MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ Dr.YAHIA FARES DE MÉDÉA

FACULTÉ DES SCIENCES ET DE LA TECHNOLOGIE

Département du Génie Électrique et de l'Informatique



Mémoire de fin d'études

Présenté pour obtenir le diplôme de Master en Informatique

Option : Ingénierie des Systèmes et Technologies du Web

Thème

Découpage de modèles pour la vérification des diagrammes d'activité SysML

Proposé et dirigé par :

M. RAHIM Messaoud

Préparé et présenté par :

M^{elle}. OTSMANE Nabila

M^{elle}. HAMDINI meriem

Année universitaire 2015 - 2016

Remerciements

Nous tenons tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce Modeste travail.

En second lieu, nous tenons à remercier notre encadreur Mr. Rahim Messaoud ses précieux conseils et son aide durant toute la période du travail.

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail. faisaient pour moi.

Dédicace Meriem

Louages à ALLAH Qui nous a guidé sur le droit chemin tout au long du travail et nous a aspiré les bons pas et les justes réflexes. Sans sa miséricorde, ce travail n'aura pas aboutit. Très cordialement, je dédie le fruit de mes cinq ans d'étude :

Mes chères parents pour leur encouragement, leur patience, leur aide continuelle long de mon chemin d'étude et soutien financier.
A Mes grandes mères que Dieu lui accorde une longue vie.

A mon oncle Kamel et sa femme. Qui m'ont aidé à surmonter tous les obstacles que j'ai rencontrés et être à mes côtés dans les moments les plus difficiles.A mes sœurs F.Z, S, W, H, F.

A ma chère copine Nabila pour son compréhension et sa patience et à toute sa famille.
A mes amies et touts les étudiants ISTW de promotion 2015-2016.

Dédicace Nabila

Merci ALLAH (mon dieu) de m'avoir donné la capacité d'écrire et de réfléchir, la force d'y croire, la patience d'aller jusqu'au bout du rêve et le bonheur de lève mes mains vers le ciel et le dire « ya Kayoum ».

Je dédie ce modeste travaille, A mes parents , école de mon enfance, qui ont été mon ombre durant toutes mes années d'études, et qui ont veillé tout au long de ma vie, à m'encourager, à me donner l'aide et à me protéger.

À mon cher frère : Y, et mes chères sœurs : M, N.

À ma sœur et Chère amie et mon binôme « Meriem » et à tout sa famille.

À mes amies et à tous mes collègues de la section ISTW, promotion 2015-2016.

Liste des acronymes

ADT ATL Development Tool

ATL ATLAS Language Transformation

BDD Block Definition Diagram

CASE Computer Aided Software Engineering

CIM Computation Independant Model

CPNtools Colored Petri Net tools

CTL Computational Tree Logic

DFD Data Flow Diagram

DTD Document Type Definition

EMF Eclipse Modeling Framework

FFBD Functional Flow Block Diagram

GMF Graphical Modeling Framework

IBD Internal Block Diagram

IDM Ingénierie Dirigée par les Modèles

ISO International Organization for Standardization

JDOM Java Document Object Model

J2EE Java 2 Enterprise Edition

KM3 Kernel Meta Meta Model

LTL Linear Temporal Logic

MBSE Model Based Sytem Engineering

MDA Model Driven Architecture

MDE Model Driven Engineering

MOF Meta Object Facility

OCL Object Constraint Language

OMG Object Management Group

PHP Hypertext Preprocessor

PDM Platform Description Model

PIM Platform Independant Model

PNML Petri Net Markup Language

PSM Platform Specific Model

QVT Querry View Transformation

RDP Réseau de Petri.

RDPHC Réseau de Petri Hiérarchique Coloré

SysML System Modeling Language

TB Transition de substituions

UML Unified Modeling Langage

XMI XML Metadata Interchange

XML eXtensible Markup Language

ملخص

تجزئة النموذج هي العملية التي تستخرج مجموعة فرعية من نموذج لغرض معين. في هذه الأطروحة، تم الاهتمام باقتراح طريقة تستند على تجزئة النموذج للتحقق من مخطط الأنشطة $SysML$ بهدف الحد من مشكلة انفجار مساحة الحالة. بما أن لغة نمذجة الأنظمة $SysML$ تفتقر إلى دلالة منهجية لهذا الغرض قمنا بتحويل مخططات الأنشطة لشبكات بيتري $(Petri)$ ليتم تطبيق تجزئة النموذج على أساس العلاقة التي تربط بين المتطلبات و مخططات الأنشطة والكتل، بحيث تمت التجزئة وفقا للمتطلبات بغرض التحقق منها،استنادا على مخطط الانشطة $SysML$.

الكلمات المفتاحية

$SysML$ ، مخطط الأنظمة، مخطط المتطلبات، لغة نمذجة الأنظمة، شبكات بيتري $(Petri)$ ، التحقق، هندسة الأنظمة.

Résumé

Le découpage de modèle (model slicing) est une opération qui extrait un sous-ensemble d'un modèle dans un but précis. Dans ce travail, on s'intéresse à la proposition d'une approche qui se base sur le découpage de modèle pour vérifier des diagrammes d'activité SysML. L'objectif est de pallier au problème de l'explosion combinatoire de l'espace d'états. Cependant, comme SysML manque de sémantique formelle, nous translatons les diagrammes d'activité vers des réseaux de Petri hiérarchiques colorés (RDPHC). Le découpage sera appliqué au niveau du RDPHC en se basant sur les relations entre exigences, activités et blocs SysML. Le découpage sera réalisé selon l'exigence à vérifier. Pour cette étape nous exploitons le diagramme d'exigences SysML.

Mots clés :

SysML, Diagramme d'activité, RDPHC, Découpage de modèles, Explosion combinatoire de l'espace d'états, Vérification, Exigence,Bloc.

Abstract

Model slicing is an operation that extracts a subset of a model for a specific purpose. In this work, we are interested in the proposition of a Slicing Based Verification Approach for the Validation of SysML Activity Diagrams. The objective is to overcome the state explosion problem. However, as SysML lacks for formal semantics, we translate the activity diagrams to hierarchical coulred Petri nets (HCPN) on which we will apply the slicing according the SysML relationships between requirements, blocks and activities. The silicing will be made according to the requirement to verify. For this step we use the SysML requirements diagram.

Key words :

SysML, Activity Diagram, Requirements, Model Slicing, State explosion problem, HCPN, Block, Formal Verification.

Table des matières

Résumé **iii**

Introduction générale **x**

I *SysML* **1**
1 Introduction . . . 1
2 Ingénierie Système . . . 1
3 Modélisation système et SysML . . . 2
4 Diagrammes SysML . . . 3
4.1 Modélisation fonctionnelle . . . 4
4.2 Modélisation structurelle . . . 5
4.3 Modélisation dynamique . . . 9
5 Conclusion . . . 15

II Réseaux de Petri **16**
1 Introduction . . . 16
2 Présentation de réseaux de Petri . . . 16
2.1 Définition informelle des réseaux de Petri . . . 16
2.2 Définitions formelles d'un réseau de Petri . . . 17
2.3 Marquage Initial . . . 17
2.4 Dynamique des RdP . . . 19
2.5 Propriétés des réseaux de Petri . . . 22
2.6 Les réseaux de Petri de haut niveau . . . 23
2.7 Outils de modélisation des réseaux de Petri . . . 26
3 Vérification . . . 27
3.1 Techniques de vérification formelle . . . 27
4 Découpage de modèles pour la vérification . . . 28
5 Conclusion . . . 29

III Ingénierie Dirigée par les Modèles **30**
1 Introduction . . . 30
2 Présentation de l'IDM . . . 30
3 Concept fondamentaux de l'IDM . . . 30
3.1 Modélisation . . . 30
3.2 Méta-modélisation . . . 31

