Rapport projet IDM

GRETHEN Clémentine WEISBECKER Lisa

Sciences du numérique-IMM M1
2022-2023

Table des matières

Table des figures

1 Introduction

Le but de ce mini-projet est la création d'une chaîne de vérification des modèles de processus SimplePDL pour vérifier sa cohérence et s'il termine. Pour cela nous utiliserons la boîte à outil TINA. Nous traduirons donc ce modèle de processus en un réseau de pétri.

2 Création des Méta-modèles avec Ecore

2.1 SimplePDL

Le modèle SimplePDL dispose d'activités, de dépendances, de ressources et guidance. Pour ce mini-projet nous avons donc rajouté les ressoures.

Choix pour l'ajout des ressources :

Nous avons crée deux classes : RessourcesRequises et Ressource. La classe Ressource fait partie du processus, hérite de ProcessElement. Elle a comme attributs un nom (String) et une quantité (int). RessourcesRequises représente la quantité de ressources nécessaire pour qu'une activité se réalise.

On a différents liens entre toutes nos classes :

On a un lien de WorkDefinition vers RessourcesRequise : c'est un lien de composition car RessourceRequise caractérise les ressources nécessaires à une WorkDefinition. Une RessourceRequise n'est associée qu'à une WorkDefinition.

On peut visualiser le modèle SimplePDL et on obtient la figure suivante :

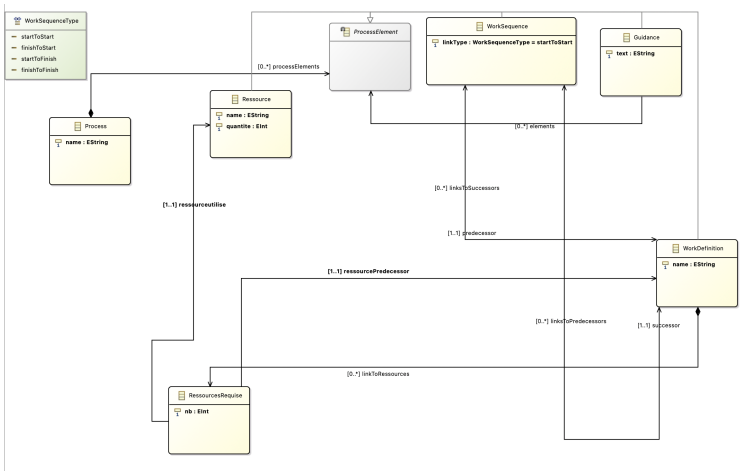

FIGURE 1 – Méta-modèle SimplePDL.

Fichiers à consulter pour cette partie : SimplePdl.ecore (méta-modèle) ; SimplePdl.aird (représentation graphique).

2.2 PetriNet

Le modèle PetriNet contient 3 élements : des places,des transitions et des arcs qui relient une place à une transition ou une transition à une place. Une place peut avoir un nombre positif ou nul de jetons.

Choix pour la construction :

Nous nous sommes inspirés du travail réalisé pour le métamodèle SimplePDL. Nous avons tout d'abord créer la classe ReseauPteri qui est un réseau de pétri. Il contient des éléments, d'où la création de la classe petriElement, reliée à ReseauPetri par composition. Les différents éléments sont ceux cités précédemment. Place et Transition héritent de Boite car elles ont les mêmes propriétés, sauf que place peut contenir des jetons. On ajoute un nom à Boite car Place et Transition ont un attribut nom. On a également un lien entre Boite et Arc, qui a été fait par symétrie à celui entre WorkSequence et WorkDefinition.

