

Chaîne de vérification de modèles de processus

El Bouzekraoui Younes — MDAA Saad

Département Sciences du Numérique - Deuxième année
2020-2021

Contents

1	Introduction	3
2	Le Métamodèle SimplePDL	3
2.1	Modelisation du métamodèle	3
2.2	Contraintes OCL	3
3	Le Métamodèle PetriNet	4
3.1	Modelisation du métamodèle	4
3.2	Contraintes OCL	4
4	Un éditeur graphique SimplePDL	5
5	Syntaxe concrète textuelle de SimplePDL avec Xtext	5
5.1	Définition de la syntaxe PDL1: (voir PDL1.xtext)	5
6	Transformation SimplePDL vers PetriNet en utilisant EMF/Java	5
6.1	Principe de la transformation d'un modèle de processus en réseau de Petri	6
7	Transformation SimplePDL vers PetriNet en utilisant ATL	6
7.1	Principe de la transformation d'un modèle de processus en réseau de Petri	6
8	Tests de la transformation M2M	6
9	Transformation PetriNet vers Tina en utilisant Acceleo	7

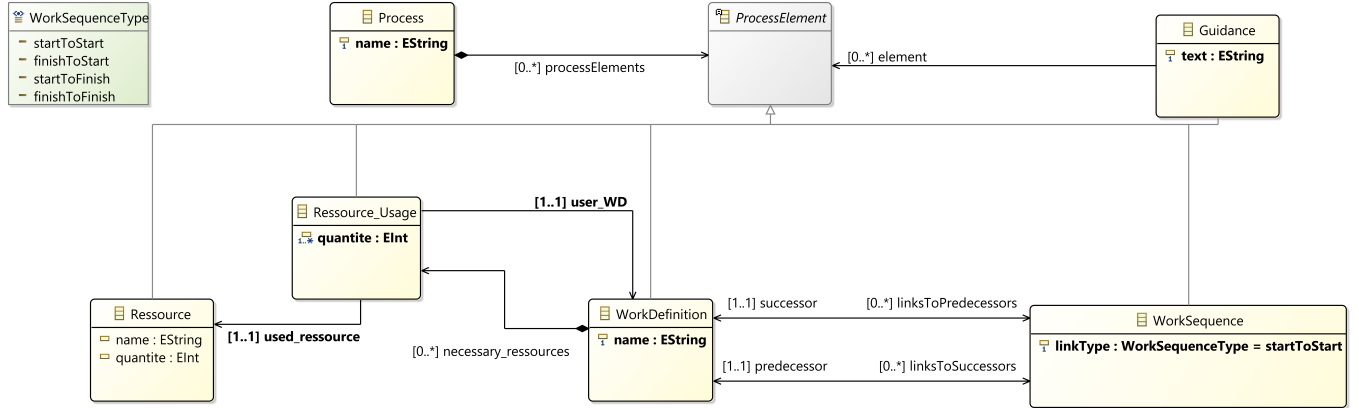
1 Introduction

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.

2 Le Métamodèle SimplePDL

2.1 Modelisation du métamodèle

On a choisi de définir le Métamodèle SimplePDL comme suit (voir **SimplePDL.ecore**):



Notre Métamodèle SimplePDL est composé de :

- **Process** : le processus qu'on modélise
- **ProcessElement** : une interface pour faire abstraction sur les éléments d'un processus
- **Guidance** : des notes écrites
- **Workdefinition** : une activité ayant un nom de type **EString**
- **WorkSequence** : une dépendance entre deux activités
- **Ressource** : une ressource ayant un nom de type **EString** et une quantité de type **EInt**
- **Ressource_Usage** : le lien entre une activité et une ressource

Pour prendre en compte l'utilisation des ressources on a ajouté les deux classes : **Ressource** et **Ressource_Usage** qui héritent de **ProcessElement**, et on a ajouté un attribut à **Workdefinition** : **necessary_resources** qui est une liste des ressources nécessaires pour une activité. Un objet **Ressource_Usage** contient un lien vers une activité et un lien vers une ressource et la quantité consommée.

