Rapport du mini-projet Ingénierie Dirigée par les Modèles

Kawtar LYAMOUDI - Nihal BELMAKHFI

November 15, 2022

1 Introduction

Objectif: L'objectif de ce mini projet est de 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 réaliser cet objectif, on a pu découvrir de nombreux outils et la manipulation/transformation de modèles. Aussi, on a du traduire un modèle de processus en un réseau de Petri et utiliser plusieurs outils de model-checking définis sur les réseaux de Petri au travers de la boîte à outils Tina.

2 Les métamodèles (SimplePDL et PetriNet)

2.1 SimplePDL

Le SimplePDL est un langage de métamodélisation qui décrit des modèles de processus. Créons tout d'abord un modèle basé sur un langage simplifié de modélisation de processus de développement. Ce modèle est appelé SimplePDL.

Ensuite, on va modifier ce modèle afin de le rendre un langage plus avancé résultant de l'ajout de la notion de ProcessElement comme généralisation de WorkDefinition (activité) et WorkSequence (dépendance).

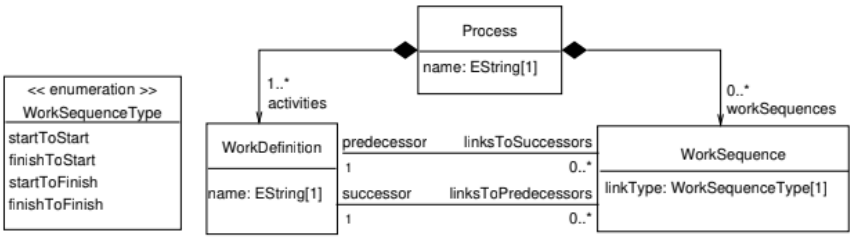

Figure 1: Diagramme du métamodèle SimplePDL simplifié

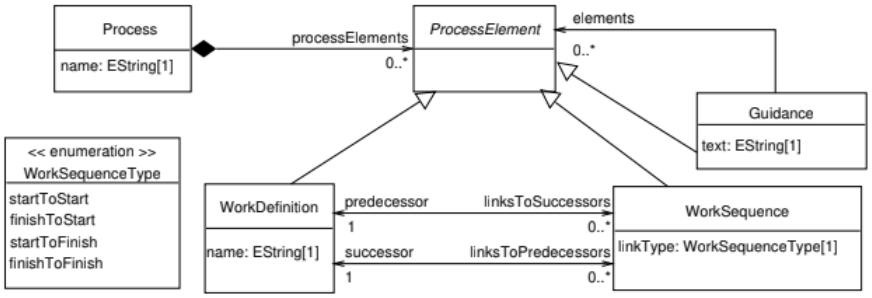

Figure 2: Diagramme du métamodèle SimplePDL avancé

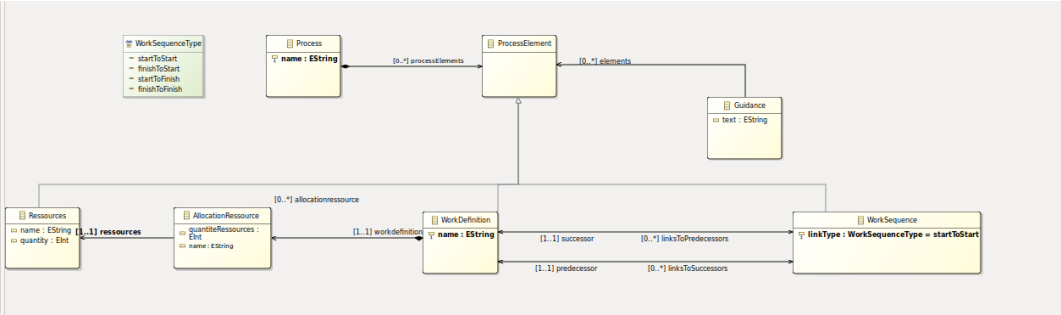

Figure 3: Le métamodèle SimplePDL aprés avoir ajouter les ressources

Eclipse Modeling Project offre des outils pour manipuler le métamodèle de SimplePDL. Pour cela on utilise: — Eclipse Modeling Framework (EMF: une framework de modélisation), il fournit une infrastructure pour la manipulation des modèles, cette infrastructure permet aussi la génération de code et des applications à partir des modèles. — ECore qui est aussi un métamodèle disposant d'un éditeur graphique pour manipuler les métamodèles.

Pour assurer la réalisation d'une certaine activité, il faut lui donner une ou plusieurs ressources afin d'assurer sa finalité.

Une ressource contient des occurences, une activité loue des occurences d'une ressources au début de son execution, ces occurences sont alors utilisés exclusivement par cette actitivé jusqu'à la fin de sa réalisation.

Ainsi la nécessité d'ajouter deux classes au méta-modèle de SimplePDL : — Ressources : une ressource est définit par son nom (EString) qui décrit son type et ses occurences (Eint). — Allocation-Ressources : cette classe représente le nombre d'occurences prises par une activité parmi les occurences d'une ressource pour effectuer sa réalisation.

Une activité (WorkDefinition) aura besoin éventuellement de plusieurs ressources, ainsi elle sera composer de plusieurs classes AllocationRessources.