On peut visualiser le modèle PetriNet sur la figure suivant :

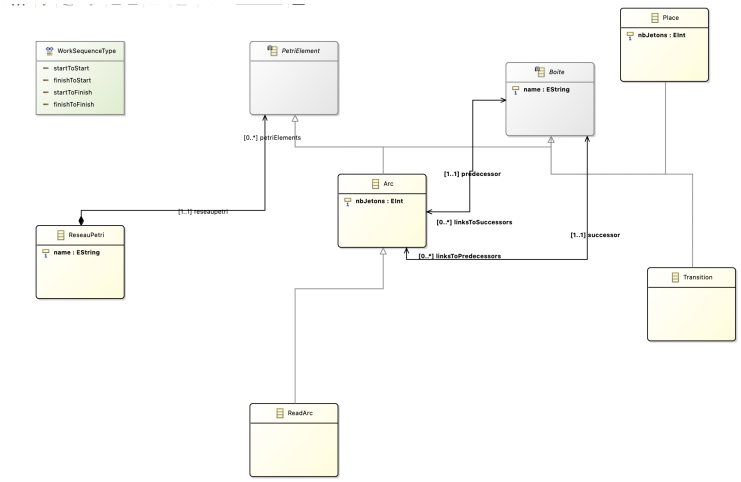

FIGURE 2 – Méta-modèle PetriNet

Fichiers à consulter pour cette partie : PetriNet.ecore (méta-modèle) ; PetriNet.aird (représentation graphique).

3 Sémantique statique avec OCL

3.0.1 Pour le cas de SimplePDL

Nous allons définir des contraintes OCL pour le méta-modèle SimplePDL.
Nous prenons les mêmes contraintes que celles des tp. Le choix que nous avons fait concerne le cas des ressources : les quantités doivent être positives et les quantités de RessourcesRequises toujours inférieurs aux quantités de la ressource associée.

Fichier à consulter pour cette partie : SimplePDL.ocl

3.0.2 Pour le cas de PetriNet

Pour vérifier le bon fonctionnement du réseau de Petri nous avons plusieurs contraintes OCL. Premièrement, tous les élèments de même type doivent avoir des noms différents. De plus,le nombre de jetons des places et des arcs doit être positif.

Fichier à consulter pour cette partie : PetriNet.ocl

3.0.3 Pour le cas de PetriNet

4 Transformation de SimplePDL en PetriNet

Choix de réalisation :

La traduction d'une WorkDéfinition en réseau de Pétri est :

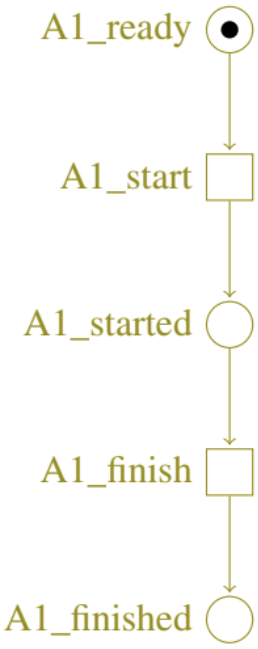

FIGURE 3 – Extrait du TD1

Rajout des WorkSequence : on rajoute une place supplémentaire. On a besoin de savoir si une activité a été commencée. On peut donc ajouter une place à côté qui mémorisera que l'activité a été commencée (TD1).

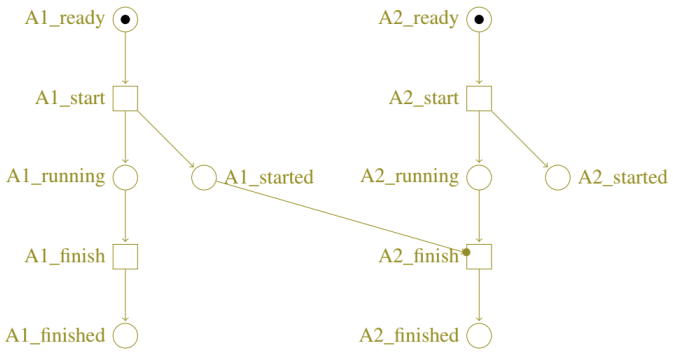

FIGURE 4 – Extrait du TD1

Enfin pour les ressources, on va créer une place ressource pour chaque ressource. Le nombre de jeton dependra de la quantité de cette ressource. Et les Ressources requises seront un arc qui va vers la transition Start de l'activité en question.