3.3 Approche MDA Model Driven Architecture (MDA) 33
3.4 Transformation de modèles 35
3.5 Eclipse Modeling Framework 37
3.6 ATL 38
4 Conclusion 40

IV Méthodologie **41**
1 Introduction 41
2 Vérification des spécifications SysML 41
3 Vérification des exigences sur les diagrammes d'activités 41
3.1 Démarche globale 42
3.2 Idée de découpage 42
3.3 Méthodologie 43
4 Conclusion 44

V Translation des diagrammes d'activité SysML en Réseaux de Petri **45**
1 Introduction 45
2 Transformation des DA vers les RdP 45
2.1 Méta-modèles 46
2.2 Processus de transformation 51
3 Transformation modèle-texte 59
3.1 Format CPN 59
4 Conclusion 64

VI Découpage du Modèle RDPHC **65**
1 Introduction 65
2 Conception du découpage 65
3 Implémentation 66
3.1 Manipulation des fichiers XML 66
3.2 Suppression 66
4 Conclusion 71

VII Réalisation et étude de cas **72**
1 Introduction 72
2 Spécifications SysML 72
2.1 Diagramme d'activité 72
2.2 Diagramme d'exigence 73
3 Transformation des spécifications SysML en RDPHC 74
4 Découpage du modèle 76
5 Conclusion 77

Conclusion générale **78**

Table des figures

I.1 Modélisation système 2
I.2 Positionnement de SysML par rapport à UML 3
I.3 Les diagrammes de SysML 4
I.4 Diagramme d'exigences 5
I.5 Diagramme de packages 6
I.6 Diagramme de définition de blocs 7
I.7 Diagramme interne de bloc du « Block1» 8
I.8 Diagramme de paramétrique 9
I.9 Diagramme de séquence 10
I.10 Diagramme d'état 11
I.11 Diagramme des cas d'utilisation 12
I.12 les Partitions d'activité 14

II.1 Exemple de réseau de Petri 17
II.2 Exemple de réseau de Petri 18
II.3 Exemple de marquage initial 18
II.4 Transition sensibilisée(a)et transition non sensibilisée(b) 19
II.5 Franchissement d'une transition 20
II.6 Ensemble des marquages accessibles 21
II.7 Graphe de Marquage 22
II.8 Réseau de Petri coloré 25
II.9 Réseau de Petri coloré Hiérarchique 26
II.10 Principe du model checking 28

III.1 Modele du MOF. 32
III.2 Les relations de bases dans l'IDM. 32
III.3 Le processus en Y de l'approche MDA. 34
III.4 Transformations principales de MDA 35
III.5 Schéma de base d'une transformation de modèles 36
III.6 Type de Transformation 37

IV.1 «satisfy» et «verify» relations dans les diagrammes d'exigence 42
IV.2 Démarche globale 43

V.1 Transformation de DA vers RDP 45

V.2 La partie supérieure de méta-modèle DA 47
V.3 Les partitions de l'activité 48
V.4 Les flux d'activité(ActivityEdge) 49
V.5 méta-modèle RdP 50
V.6 Les modifications effectué sur MMPNML 50
V.7 Extension Ecore 51
V.8 Concept de la transformation 52
V.9 Transformation de "Model" 53
V.10 Transformation de partition d'activité 54
V.11 Transformation de DA vers RDP DA 55
V.12 Transformation de "PetriNet" 61
V.13 Transformation de "Page" 61
V.14 Transformation de "Transition" 62
V.15 Transformation de "Transition de substitution" 62
V.16 Transformation de "Place" 63
V.17 Transformation de "RefPlace" 63
V.18 Transformation de "Arc" 64
V.19 Transformation de "Sous page" 64

VI.1 Idée de découpage 67
VI.2 Découpage représenté par RDP 68
VI.3 Fichier invalide par CPNtools 68
VI.4 Balise XML de l'arc 69
VI.5 Balise *XML* de Place 69
VI.6 Balise XML de Sub-Page 70
VI.7 Balise XML de Portsock 70
VI.8 Balise XML d'instance 70
VI.9 Balise XML de cpnsheet 71
VI.10 Balise XML de position 71

VII.1 L'activité principale $proteger_du_feu$ 73
VII.2 Les reslations «satisfay» et «verify» dans le diagramme d'exigence 73
VII.3 les interfaces graphiques 1 74
VII.4 les interfaces graphiques 2 75
VII.5 le résultat de transformation 1 75
VII.6 le résultat de transformation 2 76
VII.7 Le résultat de découpage en choisissant la partition Détection centrale qui satisfait l'exigence R 76

Liste des tableaux

4.1 Les éléments du diagramme d'activité conforme aux spécifications **OMG** . 13

2.1 Table contient le reste des règles de transformations 57

4.1 Comparaison entre espace d'états avant et aprés le découpage 77

Introduction générale

Les systèmes sont présents, aujourd'hui, dans tous les domaines de l'activité humaine (industrie, transports, communications, construction, etc.). Avec le temps, ces systèmes sont devenus de plus en plus complexes, et par conséquent, leur analyse et leur vérification représentent un enjeu capital.
L'IDM permet de réutiliser des modèles de formalisme appelés méta-modèles, qui sont adaptables à toutes les plateformes. Certains organismes s'intéressent de plus en plus à l'IDM et cherchent à la normaliser : c'est le cas de l'OMG (Object Management Group). Cette association qui est à l'origine du langage de modélisation UML, a défini le langage SysML (System Modeling Language), un langage de modélisation dédié à l'ingénierie système. Ce langage n'est pas formel, il est donc impossible d'établir des preuves sur la concordance entre la solution envisagée et les exigences élaborées. C'est pourquoi nous nous intéressons ici à la transformation des diagrammes d'activités SysML vers un langage formel, les réseaux de Petri.
Nous tenterons d'établir les concepts nécessaires pour une transformation automatique des diagrammes d'activités conforme à la spécification définie par l'OMG vers les réseaux de Petri. Pour décrire le comportement complet de systèmes complexes avec de nombreux composants, on a utilisé des diagrammes d'activité SysML en exploitant les partitions d'activité (activity partitions)qui peuvent être utilisées pour faire le découpage à fin de réduire l'explosion combinatoire de l'espace d'état, lorsque on considère la vérification des systèmes.
Nous proposons une démarche basée sur le découpage de modèles pour la vérification des diagrammes d'activités SysML. Ce découpage sera déduit à partir de l'exigence à vérifier et sera appliqué sur le modèle RDPHC.
Ce travail sera organisé en sept chapitres :

- **Chapitre I :** Ce chapitre présente le langage SysML et ses différents diagrammes. Particulièrement, nous focalisons sur la présentation du digramme d'activités et le digramme d'exigences.
- **Chapitre II :** Dans ce deuxième chapitre, nous présenterons le formalisme des réseaux de Petri, leurs extensions et les techniques de vérification à base de ce formalisme. Ensuite, nous présenterons CPNtools, un outil de vérification pour les RDPHC. Enfin, nous décrirons la technique de vérification sur modèles "Model-Checking" et le problème de l'explosion combinatoire de l'espace d'états.
- **Chapitre III :** Ce chapitre présentera le domaine d'ingénierie dirigée par les Modèles(IDM) et les différentes notions qui servent pour la transformation de modèle.

— **Chapitre IV :** Dans ce chapitre, nous décrirons notre méthodologie pour vérifier les exigences SysML sur les diagrammes d'activités en spécifiant les différentes étapes pour arriver à la phase de vérification.

— **Chapitre V :** Ce chapitre sera consacré à la proposition d'une transformation modèle-modèle des diagrammes d'activité SysML avec les partitions vers les RD-PHC.

