - `positiveMarking` : le marquage d'une place est ≥ 0

1.3 Exemples de modèles

Voici quelques exemples de modèles SimplePDL (sans ressources) :

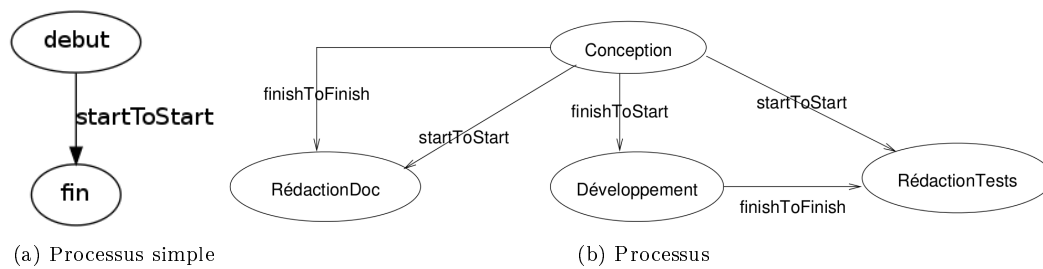


FIGURE 1.1: Exemples de modèles de processus SimplePDL

Et quelques exemples de modèles PetriNet :

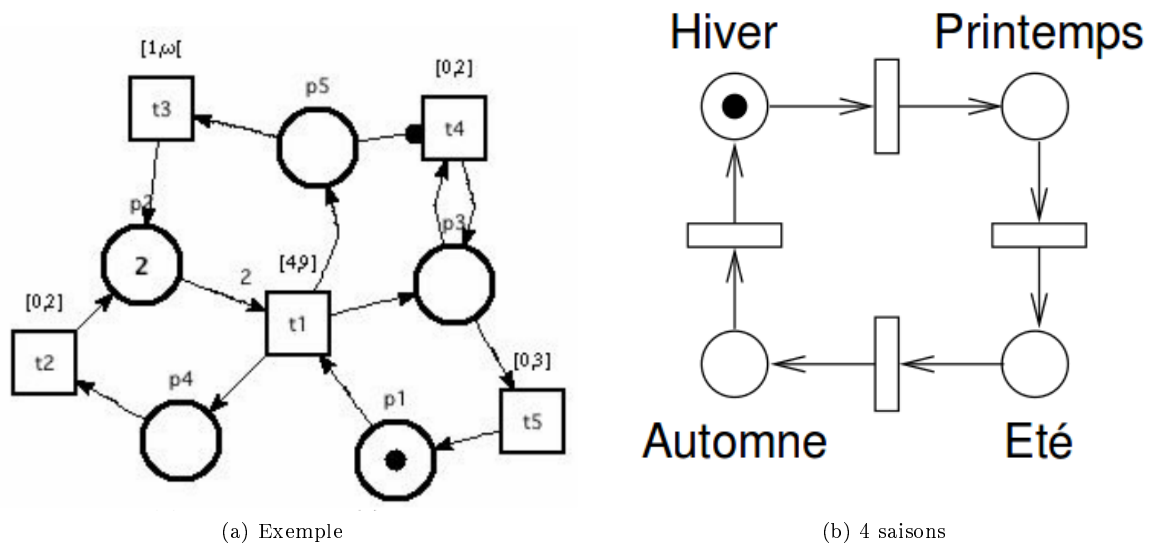


FIGURE 1.2: Exemples de modèles de réseaux de pétri

Chapitre 2

Transformations modèle à modèle (M2M)

Pour analyser les modèles de processus en SimplePDL, on va avoir besoin de les transformer en réseaux de pétri. Pour ça, on doit effectuer des transformations ModelToModel (M2M) avec comme origine un modèle SimplePDL et comme arrivée un modèle PetriNet.

On utilise l'outil ATL (plugin pour Eclipse) afin d'effectuer ces transformations.

2.1 L'outil ATL

On montre ici brièvement la syntaxe d'un fichier ATL avec quelques exemples :

```
1 — Traduire un Process en un PetriNet de même nom
2 rule Process2PetriNet {
3   from p: simplepdl!Process
4   to pn: petrinet!PetriNet (name <- p.name)
5 }
```

Les règles de transformation sont exprimées avec le mot-clé "rule". Les mots "from" et "to" permettent d'identifier les éléments de la transformation.

```
1 — Traduire une WorkSequence en un motif sur le réseau de Petri
2 rule WorkSequence2PetriNet {
3   from ws: simplepdl!WorkSequence
4   to
5     a_ws: petrinet!Arc(
6       petriNet <- ws.getProcess(),
7       readOnly <- true,
8       multiplicity <- 1,
9       predecessor <- thisModule.resolveTemp(ws.predecessor,
10        if((ws.linkType = #finishToStart) or (ws.linkType = #finishToFinish))
11          then 'p_finished'
12          else 'p_started'
13        endif),
14       successor <- thisModule.resolveTemp(ws.successor,
15        if((ws.linkType = #finishToStart) or (ws.linkType = #startToStart))
16          then 't_start'
17          else 't_finish'
18        endif)
19     )
20 }
```

La commande "resolveTemp" permet d'identifier un élément parmi ceux présents dans les règles de transformations.