2.2 Réseau de petri

Aprés avoir créer le métamodèle SimplePDL, on a fait un modèle pour les réseaux de Petri (un outil graphique mathématique qui s'inscrit dans le domaine large de la théorie des réseaux.), qui est bien un métamodèle permettant de décrire le comportement dynamiques des systèmes aux élèments discrets,

2.2.1 Composition d'un réseau de petri

Le réseau de Petri est un tuple qui se compose de plusieurs éléments :Des places, des transitions, des arcs (représentés par des flèches). Un arc doit nécessairement etre entre une place et une transition.

Une place peut contenir un ou plusieurs jetons.

2.2.2 Exécution

L'évolution du réseau de Petri commence par l'éxecution d'une transition, les jetons se déplacent de la place en entrée de la transition à la place en sa sortie. La condition de la réalisation de ce déplacement est que la transition soit franchissable (la place en entrée dispose d'un nombre de jetons superieur ou égale aux nombres de jetons indiqué sur l'arc).

2.2.3 Représentation graphique

On représente un réseau de Petri par un graphe composé des places et transitions liés les uns aux autres grace a des arcs. Une place peut contenir des jetons (nbJetons). Dans le cas de transitions franchissables, ces jetons se transportent par le biais des transitions vers une ou plusieurs places.

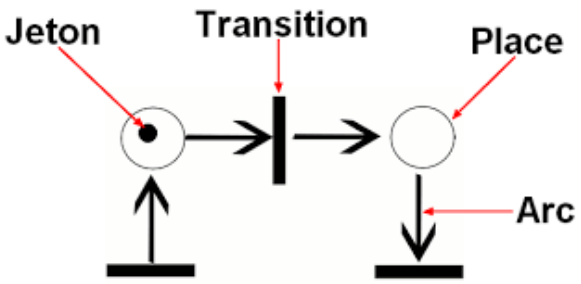

Figure 4: Représentation simple d'un réseau de petri

2.2.4 Exemple de Réseau de Petri

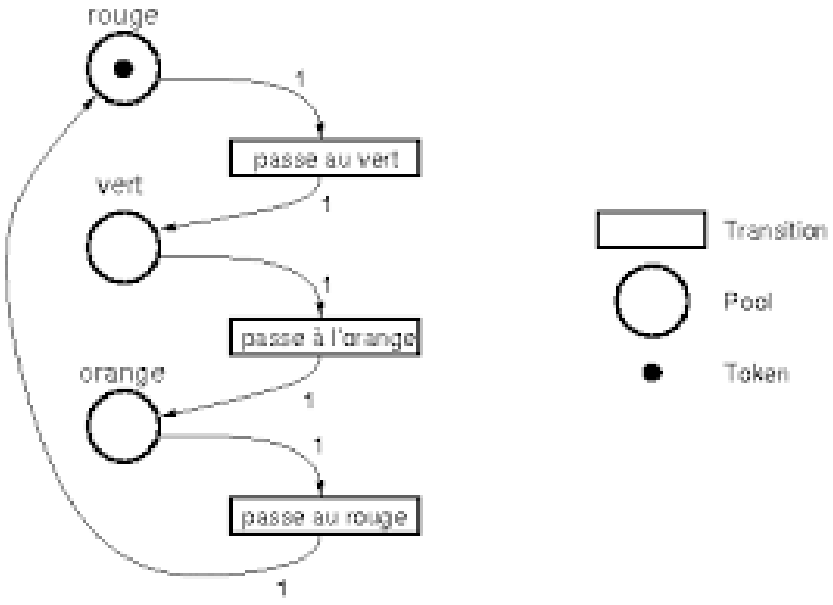

Figure 5: Exemple de réseau de petri: feu rouge.

2.2.5 Outils Ecore et EMF

On peut élaborer un métamodèle du réseau de Petri grace aux outils ECore et EMF en se basant sur la définition du réseau de Petri, ses composants principaux et l'évolution de son execution, pour ce faire, on peut se baser sur l'un des élèments suivants : — Avec un simple éditeur texte d'ECore : Syntaxe purement textuelle