— **Chapitre VI :** Dans ce chapitre nous allons présenter l'idée de découpage de modèle et son implémentation.

— **Chapitre VII :** Ce dernier chapitre illustre notre travail par un cas d'étude pratique et son réalisation dans le but de valider notre travail.

Chapitre I

SysML

1 Introduction

La conception de système donne souvent lieu à une accumulation de documentations qui doivent toutes être croisées et mises à jour pour maintenir la cohérence et respecter les spécifications du système. SysML est un moyen de regrouper dans un modèle commun à tous les corps de métiers, les spécifications, les contraintes et les paramètres de l'ensemble du système. SysML n'aborde plus la conception avec la notion de classes mais avec la notion des blocs qui deviendrons des parties mécaniques, électroniques, informatiques ou autres. L'objectif de ce chapitre est de montrer comment utiliser la notation SysML dans le cadre d'un processus complet partant des premiers concepts allant jusqu'à l'exploitation de ce dernier. Le passage de UML à SysML est très simple. Les diagrammes en SysML sont moins nombreux. De plus, SysML réutilise une bonne partie des diagrammes UML.

2 Ingénierie Système

L'ingénierie système regroupe l'ensemble des activités pour concevoir, développer, faire évoluer un ensemble de produits, processus et compétences humaines, apportant une solution économique et performante aux besoins exprimés. Elle ne prétend pas apporter la solution unique à un problème donné, mais propose une démarche s'appuyant sur des processus, outils et standards, pour transformer un besoin en solution. Elle s'applique à tout système artificiel, quels que soient sa complexité et son contexte d'utilisation. Afin d'illustrer les concepts, des exemples seront développés au fur et à mesure de l'introduction des concepts. Les méthodes de l'ingénierie Système (IS) reposent sur des approches de modélisation et de simulation pour valider les exigences, et pour vérifier ou évaluer le système. La modélisation a donc couramment été utilisée pour l'IS, que ce soit pour des représentations concrètes avec des plans ou modèles réduits ou plus abstraites avec des systèmes d'équations.

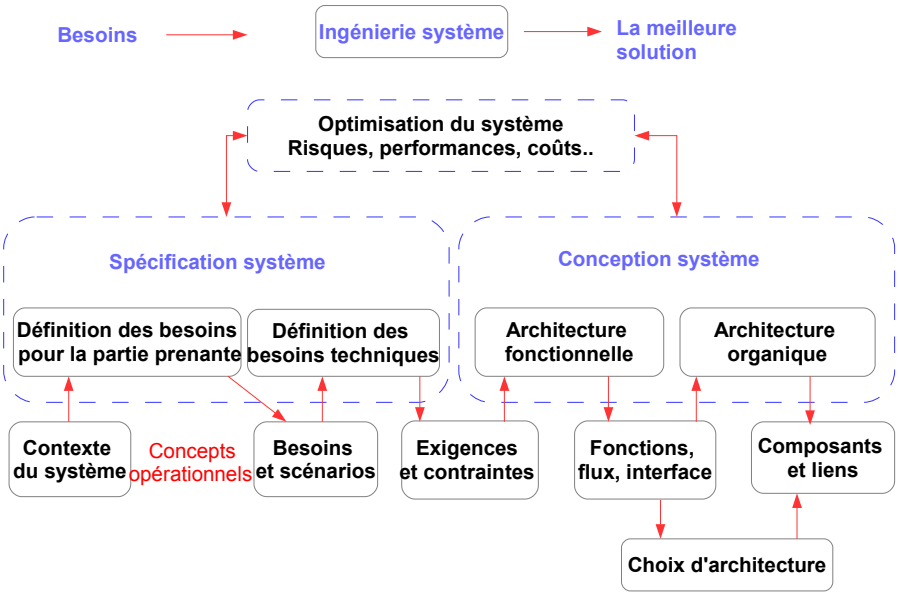


FIGURE I.1 – Modélisation système

En général, l'IS tend à modéliser tous les aspects d'un système en considérons sa décomposition fonctionnelle, son comportement et ses flux de donnés et sa décomposition structurelle. Exemples de techniques de modélisation employées :

— Le diagramme de flux de données (DFD ou Data Flow Diagram) pour définir les données traversant un système et leurs traitements éventuels.

— Le diagramme de flux fonctionnel de bloc (FFBD ou Functional Flow Block Diagram) proche du diagramme UML d'activité ou du « flowchart »[1].

3 Modélisation système et SysML

UML [2] a évolué régulièrement jusqu'à proposer des concepts permettant de faire la modélisation système. Cependant, les équipes «Système» trouvent souvent des difficultés pour utiliser UML , trop orienté logiciel . L'OMG a donc décidé d'utiliser le concept de «Profil»(ensemble de stéréotypes) proposé par UML et ainsi adapter UML pour les ingénieurs systèmes (par exemple la classe UML est devenue un bloc, en fait une classe stéréotypée «Block». Cette démarche a donné naissance au profil UML «**SysML**». La première version officielle de SysML (1.0) a été publiée par l'OMG en automne 2007. la version 1.1 est arrivée pendant l'hiver 2008. Actuellement, la version 1.3, publiée au printemps 2012. (À noter que la version 1.4 est disponible en mode « bêta » depuis mars 2014 [3] [2].) Aujourd'hui, SysML est accepté de manière générale et s'utilise de manière opérationnelle sur les projets industriels, et notamment dans les métiers critiques

de l'aéronautique et de l'espace.

L'arrivée de SysML a permis d'étendre la portée au niveau système. En effet, aujourd'hui, nous pouvons modéliser un système en SysML. SysML et UML permettent aux architectes système et logiciel de disposer d'un langage de communication commun, connu dans le monde entier [4].

4 Diagrammes SysML

SysML est basé sur UML. Il réutilise, avec une éventuelle adaptation, sept diagrammes UML (activités, séquence, états, cas d'utilisation, définition de blocs, bloc interne et package) et en propose deux nouveaux (exigences et paramétrique). La figure I.2 présente le positionnement de SysML par rapport à UML.
Les neuf diagrammes sont généralement présentés en trois groupes : comportement, exigences et structure. Les diagrammes d'activités, de définition de blocs et de bloc interne ont été modifiés de manière importante par rapport à leur version UML : cela concerne les diagrammes d'activités, de classes et de classes structurées [5].

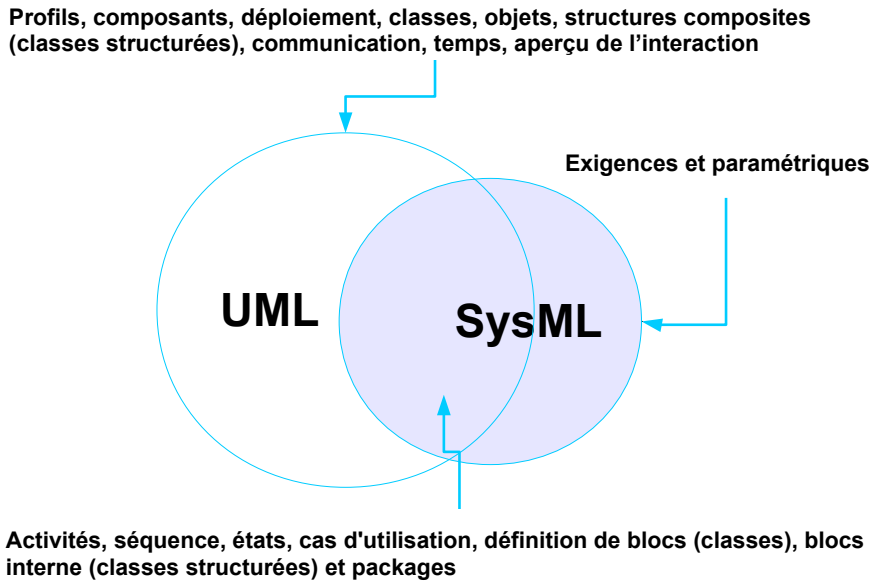


FIGURE I.2 – Positionnement de SysML par rapport à UML

Les diagrammes définis sous UML2, et ceux qui constituent SysML sont représentés ci-dessous(figure I.3). Hormis les nouveautés, la majorité des changements apportés par SysML se trouvent dans les diagrammes structurels.