2.2 De SimplePDL à PetriNet

Le code de transformation complet étant assez long je vais ici expliquer à l'aide de graphiques les opérations pour transformer les éléments SimplePDL en éléments de réseau de pétri :

2.2.1 WorkDefinition WD

J'ai voulu modifier au minimum la structure des WD et des WS dont on avait les transformations avant l'ajout des ressources aux modèles. Ainsi on assure une évolution simple des WD et WS.

On commence par une WorkDefinition WD. Avec les ressources en plus, une WD a besoin de savoir si des ressources sont disponibles pour pouvoir démarrer une activité.

- On ajoute donc une place ressources_available reliée à la transition start d'une WD.
- On ajoute également une place ressources_wait (marquée à 1) pour indiquer que la WD est en attente de ressources
- Enfin on ajoute une place ressources_returned reliée en sortie de la transition finish qui indique que la WD a fini d'utiliser les ressources

On a donc ajouté 3 places et 2 arcs à notre WD. Voici un graphique récapitulatif :

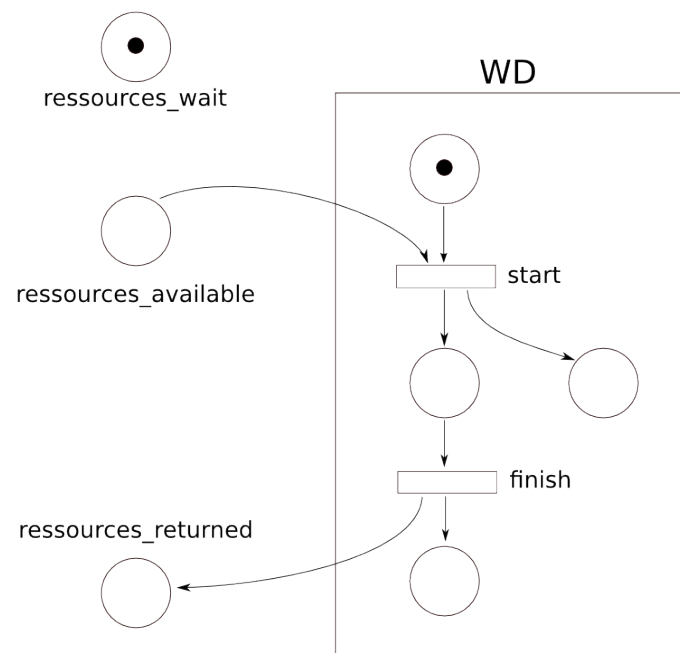


FIGURE 2.1: Transformation d'une WorkDefinition WD en réseau de pétri

2.2.2 NeedSet

Je ne décris pas le besoins (Need) car il s'agit d'une simple place avec un marquage correspondant à "quantity".

Chaque ensemble de besoins (NeedSet) correspond à un ensemble constitué de :

- une place "set" pour indiquer de quel NeedSet il s'agit
- 3 transition :
 - "take" qui est reliée à chaque ressource nécessaire pour le NeedSet
 - "makeAvailable" entre la place "set" du NeedSet et la place "ressources_available" de la WD
 - "return" reliée aux ressources qui vont être rendues
- enfin les arcs qui vont bien (voir schéma)