— Avec l'éditeur graphique Ecore de EMF : En se basant sur la Palette contenant les outils de création des élèments ECore.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="reseauPetri" nsURI="http://www.example.org/reseauPetri" nsPrefix="reseauPetri">
  <eClassifiers xsi:type="ecore:EClass" name="PetriNet">
    <eStructuralFeatures xsi:type="ecore:EReference" name="passages" upperBound="-1"
        eType="#//Passage" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="arc" upperBound="-1" eType="#//Arc"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Passage">
    <eStructuralFeatures xsi:type="ecore:EReference" name="sortants" upperBound="-1"
        eType="#//Arc" eOpposite="#//Arc/source"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="entrants" upperBound="-1"
        eType="#//Arc" eOpposite="#//Arc/destination"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Arc">
    <eStructuralFeatures xsi:type="ecore:EReference" name="source" lowerBound="1"
        eType="#//Passage" eOpposite="#//Passage/sortants"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="destination" lowerBound="1"
        eType="#//Passage" eOpposite="#//Passage/entrants"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="jetonsConsom" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="type" eType="#//ArcType"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Place" eSuperTypes="#//Passage">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="nbJetons" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Transition" eSuperTypes="#//Passage"/>
  <eClassifiers xsi:type="ecore:EEnum" name="ArcType">
    <eLiterals name="classic"/>
    <eLiterals name="readArc" value="1"/>
  </eClassifiers>
</ecore:EPackage>
```

Figure 6: Editeur Textuelle (ECore) du reseauPetri

— Avec l'éditeur arborescent Ecore d'EMF : Forme Arborescente

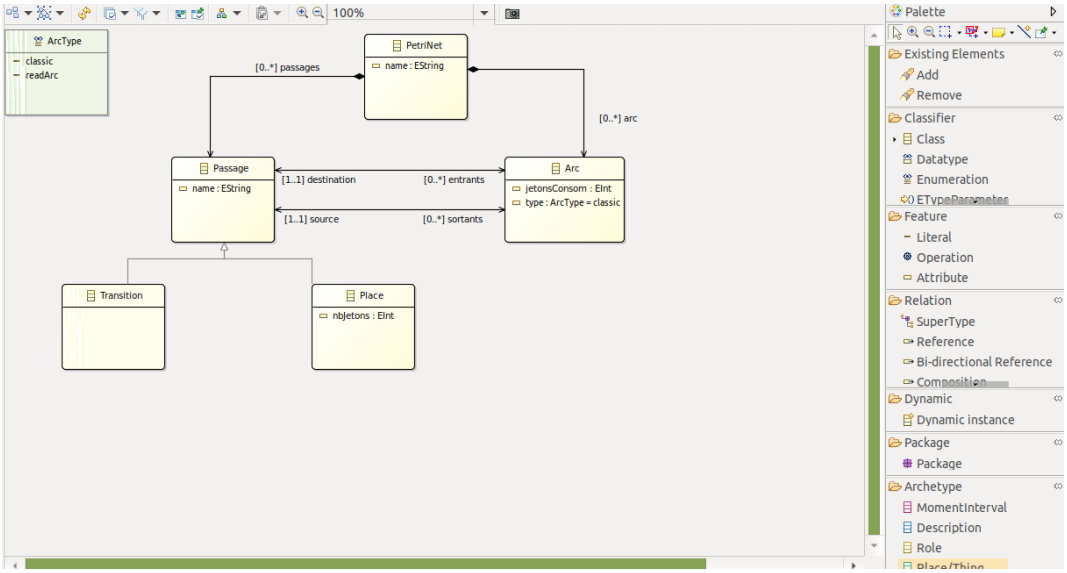

Figure 7: Editeur graphique (Ecore) du reseauPetri

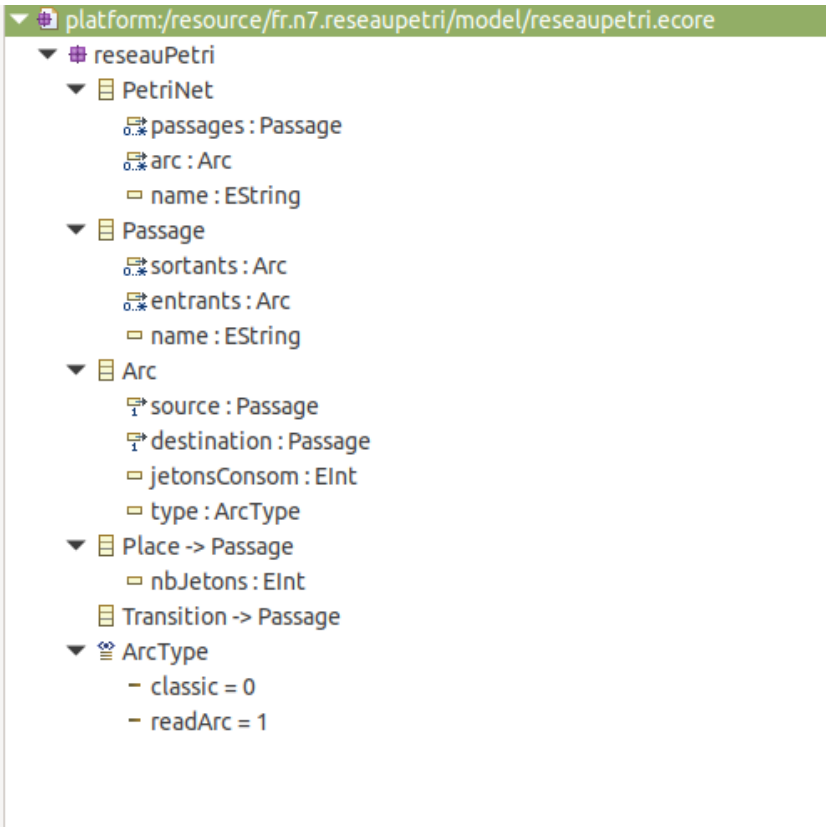

Figure 8: Editeur Arborescent (Ecore) du reseaupetri

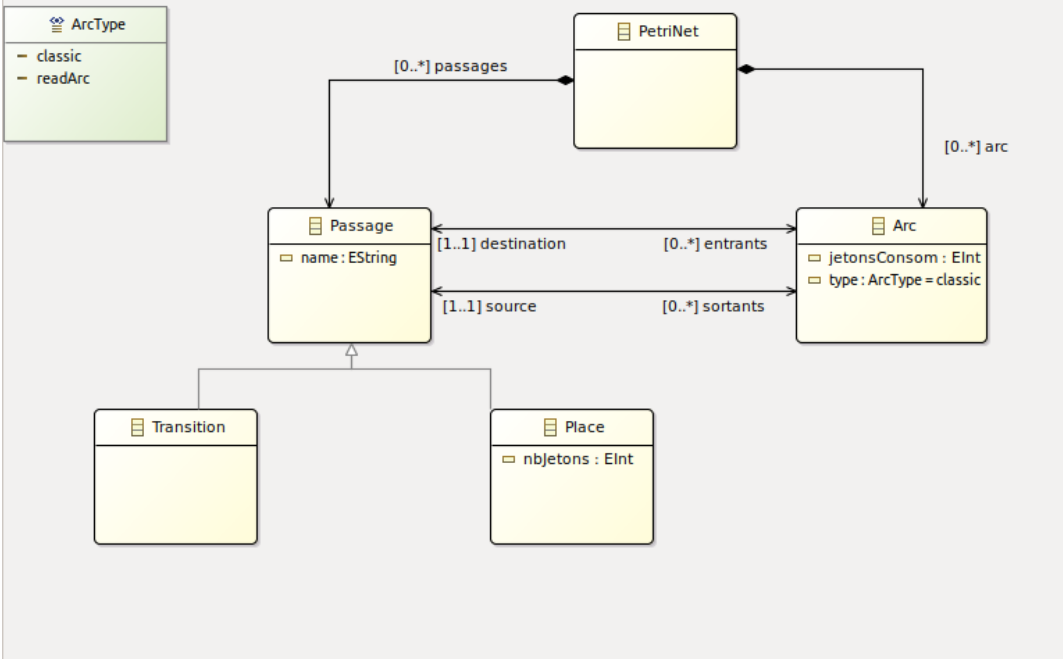

Figure 9: Le métamodèle PetriNet

