ECOLE D'INGÉNIEUR ENSEEIHT

Rapport de Génie du logiciel et des systèmes
BE

Florian GARIBAL
Lucien HAURAT

28 octobre 2016

Sommaire

1 Introduction

Dans le cadre du module de Génie du Logiciel et des systèmes nous avons été amené à travailler sur les modèles, méta-modèles à l'aide du méta-méta-modèle *eCore*. Ce projet se décompose en différentes phases qui traitent toutes de sujets et techniques différentes mais qui permettent au final de créer une chaîne de vérification.

2 Le métamodèle *SimplePDL*

2.1 Conception

Nous avons choisi d'implémenter le métamodèle SimplePDL comme proposé dans le TP2 :

— Process : l'ensemble du réseau de processus.
— ProcessElement : chaque élément du réseau de processus
— WorkDefinition : une activité
— WorkSequence : le lien entre deux activités
— Guidance : une note

Les trois derniers héritent de ProcessElement, qui est liée par une relation d'aggrégation à Process. Ainsi une instance de Process a comme attributs une liste de ProcessElement qui sont les différentes WorkDefinition, WorkSequence et Guidance du processus instancié.

Par ailleurs une *WorkSequence* permet de lier deux *WorkDefinition* en leur donnant un ordre d'éxécution. En effet un label est présent entre eux qui défini à partir de quand ils pourront s'éxécuter/se terminer. Ci dessous la liste des différentes possibilités concernant ce label :

— Start2Start (s2s) : A2 commence dés que A1 commence
— Start2Finish (s2f) : A2 finit dés que A1 commence
— Finish2Start(f2s) : A2 commence dés que A1 finit
— Finish2Finish (f2f) : A2 finit dés que A1 finit

2.2 L'ajout de la gestion des ressources

Pour les ressources, nous avons choisi d'ajouter deux classes : Resource et UseResource. Les deux héritent de ProcessElement. Nous avons aussi ajouté un attribut à WorkDefinition : *uses* qui est une liste de UseResource. Un objet *Resource* a un nom et une quantité maximale. Un objet *UseResource* contient un lien vers la Resource voulue, un poids (quantité de ressource demandée par le travail) et un lien vers la WorkDefinition associée. Ainsi, en regardant dans l'attribut *uses* d'un travail (WorkDefinition), on peut donc savoir de combien de ressources il a besoin pour pouvoir s'effectuer. Cette modélisation rend complètement indépendants les Resource et les WorkDefinition, le lien étant fait par la classe *UseResource*.

On obtient donc le diagramme suivant après l'ajout de la gestion des ressources :

