Rapport du projet d'IDM :

Mohammed NAHI/Achraf KHAIROUN

Département Sciences du Numérique - Deuxiéme année
2023-2024

Table des matières

1 Introduction

Ce rapport offre une synthèse complète du travail réalisé dans le cadre du mini-projet, dont les objectifs techniques sont de définir des métamodèles et des modèles conformes à ces métamodèles. L'accent est mis sur le renforcement des métamodèles en proposant des contraintes plus précises pour accroître leur robustesse, ainsi que sur la création de syntaxes concrètes textuelles ou graphiques à travers les métamodèles créés pour une meilleure visualisation des modèles. En outre, des propositions de transformations, tant du modèle vers le texte que du modèle vers le modèle, sont avancées pour convertir les modèles.

Dans le cadre de ce mini-projet, l'utilisation du langage SimplePDL est préconisée. Un métamodèle, des contraintes, ainsi qu'une syntaxe textuelle et graphique sont proposés pour ce langage. Ensuite, pour effectuer des transformations de SimplePDL vers les Réseaux de Petri (conformes à l'outil Tina), une métamodélisation des réseaux de Petri est suggérée, accompagnée de contraintes et d'une syntaxe textuelle (qui ne sera pas utilisée, car l'outil Tina propose déjà une syntaxe textuelle et une syntaxe graphique). Enfin, des transformations visent à convertir un modèle SimplePDL en un modèle PetriNet, puis à transformer ce dernier en un réseau de Petri conforme à Tina. Des transformations générant les propriétés de la Logique Temporelle Linéaire (LTL) sont également envisagées. Ainsi, ces transformations contribuent à l'objectif principal du mini-projet, à savoir vérifier la cohérence et surtout la terminaison d'un processus. Une transformation dédiée à la visualisation des modèles est également présentée.

Pour les choix techniques, l'utilisation de métamodèles avec le langage de métamodélisation Ecore est recommandée. Les contraintes sont formulées en utilisant le langage OCL. Les syntaxes concrètes textuelles sont définies avec Xtext, tandis que les syntaxes concrètes graphiques sont élaborées avec Sirius. Les transformations modèle à modèle sont implémentées en utilisant le langage Java, et les transformations modèle à texte sont réalisées avec Acceleo.

2 SimplePDL

2.1 Métamodélisation

Le simplePDL est un langage qui permet de décrire un processus de développement logiciel. Un simplePDL représente un processus qui se compose dun WorkDefinition qui décrit une activité, WorkSequence qui représente un relation entre deux activités, cette relation est de type WorkSequenceType, un Guidance qui permet de donner des remarques sur des activités et une Resource qui permet de réaliser des activités. Ces Resources peuvent être utilisés par plusieurs activités. Le métamodèle qui décrit le simplePDL est décrit dans le fichier SimplePDL.ecore et dont le diagramme est décrit dans la figure 1.