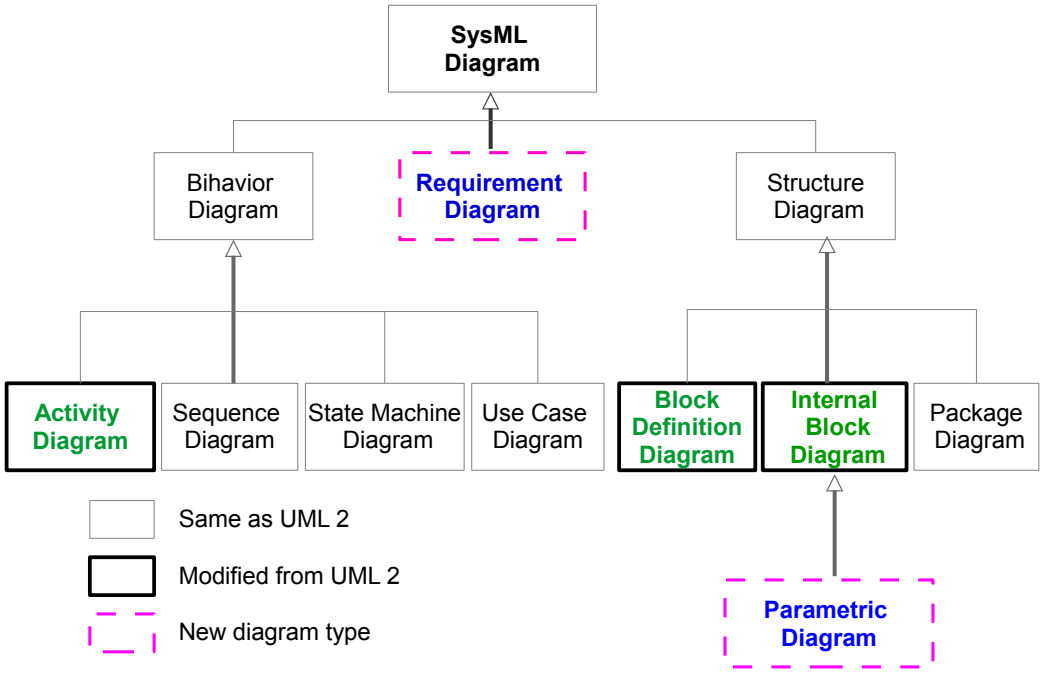


FIGURE I.3 – Les diagrammes de SysML

Ainsi SysML permet de fournir un référentiel central qui supporte les analyses du système requises par l'IS, à savoir : la décomposition fonctionnelle, les flux de données et la décomposition structurelle.

4.1 Modélisation fonctionnelle

4.1.1 Diagramme d'exigences

Le diagramme d'exigences permet de représenter graphiquement les exigences dans un modèle. Une exigence est décrite par un identifiant unique accompagné d'un texte descriptif. D'autres critères sont communément utilisés pour caractériser une exigence, par exemple : son état(proposée, validée, approuvée...), sa difficulté, sa priorité, sa version, ses dates de création et de mise à jour...[4].

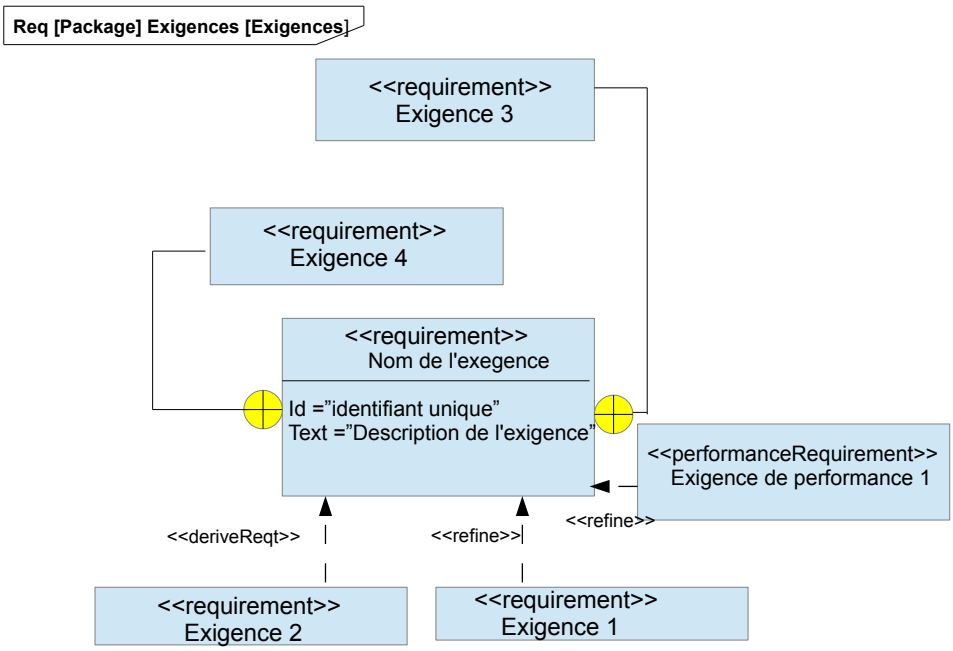


FIGURE I.4 – Diagramme d'exigences

Un exemple d'un diagramme d'exigence est présenté dans la figure I.4. Nous remarquons qu'une exigence est une classe stéréotypée avec "requirement". Aussi, des relations entre les exigences et les différents éléments de modélisation SysML peuvent être décrite.

- «**refine**» qui permet d'ajouter des précisions.
- «**derive**» qui permet de lier des exigences de niveaux différents, qui permet de décomposer une exigence en sous-exigences. Il existe d'autres relations qui peuvent être utilisées entre les blocs et les exigences :
  - «**Satisfy**» indique que le bloc satisfait l'exigence, ou encore entre les tests et les exigences.
  - «**Verify**» indique que le test vérifie l'exigence. À noter que «**Refine**» peut également être utilisé entre les exigences et les cas d'utilisation (Use Cases), car un cas d'utilisation peut raffiner une exigence.

4.2 Modélisation structurelle

SysML propose pour la modélisation structurelle quatre diagrammes :

4.2.1 Diagramme de packages

Le diagramme de packages permet de structurer le modèle en fonction de l'organisation mise en place pour le projet. Le diagramme de paquetage est la représentation graphique des relations existant entre les paquetages (ou espaces de noms) composant un système, et permet de regrouper sous une même appellation un ensemble d'éléments de modélisation tout comme avec UML. Un paquetage est susceptible de contenir n'importe quel élément de modélisation SysML et dans la pratique on utilise les paquetages pour regrouper au sein d'une même entité, un diagramme de cas d'utilisation, les diagrammes de séquence associés, le diagramme de classes et il apparait alors comme un dossier dans une arborescence de fichiers [4].
La figure I.5 présente un simple exemple de diagramme de paquetage. Dans ce type de diagramme (figure I.5), nous pouvons voir :

— des « Packages » de premier niveau tels que Étoiles, Soleil, Lune, Terre.

— de deuxième niveau (relation de contenance) : Afrique, Europe.