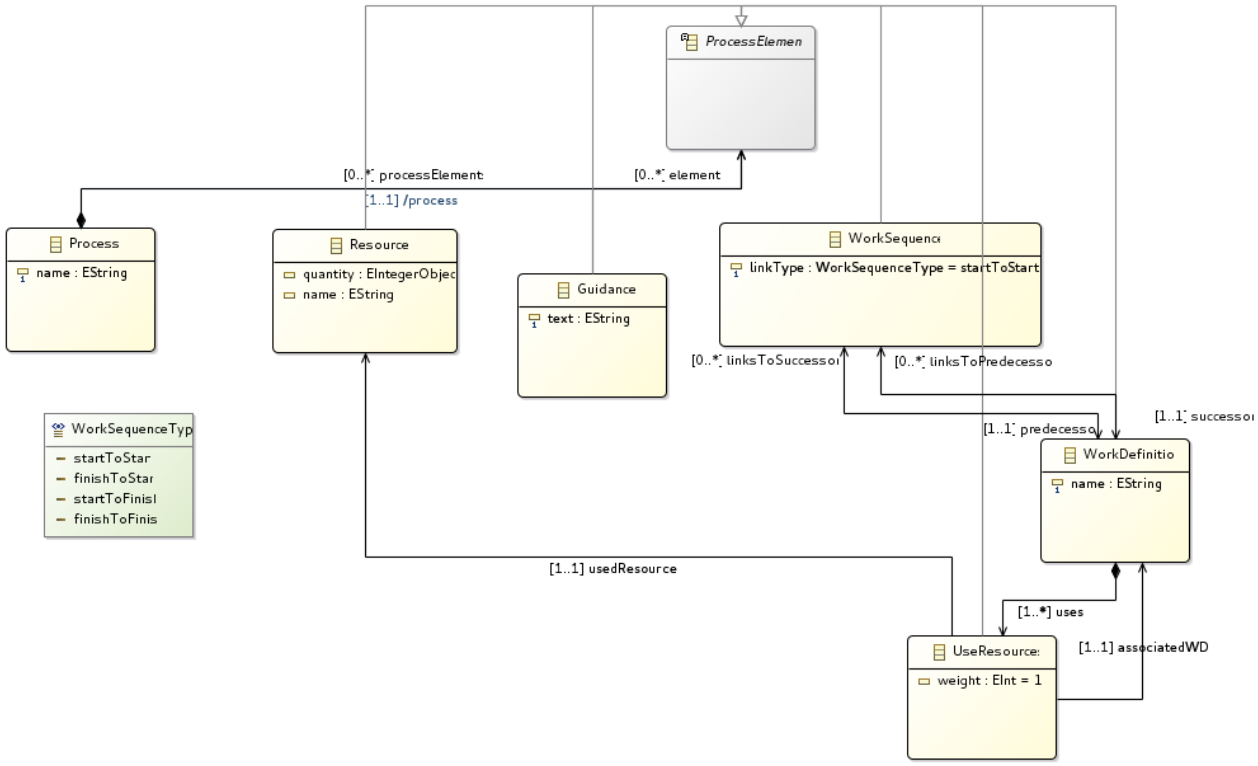

FIGURE 1 – Diagramme du métamodèle simplePDL complété par les ressources

2.3 Mise en place des *contraintes OCL*

Les contraintes que nous avons implémenté sont les suivantes :
— Tous les Process ont des noms différents
— Les noms des WorkDefinition ne sont pas vides
— Une WorkSequence ne part pas d'une WorkDefinition pour y revenir (pas de boucle)
— Les prédecesseurs et suivants d'une WorkDefinition sont dans le même Process
— Le nom d'une Resource n'est pas vide
— La quantité demandée par un UseResource est positive ou nulle
— La quantité demandée par un UseResource est inférieure ou égale au nombre max de cette Resource

3 Le métamodèle *PetriNet*

3.1 Conception

Le modèle PetriNet, implémenté en TP2, se compose des classes suivantes :
— RéseauPetri : l'ensemble du réseau
— PetriElement : Un élément du réseau
— Place : une place
— Transition : une transition entre deux places
— Arc : un lien entre une transition et une place ou inversement

Nous avons choisi de suivre le modèle utilisé pour SimplePDL et créer une classe qui contient une liste de *PetriElement*. D'autre part, toujours en prenant exemple sur SimplePDL les trois dernières classes héritent toutes de la classe *PetriElement*.

Concernant les ressources, aucune modification du modèle n'est nécessaire car les ressources, en PetriNet, ne sont en réalité que des places disposant d'un *nombre de jetons* équivalent au nombre de ressources réellement disponibles. Ensuite le nombre de ressources dont a besoin chaque activité est précisé par le poids présent sur l'Arc liant la ressource à la transition.

On obtient donc le diagramme suivant :