4.1 En utilisant JAVA

En utilisant le langage JAVA, nous avons pu transformer SimplePDL en PetriNet. Nous avons tout d'abord commencer par générer les classes Java nécessaires à la création de notre éditeur grâce au genmodel. Nous avons créé le fichier (adapté du tp4 en rajoutant les ressources) SimplePDLCreator.java qui nous permet de créer un exemple de modèle conforme au méta modèle SimplePDL ("SimplePDLCreator_Created_process.xmi") en l'exécutant. Nous avons donc similairement, créer PetriNetCreator.java qui permettra de créer un modèle conforme au métamodèle PetriNet (netReseauRessource.xmi) à partir de SimplePDLCreator_Created_process.xmi.

Fichiers à consulter pour cette partie : SimplePDLCreator.java, PetriNetCreator.java, SimplePDLCreator_Created_process.xmi, netReseauRessource.xmi

4.2 En utilisant ATL

Nous avons créé un projet ATL, puis complété le listing du TP8 pour créer le fichier SimplePDLtoPetriNet.atl. En exécutant ce fichier (avec la run configuration adapté) nous avons créer un Reseau de Petri à partir d'un modèle SimplePDL (in : SimplePDLCreator_Created_process.xmi) qui se nomme MonPetriNet.

5 Transformation de PetriNet vers tina en utilisant Acceleo

Pour cela, nous avons créé un projet Acceleo, puis écris le template acceleo toTina.mtl. Le principe d'acceleo est de s'appuyer sur des gabarits (templates) des fichier à engendrer, et le template se trouve dans le fichier toTina.mtl. En exécutant le fichier toTina.mtl on obtient procede.net.

Voilà les résultat obtenu :

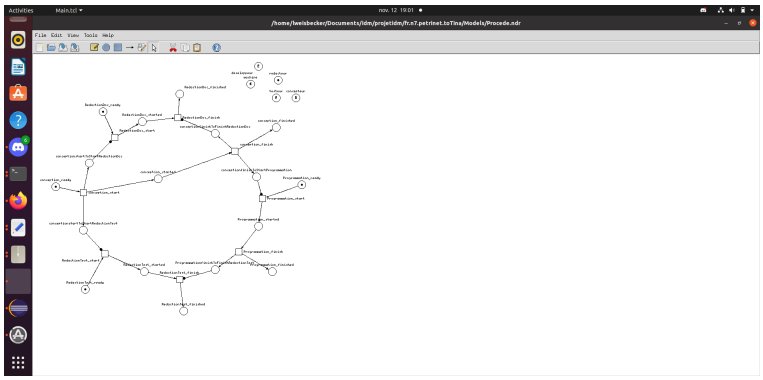

FIGURE 5 – Syntaxe graphique obtenue avec tina de la transformation M2T en en début de processus