2.3 Editeur graphique SimplePDL

La syntaxe concrète graphique fournit un moyen de visualiser et éditer un modèle. Nous allons utiliser l'outil Sirius. Il permet de définir une syntaxe graphique pour un langage de modélisation décrit en Ecore et d'engendrer un éditeur graphique intégré à Eclipse.

La troisième étape de ce mini projet consiste à développer un éditeur graphique SimplePDL pour saisir graphiquement un modèle de processus, y compris les ressources.

Pour cela, nous avons créé un projet (.design) sous Sirius. On a ajouté les outils de création d'éléments et de création de liens. Ainsi, l'utilisateur peut ajouter sur l'arborescence des WorkDefinitions, des WorkSequences, des Guidances...

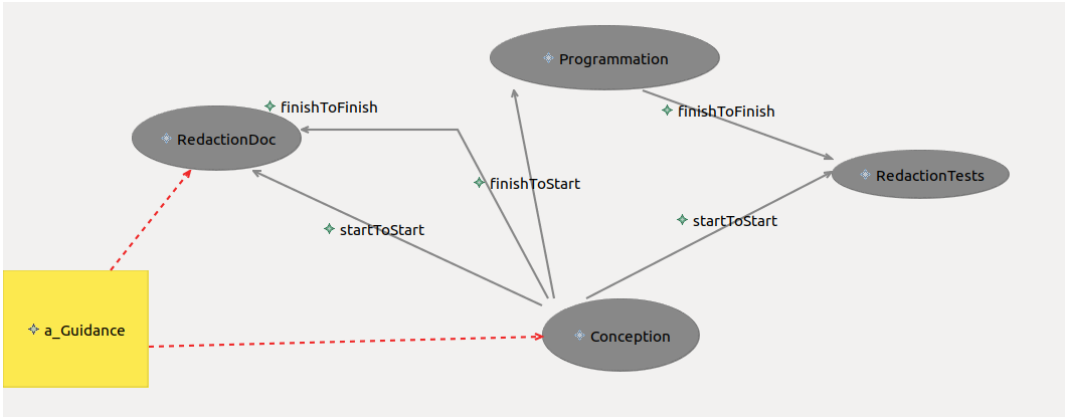

Figure 10: Edireur graphique SimplePDL

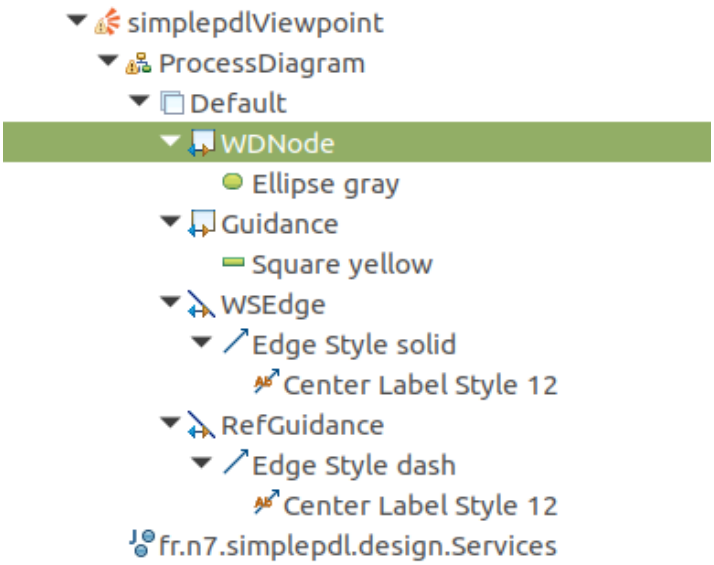

Figure 11: Outils de création pour l'éditeur graphique

2.4 Contraintes OCL

Nous avons utilisé Ecore pour définir un méta-modèle pour les processus. Cependant, le langage de méta-modélisation Ecore ne permet pas d'exprimer toutes les contraintes que doivent respecter les modèles de processus.

D'ou le role des contraintes OCL, avec qui on complète la description structurelle; sémantique statique du méta-modèle.

Le méta-modèle Ecore et les contraintes OCL définissent la syntaxe abstraite du langage de modélisation considéré.

2.4.1 Contraintes OCL pour le modèle SimplePDL

Pour le modèle SimplePDL , les contraintes sont : - Le nom d'un Process ne peut pas être vide et doit être unique : validName - deux activités différentes d'un même processus ne peuvent pas avoir le même nom : Pour chaque activité WorkDefinition On fait le parcours de tous les ProcessE- lements, on selectionne ceux du type WorkDefinition et on réalise le test manifestant que toute WorkDefinition autre que la présente possède un identifiant différent. - Une dépendance ne peut pas être réflexive : Une WorkSequence ne peut pas lier deux activités identiques. - Un Process doit avoir un nom bien définie : à savoir non nul, et dont la taille est supérieure strictement à 1. - Pour qu'une Resource soit