— les relations entre packages : Dependency, Import (une dépendance stéréotypée << >>)

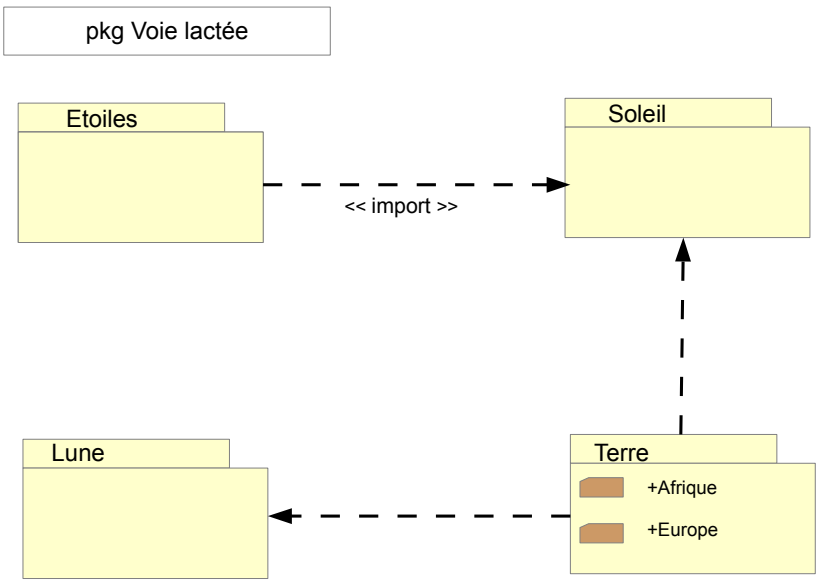


FIGURE I.5 – Diagramme de packages

4.2.2 Diagramme de définition de blocs

C'est un diagramme statique. Il montre les briques statiques : blocs, composition,associations... Il est utilisé pour décrire l'architecture matérielle du système. Un bloc est une entité

bien délimitée qui encapsule principalement des attributs(variables d'état), des opérations (procédures comportementales), des contraintes, des ports (échange de flux avec l'extérieur) et des parts (sous-blocs internes). Un bloc peut modéliser tout le système, un élément matériel ou logicielle [6].

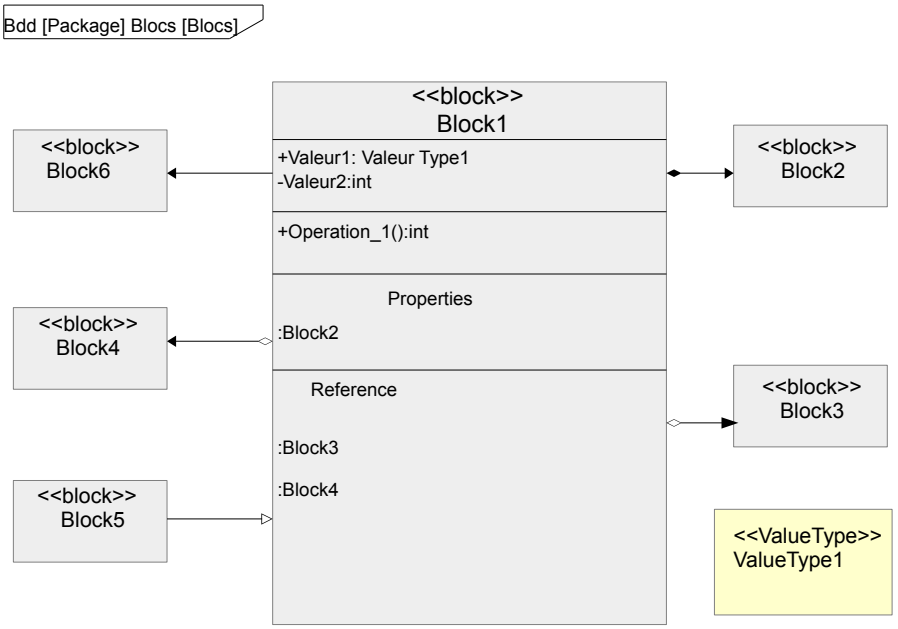


FIGURE I.6 – Diagramme de définition de blocs

4.2.3 Diagramme de bloc interne

Ce diagramme permet de modéliser la structure interne d'un bloc (identifié dans le diagramme de définition de blocs) en parties, ports et connecteurs. À noter que l'on peut représenter à travers ce diagramme plusieurs niveaux de décomposition. Ce diagramme est aussi appelé IBD (Internal Block Diagram). Le diagramme de bloc interne (Internal Block Diagram la figure I.7) est intimement lié au diagramme de définition des blocs(Block Definition Diagram la figure I.6). En effet, les relations décomposition et d'agrégation donnent naissance à des « parts » dans l'IBD. De plus, l'IBD permet l'utilisation d'un nouveau concept le« port » qui va nous permettre de modéliser la logique de connexion entre blocs. Dans la figure I.7. L'IBD permet donc de modéliser les connexions entre blocs tout en définissant le type de flux qui circule à travers les ports [4].

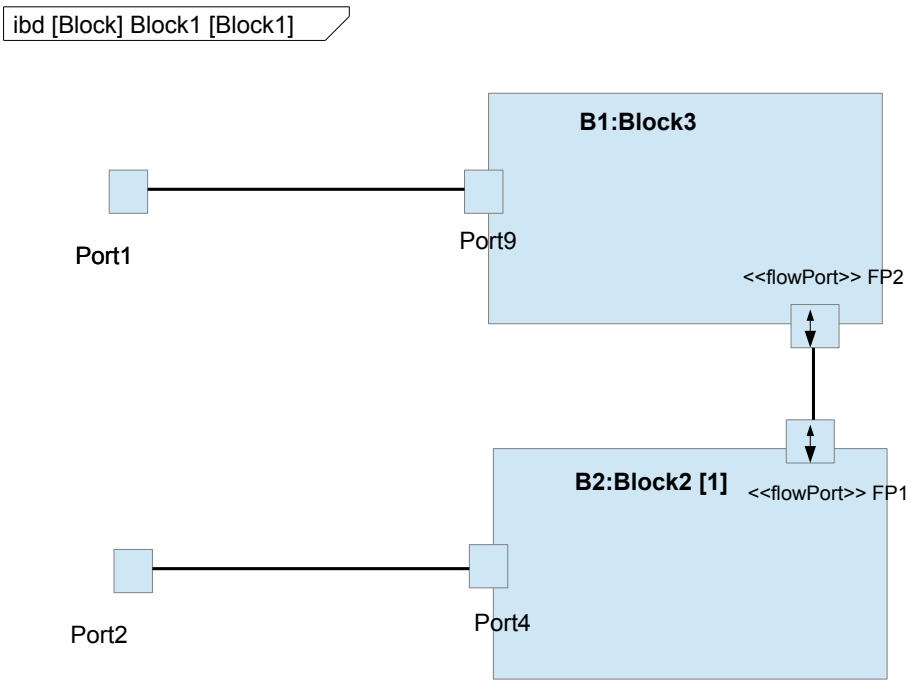


FIGURE I.7 – Diagramme interne de bloc du « Block1»

4.2.4 Diagramme paramétrique

Ce diagramme permet de modéliser des contraintes sur les valeurs des paramètres utilisés par le système. Les contraintes sont définies dans le BDD via des blocs stéréotypés «constraint» et associées au système qui doit les respecter. Une contrainte fournit des paramètres accompagnés d'une fonction de transformation (formule mathématique...)[4].