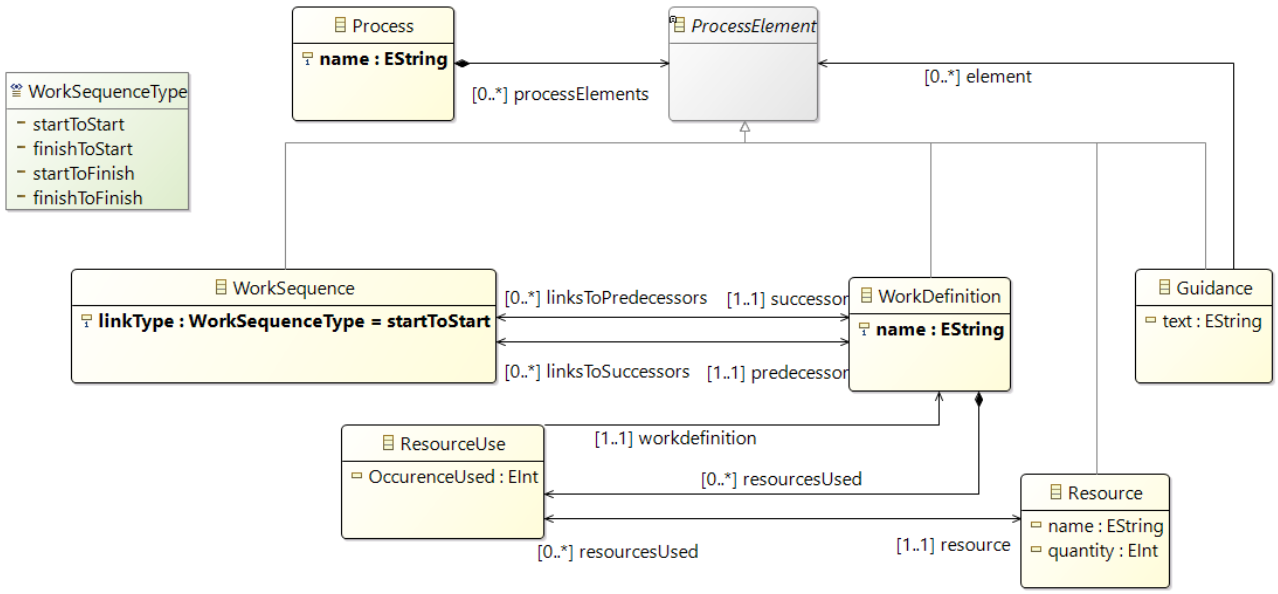

FIGURE 1 – Métamodèle SimplePDL

2.2 Contraintes OCL

Le métamodèle défini dans simplePDL.ecore nécessite un renforcement par le biais de contraintes OCL supplémentaires, visant à accroître la robustesse et la fonctionnalité des modèles générés. La section suivante propose des exemples illustrant l'importance de ces contraintes OCL.

Le fichier SimplePDL.ocl renferme des contraintes OCL spécifiques permettant d'assurer des restrictions qui ne peuvent pas être garanties par le métamodèle lui-même. Quelques contraintes cruciales sont citées :

- Le nom d'un processus doit être valide et non vide.
- Deux activités doivent appartenir au même processus.
- Une dépendance ne peut pas être réflexive.
- Deux sous-activités différentes dun même processus ne peuvent pas avoir le même nom.
- Le nom dune activité ne doit être composé que de lettres, chiffres ou soulignés, un chiffre ne peut pas être première position.
- Le nom dune activité doit être composé dau moins deux caractères.
- Loccurrence dune ressource utilisée ne peut pas prendre un nombre négatif.
- Le nom dune ressource ne doit être composé que de lettres, chiffres ou soulignés, un chiffre ne peut pas être première position.

2.3 Exemples

Le fichier Process.xmi représente le modèle correct qui satisfait toutes les contraintes, à la fois celles intégrées dans le modèle Ecore et celles spécifiées par les contraintes OCL.Les deux fichiers SimplePDLOclNonRespecte1.xmi et SimplePDLOclNonRespecte2.xmi sont deux contre exemples qui ne respectent pas les contraintes OCL.