```
import 'SimplePDL.ecore'

package simplepdl

context Process
inv warningSeverity: false
inv withMessage('Explicit message in process ' + self.name + ' (withMessage)'): false
--inv errorSeverity: null

context Process
inv validName('Invalid name: ' + self.name):
    self.name.matches('[A-Za-z_][A-Za-z0-9_]*')

context ProcessElement
def: process(): Process =
    Process.allInstances()
        ->select(p | p.processElements->includes(self))
        ->asSequence()->first()

context WorkSequence
inv successorAndPredecessorInSameProcess('Activities not in the same process : '
        + self.predecessor.name + ' in ' + self.predecessor.process().name+ ' and '
        + self.successor.name + ' in ' + self.successor.process().name
):
    self.process() = self.successor.process()
    and self.process() = self.predecessor.process()

context WorkDefinition
inv uniqNames: self.Process.processElements
                ->select(pe | pe.oclIsKindOf(WorkDefinition))
                ->collect(pe | pe.oclAsType(WorkDefinition))
                ->forAll(w | self = w or self.name <> w.name)

context WorkSequence
inv notReflexive: self.predecessor <> self.successor

context Process
inv nameIsDefined: if self.name.oclIsUndefined() then false
                   else self.name.size() > 1
                   endif

context AllocationRessource
inv ressourceSuffisante:
        self.quantiteRessources <= self.ressources.quantity


endpackage
```

Figure 12: Les contraintes OCL pour le métamodèle Simplepdl

```
import 'reseaupetri.ecore'

package reseauPetri

context Arc
inv MemeReseauDePetri('l arc, la source et la destination doivent appartenir au meme réseau de petri'):
    self.source.PetriNet = self.destination.PetriNet = self.PetriNet

context Place
inv nmbrJetonsPositif('Le nombre de jetons dans une place doit etre positif ou nul'):
    self.nbJetons >= 0

context Arc
inv TransitoPlace('la source et la destination sont différents'):
    self.source <> self.destination

context Arc
inv typeTransiToPlace('Une transition doit etre suivie nécessairement par une place'):
    self.source.oclType() <> self.destination.oclType()

context Arc
inv nmbrJetonsConsomPositif('Le nombre de jetons à consommer (sur un arc) doit etre positif ou nul'):
    self.jetonsConsom > 0

context Arc
inv SourcePlace('La source d un ReadArc doit etre nécessairement une place'):
    if self.type = readArc then self.source.oclType() = Place
    else true
    endif