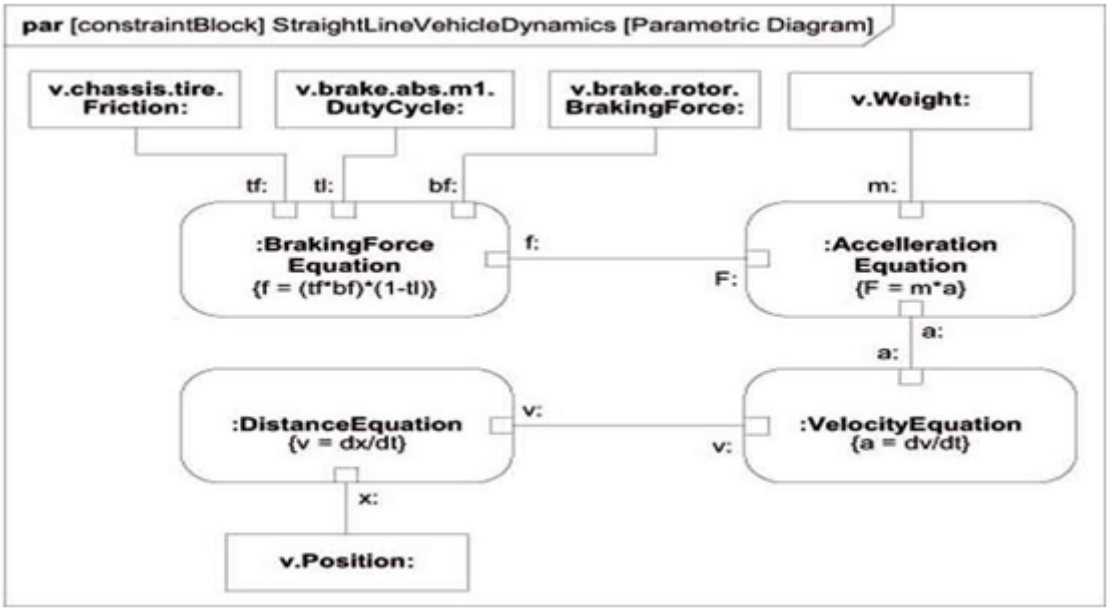


FIGURE I.8 – Diagramme de paramétrique

4.3 Modélisation dynamique

SysML propose pour la modélisation dynamique quatre diagrammes :

4.3.1 Diagramme de séquences

Le diagramme de séquences proposé par SysML est identique à celui proposé par UML. Ce diagramme est essentiellement utilisé en SysML pour modéliser les scénarios opérationnels en début de modélisation, puis pour modéliser la collaboration entre les blocs. Il représente les éléments intervenant dans le scénario, ainsi que les messages échangés dans un ordre chronologique. Dans un premier temps, on peut choisir de représenter les interactions entre l'acteur et le système. Par la suite il est possible de rentrer en détails avec un diagramme de séquence qui représente les blocs internes du système intervenant dans un scénario (pour un message émis par l'acteur, le diagramme décrit l'enchaînement des messages échangés entre les blocs internes du système). Les éléments intervenant sont représentés par des lignes de vie. Ces lignes de vies peuvent être des instances de blocs, établissant un lien avec la modélisation structurelle du système, permettant ainsi d'établir une cohérence dans le modèle représenté dans la figure I.9.

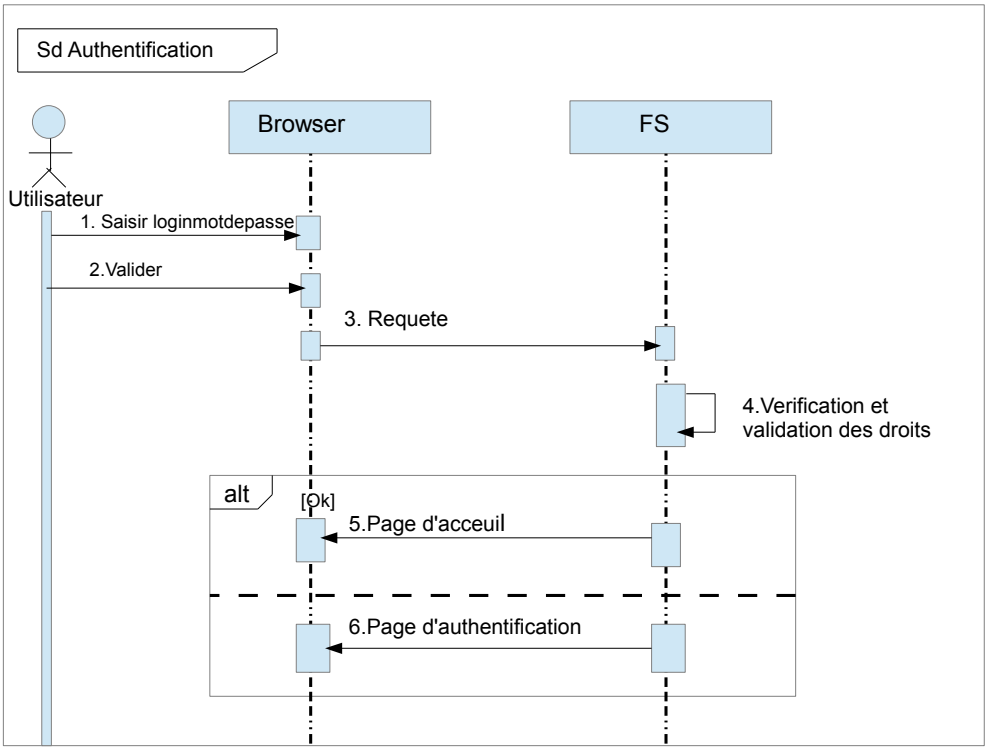


FIGURE I.9 – Diagramme de séquence

4.3.2 Diagramme d'état

C'est un diagramme dynamique. Il montre les différents états successifs et les transitions possibles des blocs dynamiques. Il représente la succession des états d'un système ou d'un sous-système. La dynamique des changements est entretenue par des événements [6].

Le diagramme d'états est utilisé avec SysML de la même manière qu'avec UML, c'est-à-dire qu'il permet de représenter le cycle de vie auquel doivent se conformer toutes les instances d'un bloc donné, ce au travers de la modélisation de tous les états possibles. Seuls les blocs qui sont importants d'un point de vue métier, ou qui sont de nature complexe, devraient avoir un diagramme d'état.

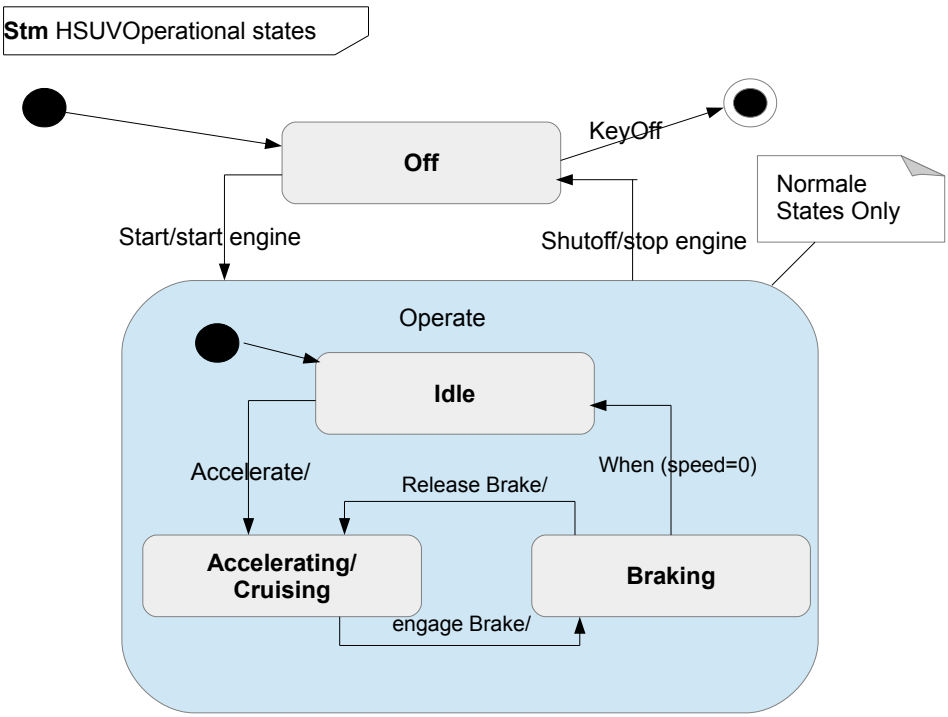


FIGURE I.10 – Diagramme d'état

4.3.3 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation permet de représenter les services offerts par le système. Chaque cas d'utilisation regroupe un ensemble de fonctionnalités. Dans ce type de diagramme (figure I.11), Les cas d'utilisation sont à l'intérieur d'un rectangle qui représente le système à réaliser. Les acteurs sont à l'extérieur du rectangle (ils utilisent le système à réaliser).
Les cas d'utilisation sont liés entre eux par des relations, les trois les plus communément utilisées sont :

— La relation «**include**» indique que le cas d'utilisation cible est inclus dans le cas d'utilisation source, il sera donc exécuté lors de l'exécution du cas source.