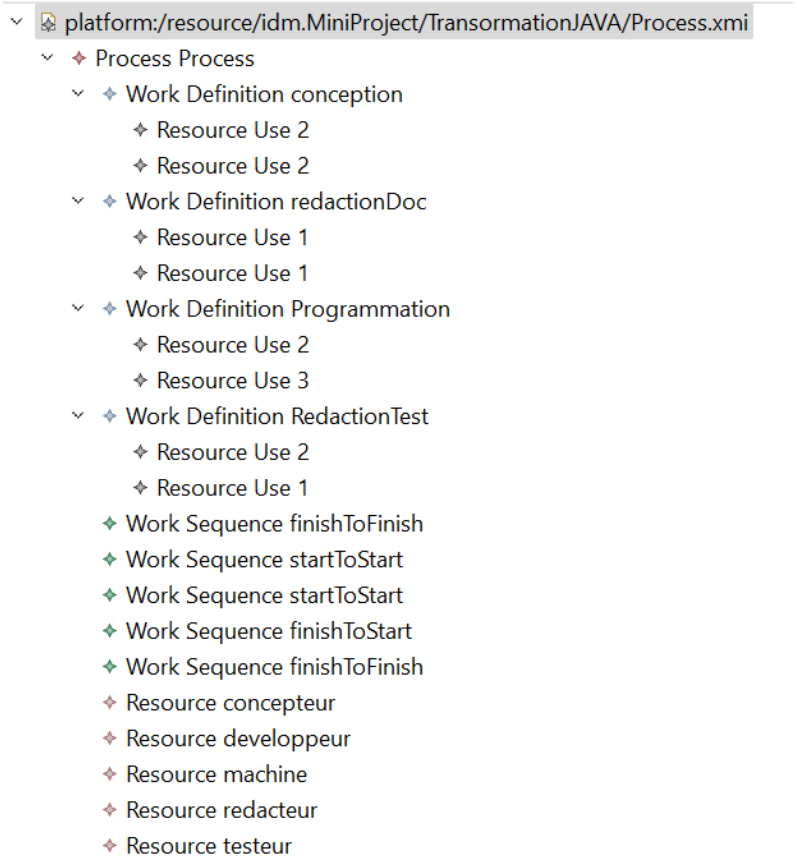

FIGURE 2 – Exemple respectant les contraintes OCL

2.4 Syntaxte Textuelle : Xtext

Le modèle précédent, conforme au métamodèle `SimplePDL.ecore`, soulève une problématique cruciale liée à la visualisation des modèles. L'éditeur arborescent s'avère insuffisant pour une compréhension optimale du modèle. Afin d'améliorer la visualisation des modèles SimplePDL, il est impératif de définir des syntaxes concrètes.

Une approche initiale consiste à représenter les modèles de manière textuelle en utilisant le plugin Xtext. Le fichier `PDL1.xtext` offre une syntaxe concrète textuelle pour décrire les processus. La Figure 3 (fichier `ProcessViaXtext.pdl`) illustre un exemple respectant cette syntaxe. Il est notable que la visualisation du même modèle est significativement améliorée grâce à l'adoption de la syntaxe textuelle.

```
process processus {
    res concepteur quantity 3
    res developpeur quantity 2
    res machine quantity 4
    res redacteur quantity 1
    res testeur quantity 2
    wd Conception {}
    wd RedactionDoc {}
    wd Developpement {}
    wd RedactionTest {}
    ws s2s from Conception to RedactionDoc
    ws s2s from Conception to RedactionTest
    ws f2f from Conception to RedactionDoc
    ws f2f from Developpement to RedactionTest
    ws f2s from Conception to Developpement

}
```

FIGURE 3 – Modèle décrit par syntaxe textuelle Xtext

2.5 Syntaxe Graphique avec Sirius

La visualisation va encore être plus améliorée avec une sytaxe concrère graphique. Une visualisation graphique des processus est présentée à laide du plugin Sirius.

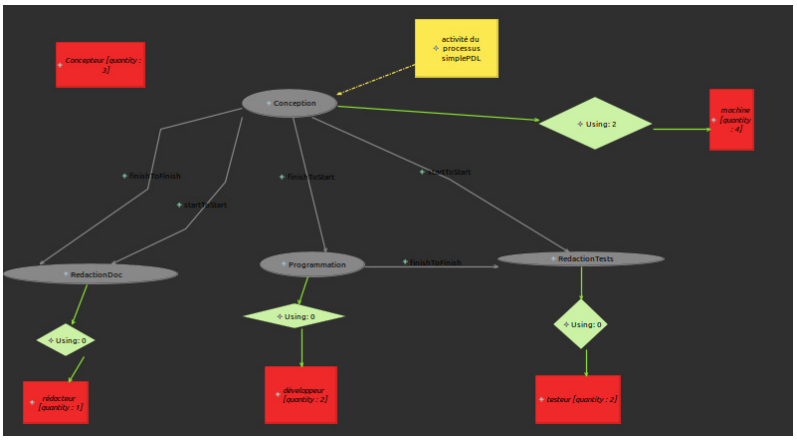

FIGURE 4 – Modèle décrit par syntaxe graphique Sirius

3 PetriNET

3.1 Métamodélisation

Pour adapter un processus à un réseau de pétri interprétable par l'outil Tina, la première étape consiste à élaborer un métamodèle de réseau de pétri, qui sera ensuite transformé en un format interprétable par Tina. Ce métamodèle, défini dans le fichier PetriNet.ecore, se compose de places, de transitions et d'arcs, où les places contiennent des jetons et les arcs ont des coûts associés. Les propriétés essentielles des réseaux de pétri, nécessaires pour garantir leur interprétation correcte, sont spécifiées soit directement dans le métamodèle, soit via des contraintes OCL détaillées dans la section suivante.