endpackage
```

Figure 13: Les contraintes OCL pour le métamodèle reseauPetri

définie, il faut que son nombre soit strictement positif - Une activité ne peut pas demander un nombre d'occurences d'une ressource supérieur aux occurences que peut offrir cette ressource à la base.

2.4.2 Contraintes OCL pour le modèle ReseauPetri

Pour le modèle Petrinet, les contraintes sont : - Toute Place doit avoit un nombre entier naturel de Jetons, en fait, ceci découle de la définition du réseau de Petri. - Le nombre de Jetons que transporte un arc doit etre un entier naturel. - Un arc ne peut pas lier deux Places, en fait, il lie une Place avec une Transition.

2.5 Syntaxe concrète textuelle de SimplePDL

Dans cette partie, on va s'interesser aux transformations d'un modèle en un texte. On parle de transformation modèle vers texte. Plus précisèment pour notre mini projet, on va générer une syntaxe textuelle à partir du métamodèle SimplePDL.

2.6 Transformation SimplePDL vers PetriNet (EMF/Java)

Dans cette étape nous devions définir une transformation SimplePDL vers PetriNet en utilisant EMF/Java, c'est-à-dire traduire les éléments de SimplePDL (Process, WorkDefinition, WorkSequence...) en élément du réseau de Petri.

2.7 Transformation SimplePDL vers PetriNet (ATL)

ATLAS Transformation Language (ATL) est un langage de transformation de modèles. Il est disponible en tant que plugin dans le projet Eclipse.

ATL se compose : — D'un langage de transformation (déclaratif et impératif) ; — D'un compilateur et d'une machine virtuelle ; — D'un IDE s'appuyant sur Eclipse ;

Nous allons utiliser ATL pour convertir un simplepdl en reseauPetri de façon beaucoup plus efficace.

On définit donc les règles pour passer d'un processus de SimplePDL à un réseau de Pétri. Pour traduire une WorkDefinition par un réseau de Pétri nous avons besoin de quatres places (Ready, Started, Running, Finished), 2 transition (Start et Finish) et 5 arcs. Les WorkSequences sont traduites par des readArc qui dépendent de leur nature.

```
[comment encoding = UTF-8 /]
[module toTina('http://www.example.org/reseauPetri')]


[template public reseauPetriToTina(aPetriNet : PetriNet)]
[comment @main/]
[file (aPetriNet.name + '.net' , false, 'UTF-8')]
net [aPetriNet.name/]

[let Defplace : OrderedSet(Place) = aPetriNet.getPlaces() ]
[for (p: Place | Defplace) ]
    pl [p.name/] ([p.nbJetons/])
[/for]
[/let]
[let Deftransition : OrderedSet(Transition) = aPetriNet.getTransitions() ]
[for (t:Transition | Deftransition)]
    tr [t.name/] [for (a : Arc |aPetriNet.arc)][if (a.destination = t)][a.source.name/][writeArcType(a)/][/if][/for] -> [for (a : Arc |aPetriNet.arc)][if (a.source = t)
[/for]
[/let]

[/file]
[/template]

[query public getPlaces(p: PetriNet) : OrderedSet(Place) =
        p.passages ->select( e | e.oclIsTypeOf(Place) )
        ->collect( e | e.oclAsType(Place) )
        ->asOrderedSet()
 /]

[query public getTransitions(p: PetriNet) : OrderedSet(Transition) =
        p.passages ->select( e | e.oclIsTypeOf(Transition) )
        ->collect( e | e.oclAsType(Transition) )
        ->asOrderedSet()
 /]