— La relation «**extend**» indique que le cas d'utilisation source étend les possibilités du cas d'utilisation cible, il sera exécuté par le cas d'utilisation source de manière optionnelle en fonction des conditions pré-définies dans le point d'extension.

— La relation de **généralisation/spécialisation** est une relation d'héritage. Le cas qui hérite réalise toutes les capacités du cas hérité, et y rajoute ses spécificités [4].

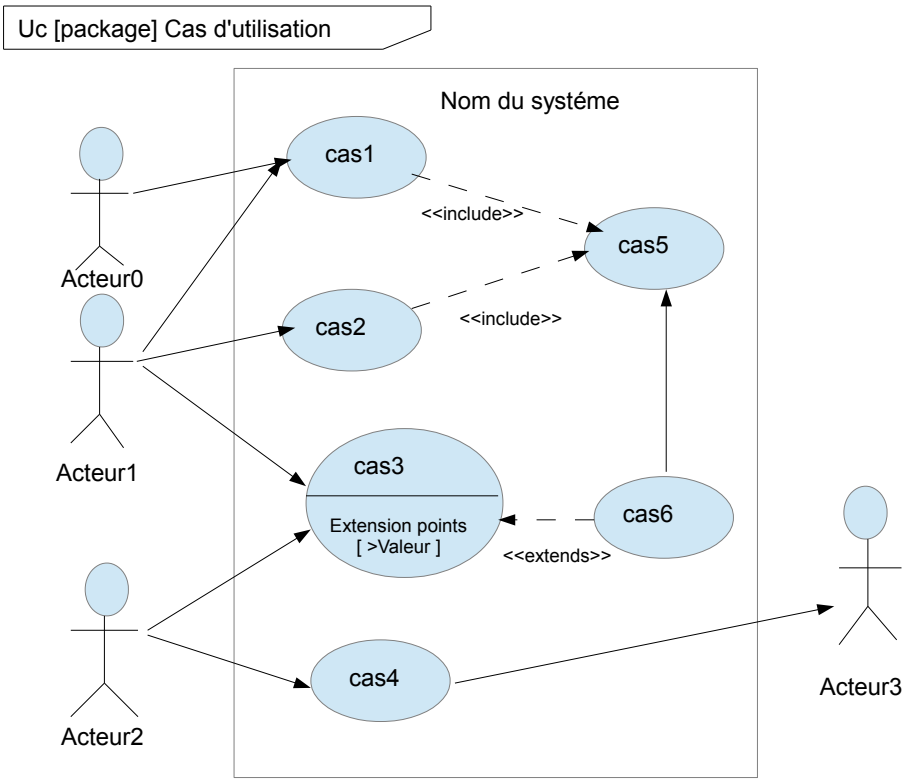


FIGURE I.11 – Diagramme des cas d'utilisation

4.3.4 Diagramme d'activité

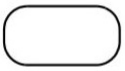
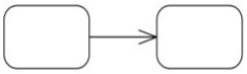
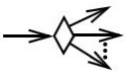
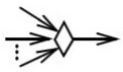
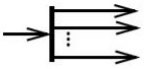
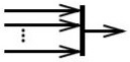
Le diagramme d'activités est un diagramme comportemental. Ce diagramme permet de représenter le déroulement d'un processus sous la forme d'une activité correspondant à une décomposition séquentielle d'actions, aussi appelées taches. Dans sa forme la plus restreinte, ce diagramme représente un algorigramme, c'est-à-dire un flux de contrôle (ce flux n'a rien à voir avec ceux présents dans le diagramme de blocs internes) [7]. Les éléments graphiques utilisés dans ce diagramme ressemblent fortement à ceux du diagramme d'états : chaque tâche est représentée par un rectangle aux coins arrondis et est ensuite reliée à une autre tâche par des transitions représentées par de simples flèches. Lorsqu'une tâche est terminée la suivante commence. en général les input et output pins qui correspondent respectivement au type d'élément requis en entrée d'une activité ou action, et à celui généré en sortie. Si une action ou activité correspond à l'opération d'un bloc, il est alors possible de vérifier que les types d'éléments définis en entrée et en sortie de cette activité soient cohérents avec la signature de l'opération du bloc ou de son interface. Toutes les propriétés des diagrammes d'activités UML sont également disponibles avec SysML ex : swimlanes etc. SysML a rajouté quelques spécificités :

— Notion de contrôle pour activer ou désactiver les actions en cours (Control Value)

— Spécification de la nature du débit sur le flot : système continu ou discret

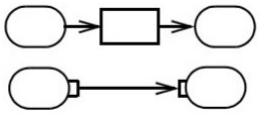
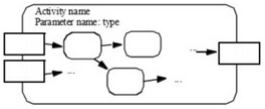
— Définition de taux et de probabilité sur les flux de contrôle ou d'objets [8].

TABLE 4.1: Les éléments du diagramme d'activité conforme aux spécifications **OMG**

Nom de nœud	**Syntaxe concret**	**Description**
Initial Node		Ceci montre un point de départ ou la première activité de l'écoulement. Dénote par un cercle plein.
Final Node		L'extrémité du diagramme d'activité est montré par un symbole de boudine, également appelé comme activité finale.
Action		La petite unité de fonction exécutable au sein d'une activité est l'action, elle est représentée sous forme d'un rectangle arrondis, ce dernier contient le nom ou brève description de l'action, si une action est appel d'activité, alors un petit symbole de fourchette est apposé en bas, adroite.
Control flow		Passage instantané et automatique à l'action suivante. Et déclenchées par la fin d'une activité et provoquant le début immédiat d'une autre.
Décision node		Une alternative spécifie des chemins alternatifs, chacun basé sur la valeur d'une expression booléenne (condition de garde). Qui valide ou non le déclenchement d'une transition, les conditions de garde ne doivent pas se chevaucher et couvrir tous les cas possibles.
Merge Node		Un noeud de contrôle qui rassemble de multiples flux alternatives entrant et accepte un flux sortant unique. Merge ne doit pas être utilisé pour synchroniser les flux simultanés.
Fork Node		La décomposition du flux de contrôle en deux ou plusieurs flux de contrôles.
Join Node		La recomposition de flux de contrôle de deux ou plusieurs flux de contrôle en un seule, et permet de spécifier la synchronisation d'activités concurrentes, Les flux qui sortent doivent balancer les flux qui entrent dans la jonction correspondant.

à suivre dans la page suivante

TABLE 4.1 – *Suivant*

Nom de noeud	Syntaxe concret	Description
ObjectFlow		Représente la participation d'un objet dans un flux de contrôle, et sont représentés par des relations de dépendance entre objets et états d'action ou d'activités.
Activity		Décrit l'exécution de fonctionnalités ou de comportements. elle est modélisée par plusieurs nœuds d'actions, les nœuds de contrôle et les nœuds d'objets.