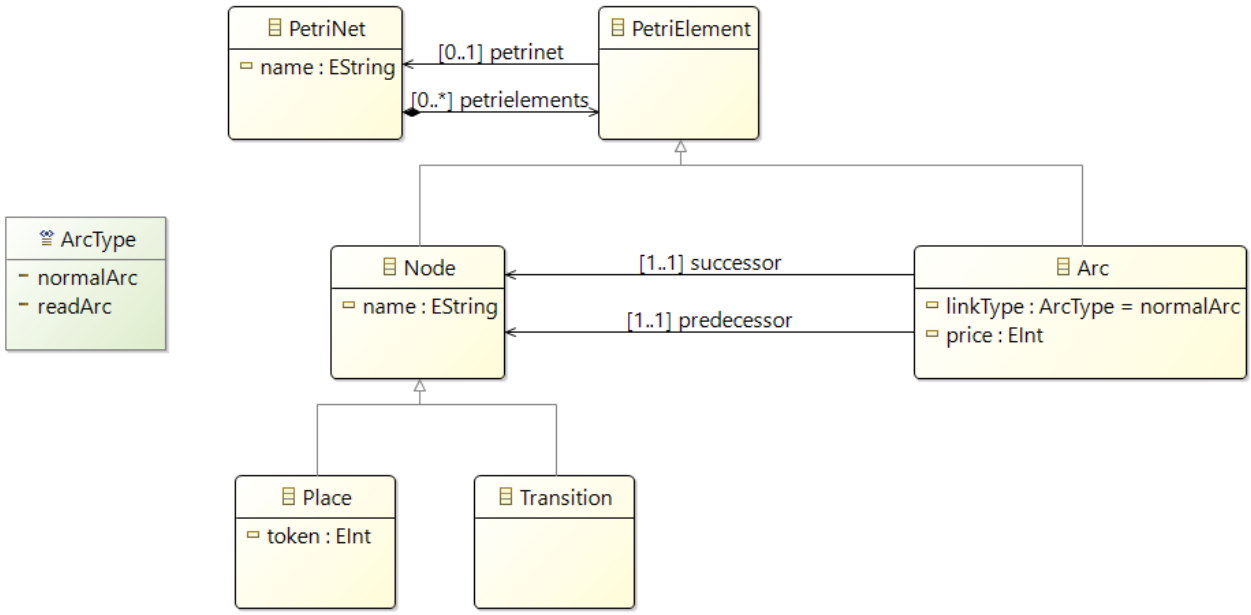

FIGURE 5 – Métamodèle PetriNet

3.2 Contraintes OCL

Le métamodèle défini dans PetriNet.ecore nécessite un renforcement à travers des contraintes OCL supplémentaires pour garantir la création de modèles robustes et plus fonctionnels. La prochaine section propose des exemples illustrant l'importance de ces contraintes OCL.

Les contraintes OCL, présentes dans le fichier PetriNet.ocl, sont essentielles pour garantir des conditions qui ne peuvent pas être assurées par le métamodèle seul. Quelques contraintes cruciales sont énumérées :

- Un PetriNet doit avoir un nom valide et non vide.
- Le nom d'une Place doit être composé dau moins un caractère et ne doit être composé que de lettres, chiffres ou soulignés, un chiffre ne peut pas être première position.
- Deux Places ne peuvent pas avoir le même nom.
- Les tokens d'une place doivent être positifs.
- Le nom d'une Transition doit être composé dau moins un caractère et ne doit être composé que de lettres, chiffres ou soulignés, un chiffre ne peut pas être première position.
- Deux Transitions ne peuvent pas avoir le même nom.
- Un Arc ne peut pas être réflexif.
- Le prédécesseur et le successeur d'un arc doivent être de types différents.
- Le coût d'un arc est au moins égal à 0.