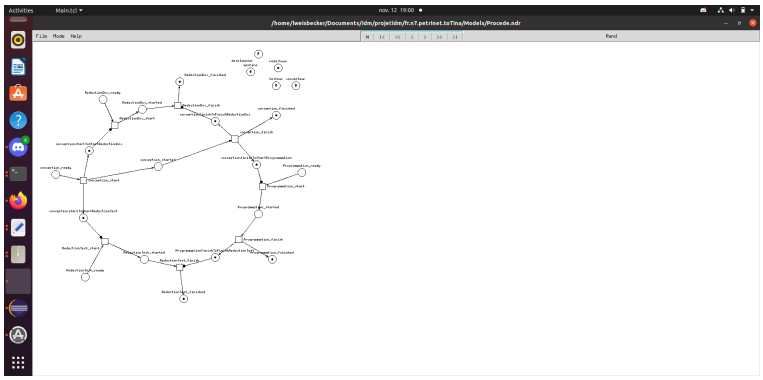

FIGURE 6 – Syntaxe graphique obtenue avec tina de la transformation M2T en en fin de processus

```
net Procédé
        pl concepteur (3)
        pl développeur (2)
        pl machine (4)
        pl rédacteur (1)
        pl testeur (2)
        pl conception_ready (1)
        pl conception_started (0)
        pl conception_finished (0)
        pl RedactionDoc_ready (1)
        pl RedactionDoc_started (0)
        pl RedactionDoc_finished (0)
        pl Programmation_ready (1)
        pl Programmation_started (0)
        pl Programmation_finished (0)
        pl RedactionTest_ready (1)
        pl RedactionTest_started (0)
        pl RedactionTest_finished (0)
        pl conceptionstartToStartRedactionDoc (0)
        pl conceptionfinishToFinishRedactionDoc (0)
        pl conceptionfinishToStartProgrammation (0)
        pl conceptionstartToStartRedactionTest (0)
        pl ProgrammationfinishToFinishRedactionTest (0)
        tr conception_start   conception_ready      -> conception_started conceptionstartToStartRedactionDoc conceptionstartToStartRedactionTest
        tr conception_finish   conception_started      -> conception_finished conceptionfinishToFinishRedactionDoc conceptionfinishToStartProgrammation
        tr RedactionDoc_start   RedactionDoc_ready      conceptionstartToStartRedactionDoc   ?1   -> RedactionDoc_started
        tr RedactionDoc_finish   RedactionDoc_started      conceptionfinishToFinishRedactionDoc   ?1   -> RedactionDoc_finished
        tr Programmation_start   Programmation_ready      conceptionfinishToStartProgrammation   ?1   -> Programmation_started
        tr Programmation_finish   Programmation_started      -> Programmation_finished ProgrammationfinishToFinishRedactionTest
        tr RedactionTest_start   RedactionTest_ready      conceptionstartToStartRedactionTest   ?1   -> RedactionTest_started
        tr RedactionTest_finish   RedactionTest_started      ProgrammationfinishToFinishRedactionTest   ?1   -> RedactionTest_finished
```