2.2 Contraintes OCL

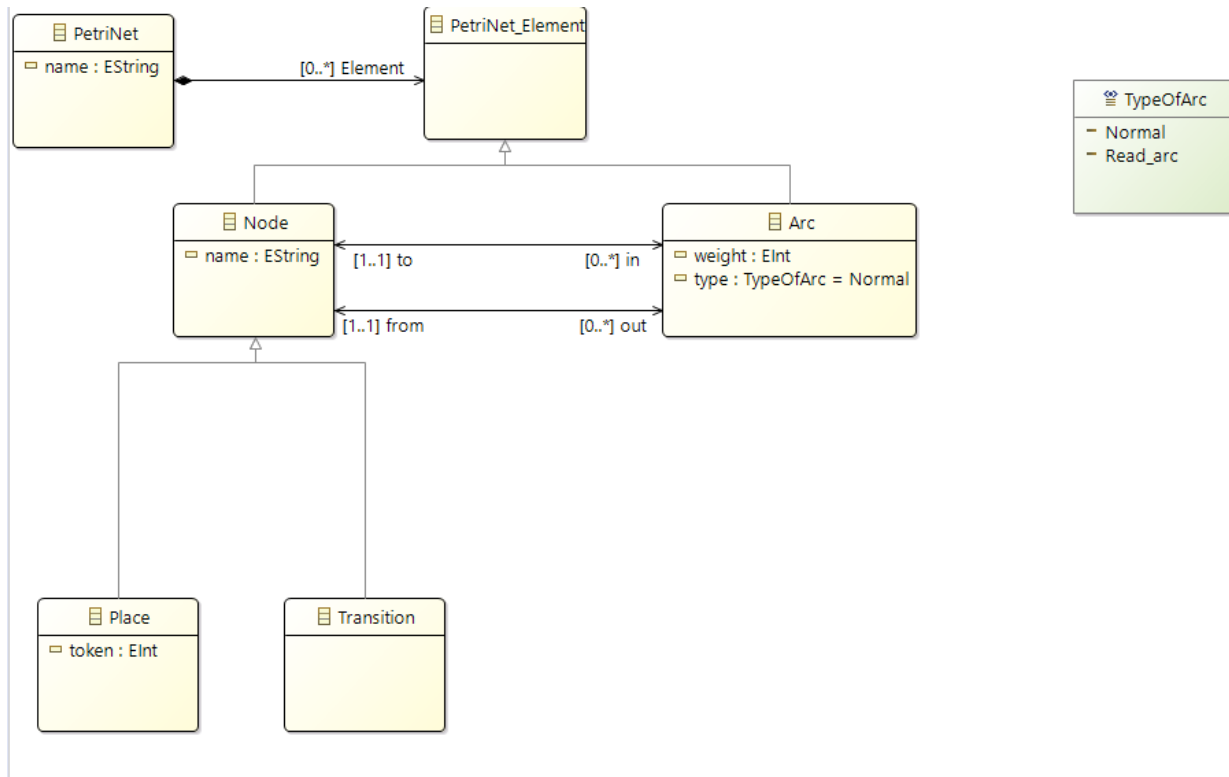
Afin de capturer plus de contraintes On a ajouté les règles ocl suivantes (voir **SimplePDL.ocl**):

- une dépendance ne peut pas être réflexive.
- deux sous-activités différentes d'un même processus ne peuvent pas avoir le même nom.
- le nom d'une activité doit être composé d'au moins un caractère.
- le nom d'une ressource doit être composé d'au moins un caractère.
- deux ressources ne peuvent pas avoir le même nom.
- la quantité d'une ressource doit être ≥ 0
- la quantité d'une ressource utilisée doit être ≤ 0
- la quantité d'une ressource utilisée par une activité doit être inférieur ou égale à la quantité disponible

3 Le Métamodèle PetriNet

3.1 Modelisation du métamodèle

On a choisi de définir le Métamodèle SimplePDL comme suit (voir **PetriNet.ecore**):



Notre Métamodèle PetriNet est composé de :

- PetriNet : le réseau petri qu'on modélise
- PetriNet_Element : une interface pour faire abstraction sur les éléments d'un réseau petri
- Node : une interface pour faire abstraction sur une place et une transition
- Arc : modélise le lien entre une place et une transition de type Normal ou Read_arc
- Place
- Transition

3.2 Contraintes OCL

Afin de capturer plus de contraintes On a ajouté les règles ocl suivantes (voir **PetriNet.ocl**):

- un arc doit suivre le modele suivant place arc transition arc place.
- le poids d'un arc ne peut pas être nul (sinon l'arc ne sert à rien).
- le nombre des jetons dans une place est positive.
- le nom d'une node doit être composé d'au moins un caractère.
- deux nodes différente peuvent pas avoir le même nom.
- le nom d'un réseau Petri doit être composé d'au moins un caractère.