3.3 Exemples

Le fichier Petri.xmi représente un modèle correct qui satisfait l'ensemble des contraintes, à la fois celles intégrées dans le modèle Ecore et celles spécifiées par les contraintes OCL.

4 Transformations

4.1 Transformation M2M : SimplePDL2PetriNet

4.1.1 En utilisant EMF/Java

La première transformation, considérée comme la plus complexe par les auteurs de ce rapport, s'articule autour de la conversion modèle à modèle. Ce processus consiste à transformer un modèle conforme au méta-modèle d'un processus SimplePDL en un autre modèle respectant le méta-modèle du réseau de Pétri. La mise en uvre de cette transformation est réalisée à l'aide du code Java contenu dans le fichier `SimplePDL2PetriNetViaJava.java`.

Un exemple concret de cette transformation implique la définition, pour chaque activité A, de quatre places distinctes (`A_ready`, `A_running`, `A_started`, `A_finished`) et de deux transitions (`A_start` et `A_finish`), comme détaillé dans le fichier `Petri.xmi`. Cependant, cette méthode va au-delà en introduisant une représentation des ressources dans le réseau de Pétri. Chaque ressource est représentée par une place, avec deux arcs associés : un arc autorisant une *Work Definition* à emprunter le nombre nécessaire de ressources et un autre arc de libération permettant de libérer les ressources une fois la *Work Definition* terminée.

Pour illustrer, dans le cas d'une *Work Sequence* entre deux activités, une liaison est établie via un arc de lecture (`readArc`) entre la transition d'une activité et la place de l'autre activité, selon le type spécifique de *Work Sequence*. Dans l'exemple fourni, où `A1 start2start A2`, la liaison est réalisée entre la place `A1_started` et la transition `A2_start`. Cette approche enrichie inclut une gestion explicite des ressources, renforçant ainsi la représentation du modèle *PetriNet* généré à partir du modèle *SimplePDL*.

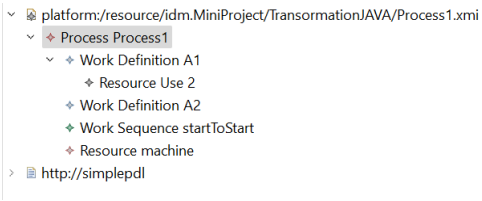

FIGURE 6 – Modèle SimplePDL

FIGURE 7 – Résultat de la transformation en PetriNet

4.1.2 En utilisant ATL

En complément de la transformation en Java, une autre méthode de transformation a été mise en oeuvre en utilisant ATL. Le code correspondant est disponible dans le fichier SimplePDLTo-PetriNetViaATL.atl. Cette méthode applique le même principe de conversion d'un modèle Sim-

plePDL vers un modèle PetriNet, offrant une alternative ou une extension à l'approche Java décrite précédemment.

4.2 Transformation M2T : PetriNet2Tina (Acceleo)

Après l'exécution du code Java, on obtient un modèle de processus conforme au métamodèle du réseau de pétri. Ce modèle passe ensuite par une transformation modèle à texte qui convertit tout modèle conforme à un PetriNet en un texte respectant la syntaxe de la boîte à outils Tina, grâce au plugin Acceleo. La transformation est définie dans le fichier toTina.mtl, et les fichiers Example.net et WorkDefinition.net illustrent un exemple qui est également présenté dans les figures 8 et suivantes.