[template public writeArcType(arc : Arc)]
[if (arc.type = ArcType::classic and arc.jetonsConsom > 1)]
*[arc.jetonsConsom/][elseif (arc.type = ArcType::readArc)]
?[arc.jetonsConsom
/][/if]
[/template]
```

Figure 14: ransformation modèle à texte de PetriNet vers Tina

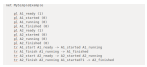

Figure 15: exemple du tp transformé en tina

En d'autres termes, pour une WorkSequence de type "StartToStart" le readArc relie la place Started à la place transition Start, une de type "StartToFinish" relie la place Started à la place transition Finish, une de type "FinishToStart" relie la place Finished à la place transition Start et une de type "FinishToFinish" relie la place Finished à la place transition Finish. Les ressources sont des Places (avec Nom et nbJetons) et les allocationRessources des Arcs pondérés allant d'une ressource vers une place de transition Start ou d'une place de transition Finish vers une ressource.

2.8 Valider la transformation

Dans la partie de la validation de la transformation, on doit valider la transformation SimplePDL vers PetriNet en faisant des tests. Nous avons donc pris l'exemple SimplePDL du sujet (avec les ressources) et nous l'avons transformé en réseau de Pétri.

2.9 Transformation PetriNet vers Tina en utilisant Acceleo

Le but de cette tâche était de définir une transformation modèle à texte de PetriNet vers Tina en utilisant Acceleo. La syntaxe textuelle voulue est celle utilisé par les outils de Tina, à savoir la syntaxe en extention .net . Ainsi, La boite à outils Tina va nous permettre ensuite de visualiser graphiquement le modèle et de le simuler avec l'outil nd (Net Draw).

Nous avons appliqué cette transformation à un exemple simple du TP (aprés avoir généré le out de java)

Et aprés on l'a visualisé par l'outil circo