4 Un éditeur graphique SimplePDL

5 Syntaxe concrète textuelle de SimplePDL avec Xtext

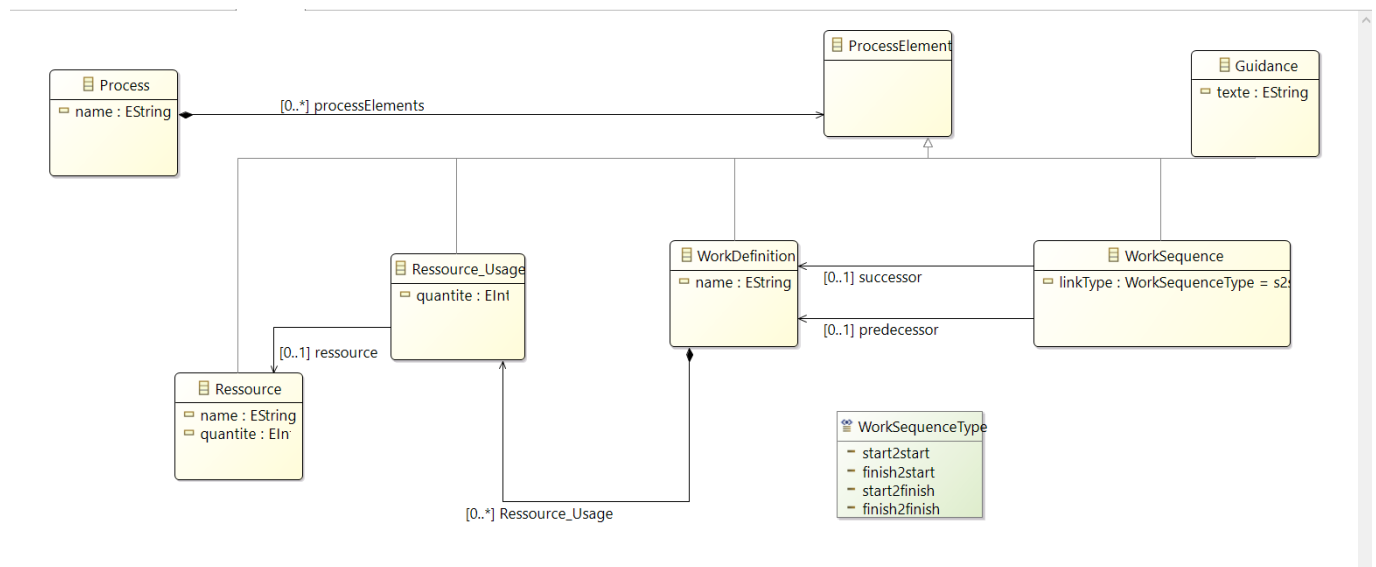
Xtext consiste à définir une syntaxe concrète textuelle pour un DSML au travers d'une grammaire, et il permet aussi de construire un métamodèle conforme à la syntaxe concrète décrite

5.1 Définition de la syntaxe PDL1: (voir PDL1.xtext)

Un exemple de la syntaxe textuelle souhaité est donnée dans l'exemple ci-dessous.

```
process dev {  
  rs concepteur qt 3  
  rs developpeur qt 2  
  rs machine qt 4  
  rs redacteur qt 2  
  rs testeur qt 2  
  
  wd Conception needs to get 2 of concepteur get 1 of machine  
  wd RedactionDoc needs to get 1 of machine get 1 of redacteur  
  wd Programmation needs to get 2 of developpeur get 3 of machine  
  wd RedactionTests needs to get 2 of machine get 1 of testeur  
  
  ws f2f from Conception to RedactionDoc  
  ws s2s from Conception to RedactionDoc  
  ws f2f from Conception to Programmation  
  ws s2s from Conception to RedactionTests  
  ws f2f from Programmation to RedactionTests  
}
```

Le métamodèle généré à partir de cette syntaxe textuelle est donnée ci-dessous :



Ce métamodèle est différent de celui qu'on définit au début.