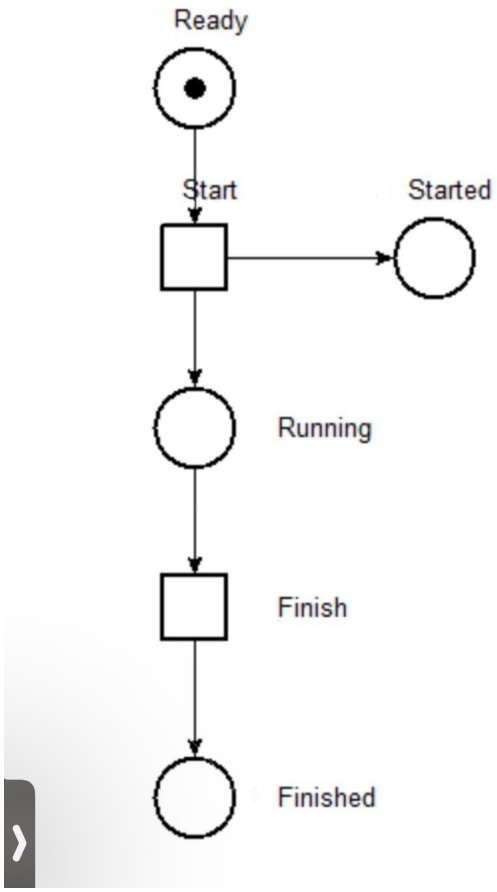

FIGURE 8 – Transformation modèle à texte par Acceleo : WorkDefinition

```
pl ready (1)
pl srunning (0)
pl finished (0)
pl started (0)
tr start ready*1 -> started*1 running*1
tr finish running*1 -> finished*1
```

FIGURE 9 – Transformation modèle à texte par Acceleo : WorkDefinition

4.3 Transformation M2T : ToDot (Acceleo)

Une autre transformation modèle à texte potentiellement utile consiste à convertir un modèle de réseau de Petri en un fichier .dot. Ce format permet de représenter graphiquement le réseau de Petri, facilitant son importation via diverses extensions telles que PDF. Cette transformation est réalisée au moyen du plugin Acceleo.

Le fichier todot.mtl propose cette transformation, et les fichiers PetriNet.pdf et WorkDefinition.pdf

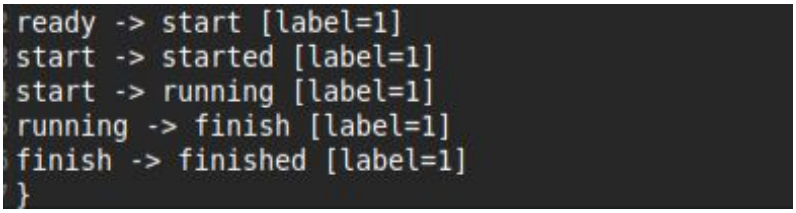

FIGURE 10 – Transformation modèle à texte par Acceleo : Dot

4.4 Transformation M2T : toLTL (Acceleo)

En utilisant des modèles de processus conformes au métamodèle de réseau de Pétri (c'est-à-dire un fichier .petrinet), il est possible de générer automatiquement les propriétés de la Logique Temporelle Linéaire (LTL) qui vérifient la terminaison du processus, plutôt que de les rédiger manuellement en fonction du modèle. C'est pourquoi nous effectuons une transformation modèle à texte qui permet de créer un fichier .ltl contenant la condition de terminaison du processus : toutes les activités doivent parvenir à l'état "finished". Le fichier toLTL contient le code de cette transformation.

FIGURE 11 – Transformation modèle à texte par Acceleo : LTL

5 Conclusion

Dans le projet d'ingénierie dirigée par les modèles, les rédacteurs de ce rapport ont tous réussi à manuvrer des outils et des modèles pour pratiquer l'ingénierie. Ce qui se révèle particulièrement captivant dans ce mini-projet, c'est la résolution d'un problème spécifique (la vérification de la terminaison d'un processus de développement logiciel). Grâce aux travaux pratiques, nous avons acquis une diversité d'outils pour résoudre cette problématique, reflétant ainsi le travail d'un ingénieur, quelle que soit sa spécialisation. D'un autre côté, le mini-projet ressemble à un assemblage de blocs Lego. À chaque notion assimilée, une pièce du puzzle est construite, et à la fin, l'assemblage de toutes ces pièces permet de répondre au problème initial. Bien sûr, des défis ont émergé au début avec chaque nouvelle notion assimilée, mais grâce à l'encadrement et à un effort individuel conséquent, nous avons acquis les compétences de base pour résoudre des problèmes similaires.