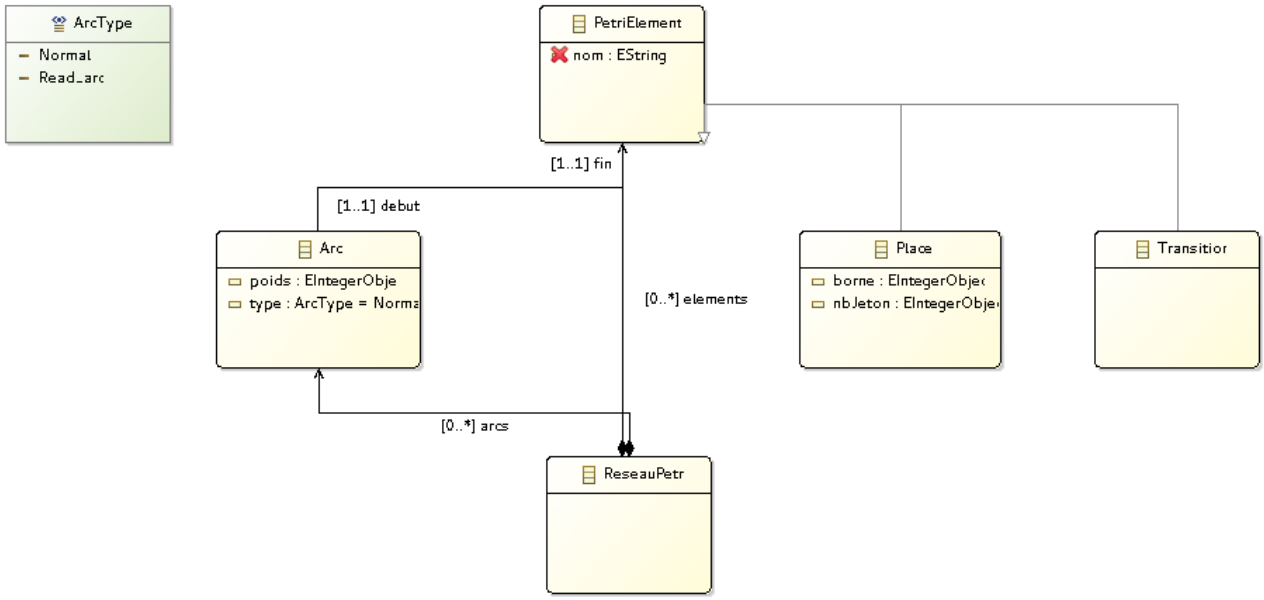

FIGURE 2 – Diagramme du métamodèle PetriNet

3.2 Mise en place des *contraintes OCL*

Les contraintes que nous avons implémenté sont les suivantes :
— Arcs :
  — Les arcs sont correct c-a-d ont une **place en début et une transition en fin** ou l'inverse **mais pas** une place en début et fin ou une transition en début et fin
  — Les arcs ne bouclent pas : extrémité différente de la fin
  — Le poids sur un arc e peut être négatif
— Place :
  — Le nombre de jetons que **dispose** la place est positif ou nul
  — Le nombre **maximum** que la place peut disposer est strictement positif
— PetriElement :
  — Le nom de l'élément ne peut être nul ou vide
— ReseauPetri :
  — Le nom du réseau ne peut être nul ou vide

4 Design

Durant les TP nous avons pu représenter et éditer le modèle du PetriNet grâce à Sirius dans Eclipse. Malheureusement, le déploiement des greffons pour SimplePDL n'a pas marché et Eclipse donnait constamment la même erreur : le plugin avec l'uri *http ://simplepdl* n'existe pas.

Cependant nous avions réussi à définir les différentes formes que nous souhaitions voir apparaître dans notre éditeur graphique tel quelle :

— WorkDefinition : Losange Vert
— WorkSequence : Flèche noire avec label (s2s, f2s, ...)
— Guidance : Rectangle jaune
— Resource : E(c)llipse
— UseResource : Flèche rouge avec label (quantité de ressources)

5 Syntaxe textuelle concrête *Xtext*

La syntaxe contextuelle *Xtext* a pour but de définir des éditeurs de texte permettant de "compiler" à la volée les fichiers du type créé. En effet ces éditeurs sont dotés d'auto complétion qui permettent une écriture plus facile et plus rapide. D'autre part en définissant un fichier *Xtext* on est libre de définir la syntaxe de notre choix, à savoir la ponctuation, l'ordre des objets, etc.

De ce fait nous avons décidé de le définir de la façon suivante :

— Process : **Process <name> { [ProcessElement]* }**
— ProcessElement : Peut être soit une *WorkDefinition*, soit une *WorkSequence* soit une *Resource* soit une *UseResource*
— WorkDefinition : **wd <name>**
— WorkSequence :
ws <linkType> {
from <WorkDefinition>
to <WorkDefinition>
}
— Resource : **res <name> : <quantitéDispo>**
— UseResource : **<WorkDefinition> require <nbRessources> to <Ressource>**

Exemple d'un simplePDL représenter sous cette forme :

```
process Process1 {
  wd A1
  wd A2
  ws s2s {
    from A1
    to A2
  }
  res R1 : 5
  res R2 : 2
  A1 require 4 to R1
  A2 require 2 to R2
}
```

FIGURE 3 – Test d'un fichier avec la syntaxe textuelles simplePDL choisie

6 Transformation modèle à modèle : règles *ATL*

Afin de passer d'un simplePDL à un PetriNet de façon rapide et relativement facile nous avons défini des règles ATL qui, pour chaque élément du simplePDL, défini ce qu'il devient dans le PetriNet.

Afin d'écrire ces règles nous avons suivi les indications données et nous en sommes arrivés aux règles suivantes :

— Process : le *Process* devient un *PetriNet* du même nom

— WorkDefinition : Un *WorkDefinition* devient 4 places correspondant à 4 états différents pour le WorkDefinition à savoir **not started,start, in progress/running, finished**. L'ensemble de ces places sont liées par deux transitions qui correspondent au moment où l'activité se lance et se termine.