Sur un deuxième exemple plus compliqué: SimplePDL qu'on a créé à partir du sujet on a appliqué la transformation Tina (aprés avoir généré le out de java)

Comme pour le premier exemple, on le visualise ensuite par circo

Ensuite par stepper simulator

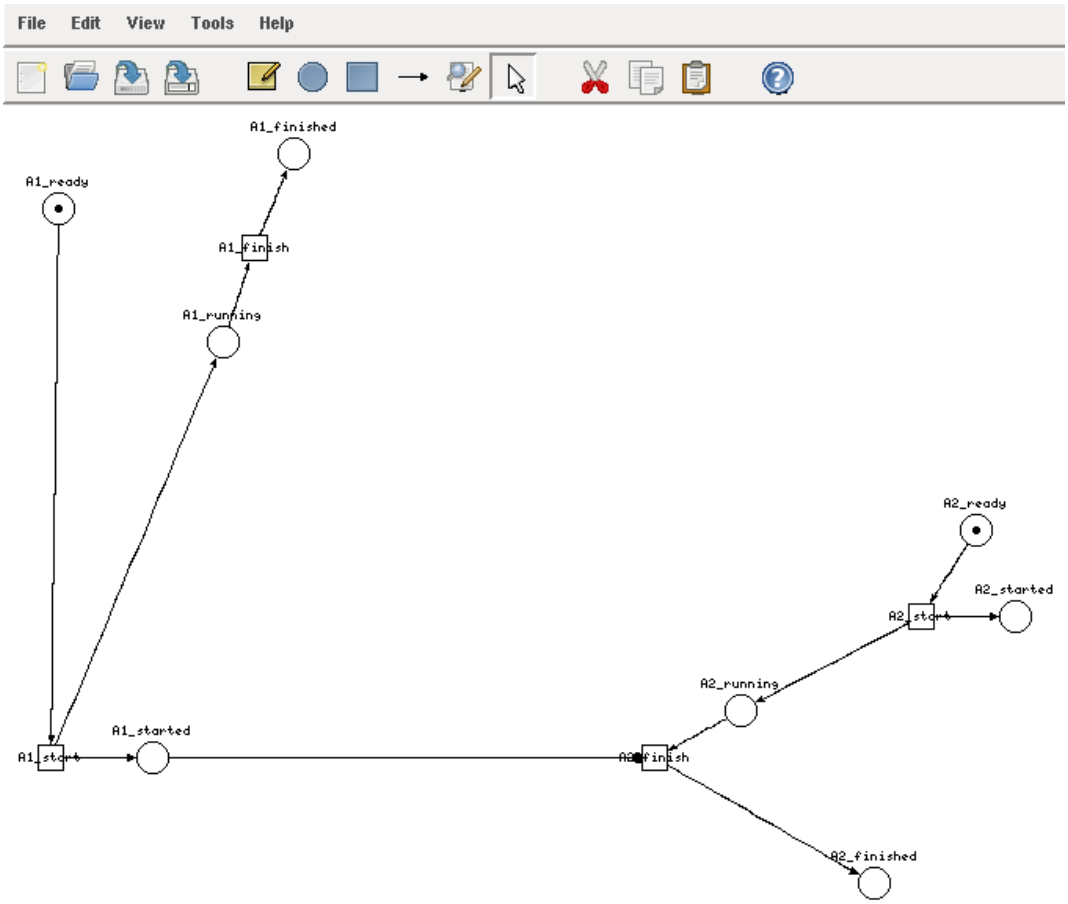

Figure 16: exemple du tp visualisé par circo

Figure 17: exemple SimplePDL transformé en tina

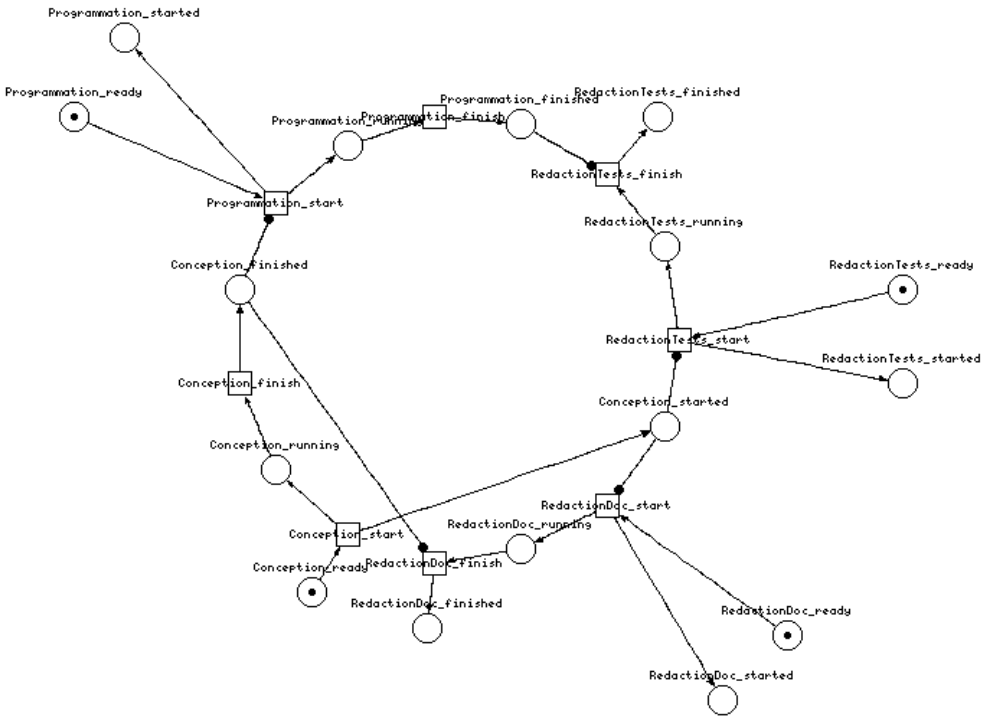

Figure 18: exemple SimplePDL visualisé en circo

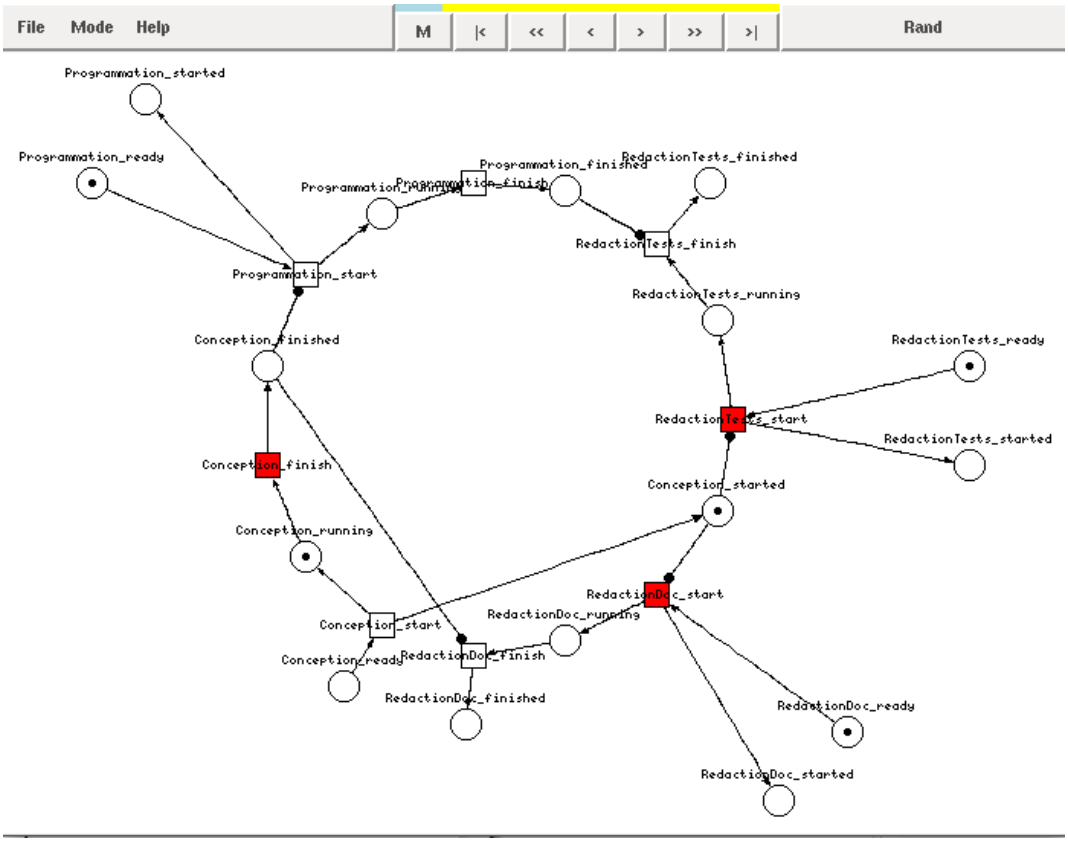

Figure 19: exemple SimplePDL visualisé par stepper simulator

Figure 20: gérération du fichier ExempleProcess1.ktz pour pouvoir vérifier les propriétés LTL par la suite

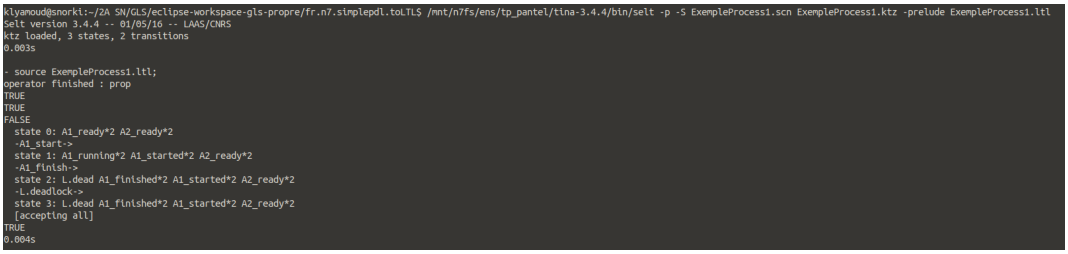

Figure 21: vérification des propriétés LTL sur l'exemple simple du TP

2.10 Propriétés LTL pour la vérification

Dans un premier temps, le but était d'engendrer les propriétés LTL permettant de vérifier la terminaison d'un processus et les appliquer sur différents modèles de processus.

Pour cela, nous avons créé un projet Simplepdl.ToLTL et nous avons définis ces propriétés :

Ensuite, nous avons donc placé ces contraintes dans un fichier (.mtl) :

2.11 Valider la transformation écrite

Les règles nous permettant de valider la transformation écrite sont : - Si le jeton est dans la place Ready, il n'y a pas de jeton dans les places Started, Running et Finished. - S'il n'y a pas de jetons dans la place Ready, il y a un jeton dans la place Started, et un dans la place Running ou Finished.

2.12 Conclusion

Ce mini projet nous a montrer l'importance de la modèlisation dans la résolution des problèmes et de manipuler les différentes notions que l'on y trouve (modèle, métamodèles, transformations). Et finalement, il nous a permis de nous familiariser avec les outils comme Eclipse, Ecore, Acceleo où Sirius qui sont assez complexe à manipuler.