FIGURE 7 – Syntaxte textuelle de la transformation M2T

6 Validation de la transformation SimplePDL2PetriNet avec LTL

Pour vérifier nos résultats, nous utilisons le modelchecker de tina pour la logique temporelle TLT. Pour cela nous avons créé les deux fichiers Procede_inv.ltl (Pour invariants de SimplePDL pour valider la transformation écrite) et Procede_term.ltl (Pour permettre de vérifier la terminaison d'un processus). Pour vérifier la terminaison, on vérifie que toutes les activités sont bien dans l'état finished à la fin de l'évolution du processus. Voilà ce que nous obtenons(fichiers term_out.txt et inv_out.txt) :

```
lweisbecker@lweisbecker-VirtualBox:~/Downloads/tina-3.7.0-amd64-linux/tina-3.7.0/bin$ ./tina /home/lweisbecker/Documents/idm/
projetidm/fr.n7.petrinet.toTina/Models/Procede.net /home/lweisbecker/Documents/idm/projetidm/fr.n7.petrinet.toTina/Models/
Procede.ktz

# net Procede, 22 places, 8 transitions, 26 arcs                      #
# bounded, not live, not reversible                                   #
# abstraction        count      props      psets       dead       live #
#      states           26         22         26          1          1 #
# transitions           47          8          8          0          0 #
lweisbecker@lweisbecker-VirtualBox:~/Downloads/tina-3.7.0-amd64-linux/tina-3.7.0/bin$ ./selt -p -S /home/lweisbecker/
Documents/idm/projetidm/fr.n7.petrinet.toTina/Models/Procede.scn /home/lweisbecker/Documents/idm/projetidm/
fr.n7.petrinet.toTina/Models/Procede.ktz -prelude /home/lweisbecker/Documents/idm/projetidm/fr.n7.simplePDL/
modele.processus.term.ltl
Selt version 3.7.0 -- 05/19/22 -- LAAS/CNRS
ktz loaded, 26 states, 47 transitions
0.000s

- source /home/lweisbecker/Documents/idm/projetidm/fr.n7.simplePDL/modele.processus.term.ltl;
operator process_fini : prop
TRUE
TRUE
FALSE
  state 0: L.scc*25 Conception_ready Programmation_ready RedactionDoc_ready RedactionTest_ready concepteur*3 developpeur*2
machine*4 redacteur testeur*2
  -Conception_start->
  state 1: L.scc*24 Conception_started ConceptionstartToStartRedactionDoc ConceptionstartToStartRedactionTest
Programmation_ready RedactionDoc_ready RedactionTest_ready concepteur*3 developpeur*2 machine*4 redacteur testeur*2
  -Conception_finish->
  state 2: L.scc*20 Conception_finished ConceptionfinishToFinishRedactionDoc ConceptionfinishToStartProgrammation
ConceptionstartToStartRedactionDoc ConceptionstartToStartRedactionTest Programmation_ready RedactionDoc_ready
RedactionTest_ready concepteur*3 developpeur*2 machine*4 redacteur testeur*2
  -Programmation_start->
  state 3: L.scc*14 Conception_finished ConceptionfinishToFinishRedactionDoc ConceptionfinishToStartProgrammation
ConceptionstartToStartRedactionDoc ConceptionstartToStartRedactionTest Programmation_started RedactionDoc_ready
RedactionTest_ready concepteur*3 developpeur*2 machine*4 redacteur testeur*2
  -Programmation_finish->
  state 4: L.scc*8 Conception_finished ConceptionfinishToFinishRedactionDoc ConceptionfinishToStartProgrammation
ConceptionstartToStartRedactionDoc ConceptionstartToStartRedactionTest Programmation_finished
ProgrammationfinishToFinishRedactionTest RedactionDoc_ready RedactionTest_ready concepteur*3 developpeur*2 machine*4 redacteur
testeur*2
  -RedactionDoc_start->
  state 5: L.scc*5 Conception_finished ConceptionfinishToFinishRedactionDoc ConceptionfinishToStartProgrammation
ConceptionstartToStartRedactionDoc ConceptionstartToStartRedactionTest Programmation_finished
ProgrammationfinishToFinishRedactionTest RedactionDoc_started RedactionTest_ready concepteur*3 developpeur*2 machine*4
redacteur testeur*2
  -RedactionDoc_finish->
  state 6: L.scc*2 Conception_finished ConceptionfinishToFinishRedactionDoc ConceptionfinishToStartProgrammation
ConceptionstartToStartRedactionDoc ConceptionstartToStartRedactionTest Programmation_finished
ProgrammationfinishToFinishRedactionTest RedactionDoc_finished RedactionTest_ready concepteur*3 developpeur*2 machine*4
redacteur testeur*2
  -RedactionTest_start->
  state 7: L.scc Conception_finished ConceptionfinishToFinishRedactionDoc ConceptionfinishToStartProgrammation
ConceptionstartToStartRedactionDoc ConceptionstartToStartRedactionTest Programmation_finished
ProgrammationfinishToFinishRedactionTest RedactionDoc_finished RedactionTest_started concepteur*3 developpeur*2 machine*4
redacteur testeur*2
  -RedactionTest_finish->
  state 8: L.dead Conception_finished ConceptionfinishToFinishRedactionDoc ConceptionfinishToStartProgrammation
ConceptionstartToStartRedactionDoc ConceptionstartToStartRedactionTest Programmation_finished
ProgrammationfinishToFinishRedactionTest RedactionDoc_finished RedactionTest_finished concepteur*3 developpeur*2 machine*4
redacteur testeur*2
  -L.deadlock->
  state 9: L.dead Conception_finished ConceptionfinishToFinishRedactionDoc ConceptionfinishToStartProgrammation
ConceptionstartToStartRedactionDoc ConceptionstartToStartRedactionTest Programmation_finished
ProgrammationfinishToFinishRedactionTest RedactionDoc_finished RedactionTest_finished concepteur*3 developpeur*2 machine*4
redacteur testeur*2
  [accepting all]
0.001s

-
```

FIGURE 8 – Résultat pour la vérification de la terminaison

Analyse : la première propriété est vraie, cela signie qu'elle est dans un état final.La dernière est fausse : elle doit l'être car le processus va terminer dans un temps.