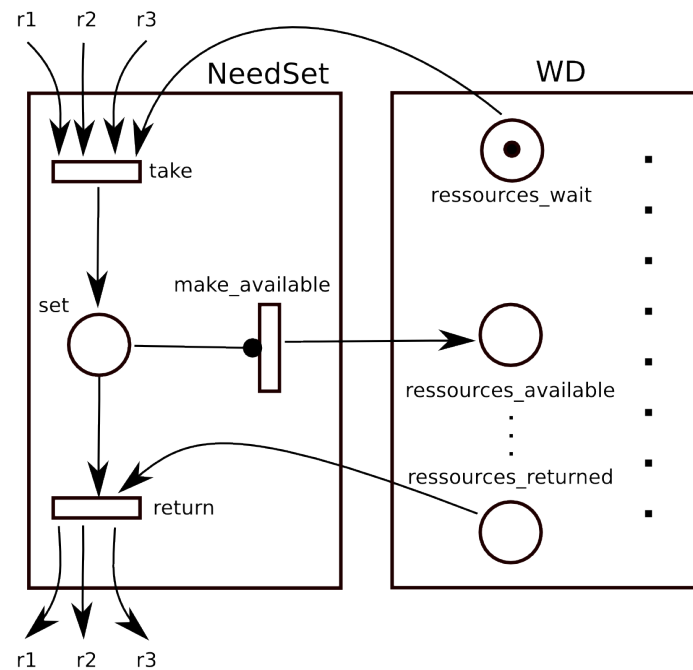


FIGURE 2.2: Transformation d'un ensemble de besoins NeedSet en réseau de pétéri

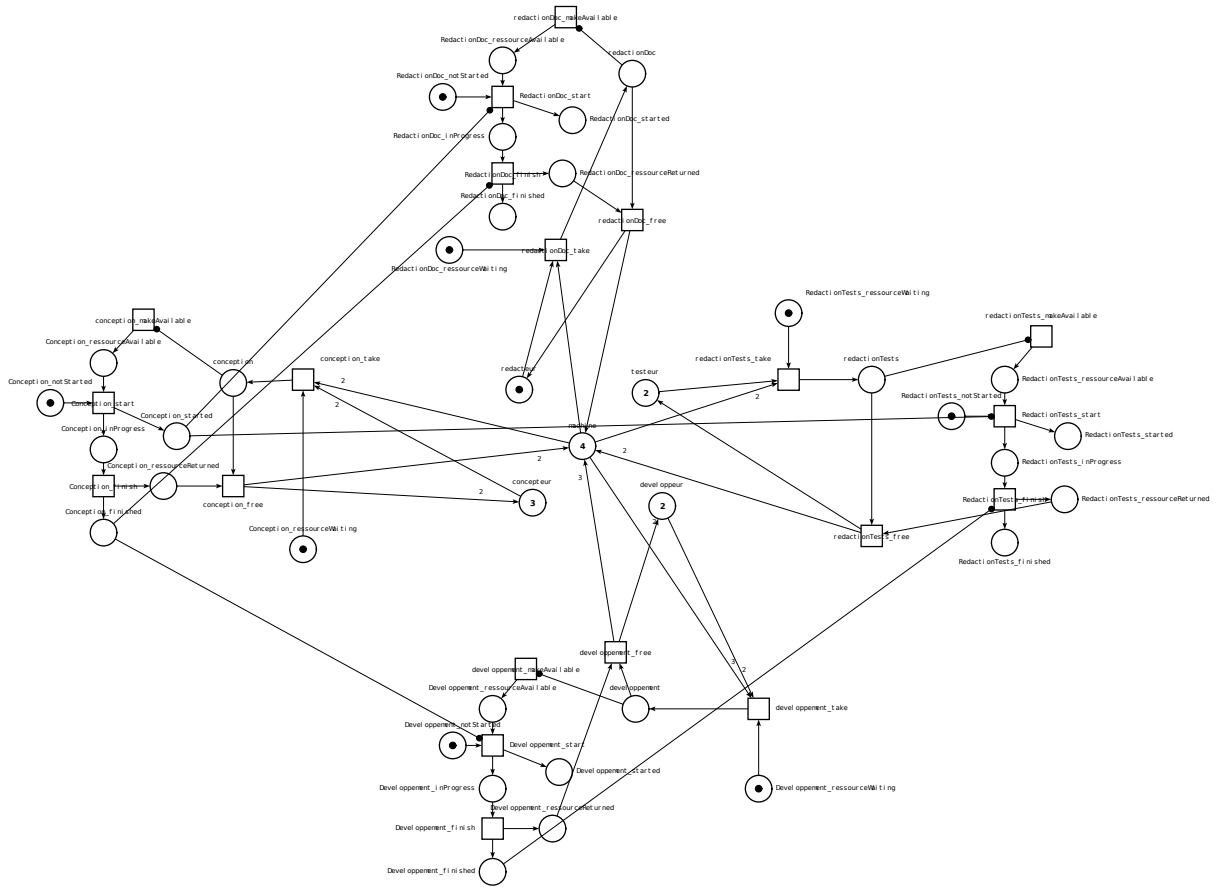
Chapitre 3

Transformations modèles à textes (M2T)

3.1 L'outil Acceleo

3.2 SimplePDL

3.3 PetriNet



Chapitre 4

Transformations textes à modèles (T2M)

4.1 L'outil Xtext

4.2 SimplePDL

4.3 PetriNet