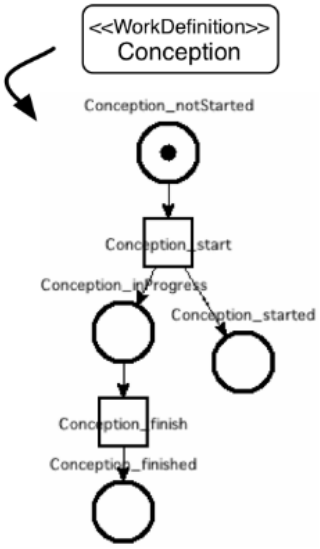

FIGURE 4 – Schéma de la transformation d'une WorkDefinition en quatre places et 2 transition

— WorkSequence : Selon le type de lien on lie les places et transitions tels qu'il suit :

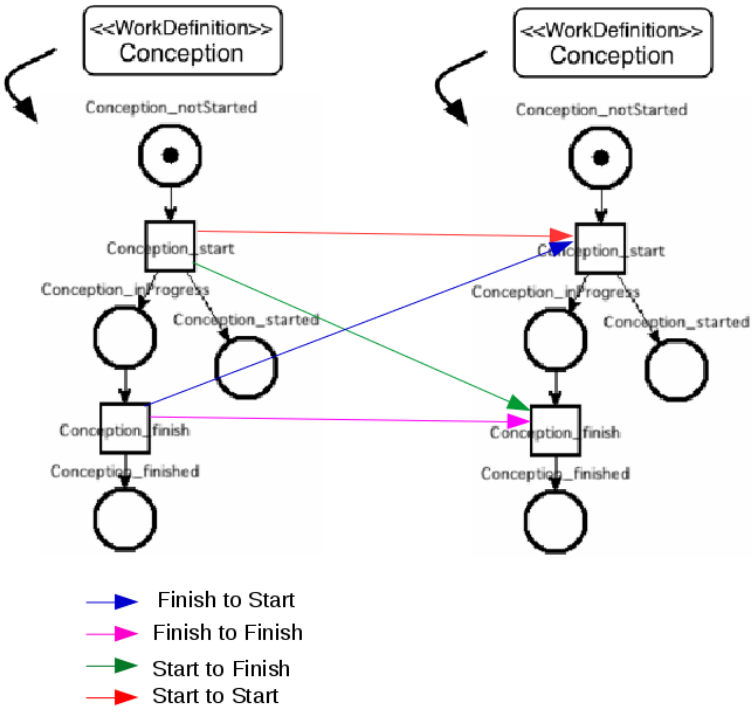

FIGURE 5 – Schéma de la transformation d'une WorkSequence

— Resource : Devient une place avec le nombre de jeton égal au nombre de ressources dispo
— UseResource : Devient deux arcs, un entre le début de l'activité et la ressource précédemment crée et un entre la ressource et la fin de l'activité. Le schéma suivant décrit la transformation :

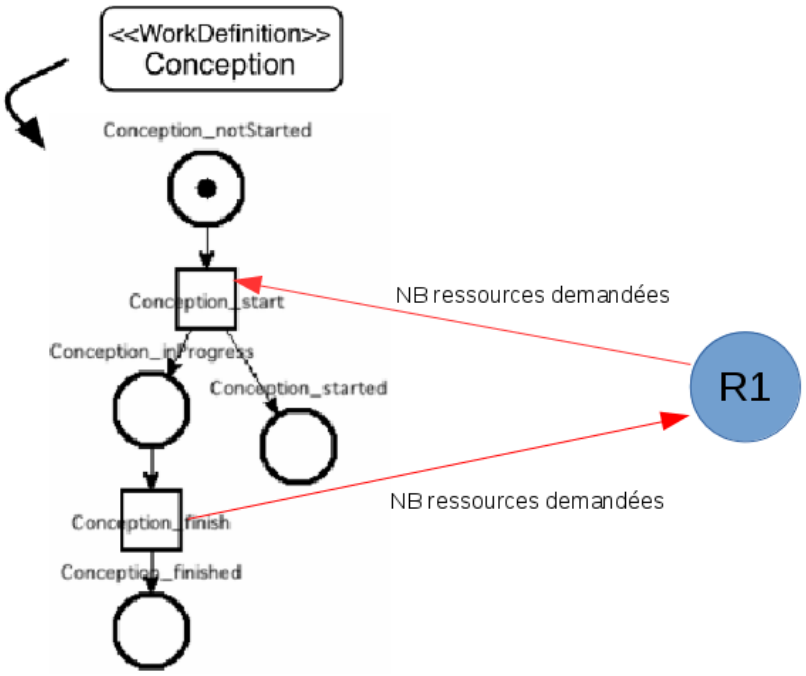

FIGURE 6 – Schéma de la transformation d'un UseResource en PetriNet

7 Conclusion

Ce projet fut très intéressant autant par le contenu que par la découverte des techniques utilisées. En effet nous n'avions jamais travaillé avec ce genre de techniques et plug-in sur Eclipse. Cependant nous avons eu de nombreux problèmes avec le logiciel Eclipse ce qui nous a très ralenti dans l'accomplissement de toutes les tâches demandées. Ces erreurs étaient le plus souvent une non-reconnaissance des packages "registered" ou plugins, en particulier simplepdl, ou alors un problème de lancement d'Eclipse.