6 Transformation SimplePDL vers PetriNet en utilisant EMF/Java

EMF nous permet de générer les classes Java à partir des modèles SimplePDL et PetriNet, ce qui nous permet de définir une transformation de SimplePDL vers PetriNet en utilisant Java (voir **SimplePDL2PetriNet.java**)

6.1 Principe de la transformation d'un modèle de processus en réseau de Petri

Le principe de la transformation d'un modèle SimplePDL en Réseau de Pétri est le suivant :

- Un élément Process devient un élément PetriNet.
- Une WorkDefinition devient 4 places ready, started, running et finished et deux transitions start et finish.
- Une WorkSequence devient un read arc entre une place de l'activité précédente (started ou finished) et une transition de l'activité cible (start ou finish).
- Une Ressource devient une place, le nombre de jetons indique la quantité
- Un Ressource.Usage devient deux arc , un arc dont la multiplicité est la quantité utilisé allant de la place ressource vers la transition start de l'activité . et l'autre arc allant de la transition finish vers la place ressource

7 Transformation SimplePDL vers PetriNet en utilisant ATL

ATL est un langage et une boîte à outils de transformation de modèles. Dans le domaine de l'ingénierie pilotée par les modèles, ATL fournit des moyens de produire un ensemble de modèles cibles à partir d'un ensemble de modèles source.

7.1 Principe de la transformation d'un modèle de processus en réseau de Petri

On a suivit les mêmes principes décrit dans la transformation java, on a défini les règles suivantes : (voir **SimplePDL2PetriNet.atl**).

Vue qu'on a pas défini précédemment une référence opposite entre PetriNet et PetriNetElement l'opération suggéré lors du tp:

```
net <- wd.getProcess()
```

n'est pas possible, donc on a ajouté les places / transitions / arc créés lors de la règle

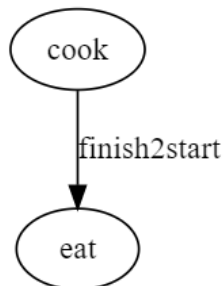
```
Process2PetriNet {...}
```

en utilisant

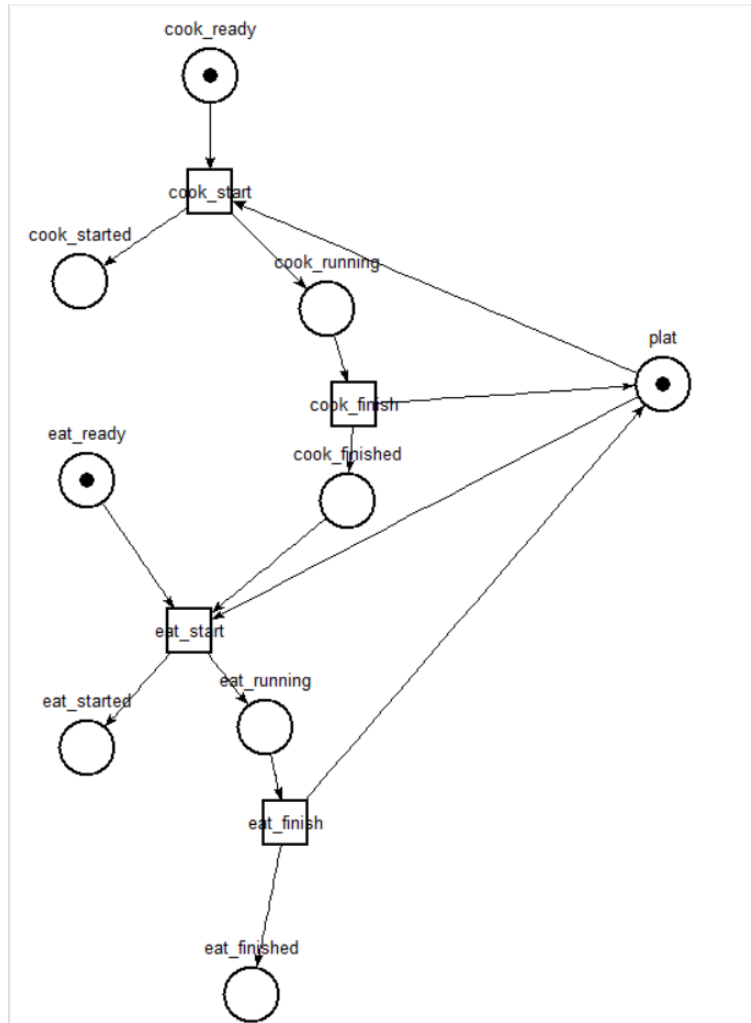
```
thisModule.resolveTemp(...)
```

8 Tests de la transformation M2M

On commence par tester la transformation sur un modèle simple (voir **pdl-simple.xmi**), il est composé de 2 activités : (cook et eat) ,une dépendance de type finishTostart entre les 2 activités, et une ressource plat utilisé par cook et eat.



la transformation nous donne le modèle Petrinet suivant:



en utilisant le stepper analyser de l'outil netdraw notre modèle Petrinet semble avoir le comportement correcte

9 Transformation PetriNet vers Tina en utilisant Acceleo