Pour la validation de la transformation, on a définit un invariant pour vérifier que les activitésne peuvent pas être dans deux états différents en même temps et la propriété créée est tojours vraie :

```
lweisbecker@lweisbecker-VirtualBox:~/Downloads/tina-3.7.0-amd64-linux/tina-3.7.0/bin$ ./selt -p -S /
home/lweisbecker/Documents/idm/projetidm/fr.n7.petrinet.toTina/Models/Procede_inv.scn /home/
lweisbecker/Documents/idm/projetidm/fr.n7.petrinet.toTina/Models/Procede.ktz -prelude /home/
lweisbecker/Documents/idm/projetidm/fr.n7.simplePDL/modele.processus.inv.ltl
Selt version 3.7.0 -- 05/19/22 -- LAAS/CNRS
ktz loaded, 26 states, 47 transitions
0.000s

- source /home/lweisbecker/Documents/idm/projetidm/fr.n7.simplePDL/modele.processus.inv.ltl;
TRUE
TRUE
TRUE
TRUE
0.001s
```

FIGURE 9 – Résultat pour la vérification de la transformation

7 Création des modèles Sirius décrivant l'éditeur graphique pour SimplePDL.

Nous avons utilisé l'outil Sirius dans l'éclipse de déploiement en suivant les tp et en créant les fichier simplepdl.odesign et representations.aird. Le fichier sirius.png illustre le résultat.

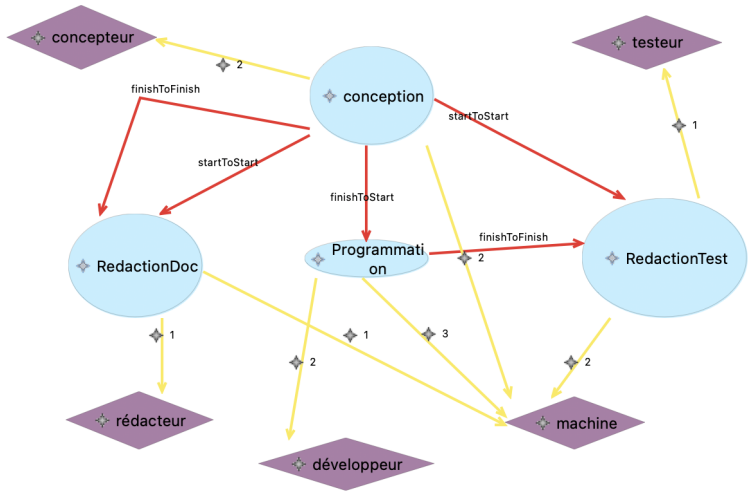

FIGURE 10 – Syntaxe graphique du modèle étudié

Pour la réprésentation graphique, nous avons choisi des ellipses bleues pour les WorkDefinitions et des losanges violets pour les ressources. les WorkSequences sont représentées par des flèches rouges entres les WorkDefinitions, labellées par le Type de lien (LinkType) : StartToStart, StartToFinish etc... Enfin, les RessourcesRequises sont réprésentées par des flèches jaunes reliant la WorkDefinition et la ressource utilisée. Elles sont labellées par le nombre de ressource nécessaire à la complétion de l'activité.

8 Conclusion

Le projet nous a permis de mettre en application et d'approfondir nos connaissances en IDM, notamment de découvrir de nombreuses fonctionnalités du logiciel Éclipse. Nous avons également pu reprendre chaque TP, les refaire et les comprendre (ce qui n'était pas forcément le cas lors des séances).

Une possibilité d'amélioration pour le projet serait la meilleure prise en compte des ressources par Tina. Elle ne semble pas être liée correctement au processus. Une explication envisageable serait une erreur lors de la définition des ressources requises dans le métamodèle SimplePDL : il aurait peut-être fallu définir un LinkType entre les ressources et les WorkDefinition.

9 Liste des fichiers du rendu

SimplePDL.ecore

SimplePDL.aird

SImplePDL.ocl

PetriNet.ecore

PetriNet.ocl

PetriNet.aird

sirius.png

PetriNetCreator.java

SimplePDLCreator.java

simplePDLCreator_Created_Process.xmi

netReseauRessource.xmi

SimplePDL2PetriNet.atl

MonPetriNet
toTina.mtl

finished.png

ready.png

procede.net

modele.processus.inv.ltl

modele.processus.term.ltl

term_out.txt

inv_out.txt

simplepdl.odesign