- **Partition d'activité**

Dans les diagrammes d'activité, les partitions sont utilisées pour regrouper les nœuds et arcs d'activité ayant des caractéristiques communes. Elles fournissent une vue des comportements dans une activité, mais n'affectent pas le flux de contrôle ou le flux d'objet. Les partitions sont utiles dans la modélisation et peuvent indiquer les unités organisationnelles responsables des différentes activités dans un processus.
Chaque partition d'une activité comporte un nom unique qui spécifie les responsables des activités dans la partition. Comme le montre la figure I.12 suivante, une partition est représentée par un rectangle dans l'activité.

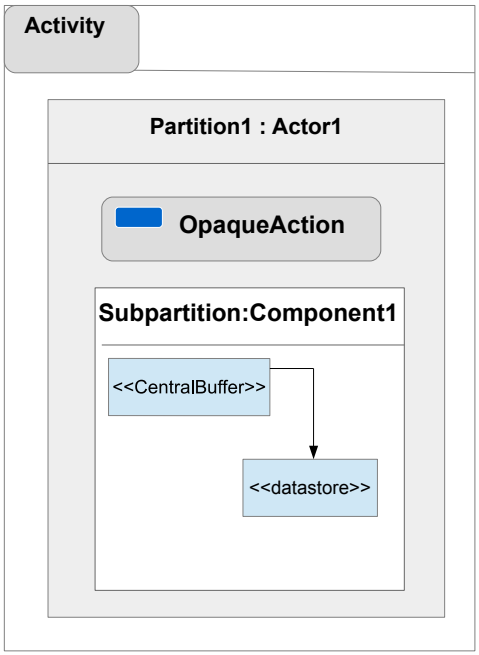


FIGURE I.12 – les Partitions d'activité

5 Conclusion

SysML est une notation pour la spécification et la conception des systèmes. C'est un profil UML qui réduit le nombre de diagrammes utilisés par rapport à UML tout en y ajoutant des concepts et des nouveaux diagrammes plus adaptés aux approches de modélisation système (par exemple les flux continus de type fluide, ondes...). Les diagrammes d'activités permettent de mettre l'accent sur les traitements. Ils sont donc particulièrement adaptés à la modélisation du cheminement de flots de contrôle et de flots de données. Ils permettent ainsi de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation. SysML est donc un langage suffisamment flexible pour permettre son utilisation dans divers cas de développement, mais il souffre principalement d'une faible sémantique formelle.

Chapitre II

Réseaux de Petri

1 Introduction

Ce chapitre permet d'initier à un outil de modélisation très puissant et très Répondu notamment dans le monde recherche et développement : RdPHC. Cet outil permet essentiellement la modélisation des systèmes quelque soit leurs domaines d'application (Informatique, Télécommunication, Production, ...). Nous présenterons dans un premier temps quelques définitions et formalismes concernant cet outil. Ensuite nous discuterons des propriétés très utiles dans l'analyse des RdPHC. A la fin du chapitre, nous présentons le CPN-tools que nous avons utilisé comme support de vérification pour les réseaux de Petri. Ensuite, nous abordons l'utilisation des techniques de vérification fondées sur les modèles "Modèle-Checking" ainsi le découpage de modèle pour la vérification.

2 Présentation de réseaux de Petri

Les Réseaux de Petri permettent de modéliser les systèmes. Ils ont été inventés par Carl Adam Petri, un mathématicien Allemand contemporain. Il a défini un outil mathématique très général permettant de décrire les relations existantes entre des conditions et des événements pour modéliser le comportement de systèmes dynamiques à évènements discrets. Les RdP datent de 1962. C'est un outil très général, modélisant aussi bien les protocoles de communication informatiques que des systèmes de production. Il est à l'origine du Grafcet (ce dernier étant spécialisé dans la description de la commande de systèmes automatisés). Il permet d'obtenir une représentation mathématique modélisant le système. L'analyse de cette représentation peut révéler des caractéristiques importantes du système concernant sa structure et son comportement dynamique [9]. Les résultats d'une telle analyse sont utilisés pour évaluer le système et en permettre la modification et/ou l'amélioration de cas échéant.

2.1 Définition informelle des réseaux de Petri

Un réseau de Petri est un graphe biparti orienté composé de deux types de noeuds :

— les places permettent de décrire les états du système modélisé.

— les transitions représentent les changements d'état.

Les places et transitions sont reliées par des arcs orientés (figure II.1). Une place peut contenir un nombre entier de jetons ou marques. L'ensemble des marques présentes à un instant donné dans les places constitue le marquage du réseau à cet instant [10].

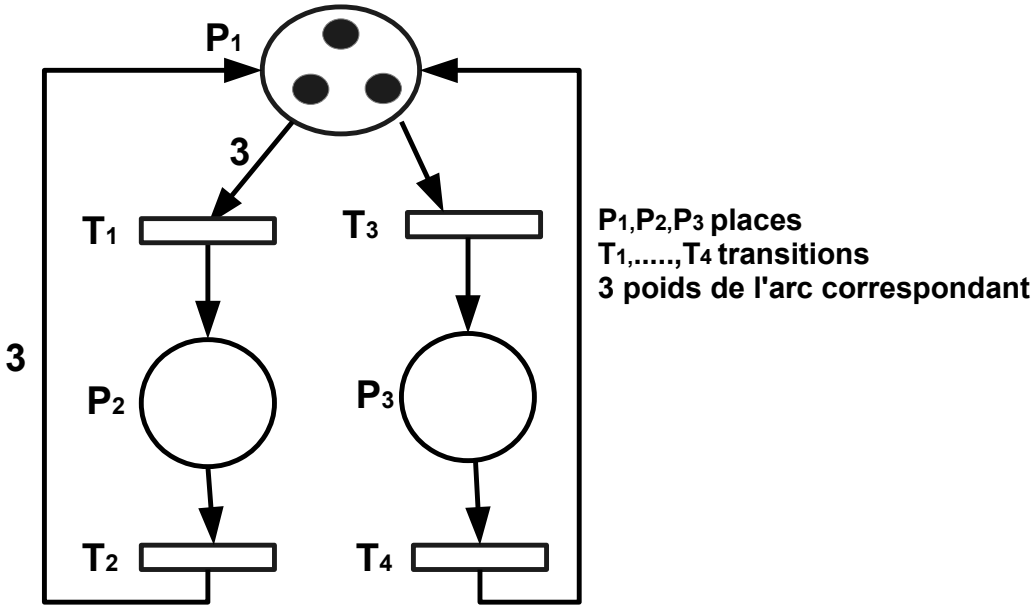


FIGURE II.1 – Exemple de réseau de Petri

2.2 Définitions formelles d'un réseau de Petri

Un réseau de Petri est un quadruplet : R = (P, T, Pre, Post)Où :

— $P = \{ P_1, P_2, ..., P_m \}$ un ensemble fini de places.

— $T = \{ T_1, T_2, ..., T_n \}$ un ensemble fini de transitions, disjoint de P.

— $Pre : P \times T \rightarrow \mathbb{N}$ l'application incidence avant.

— $Post : P \times T \rightarrow \mathbb{N}$ l'application incidence après .

L'application Pre donne les poids des arcs placés en amont des transitions. L'application Post donne les poids des arcs placés en aval des transitions [10].

2.3 Marquage Initial

Le marquage dit initial décrit l'état initial du système modélisé. À tout marquage accessible à partir du marquage initial par franchissement d'une séquence de transitions correspond un état du système. Il est à noter, la place n'est pas un état mais participe à

travers son marquage à la description d'un ou de plusieurs états. L'ensemble des marquages accessibles constitue le graphe d'états [10].

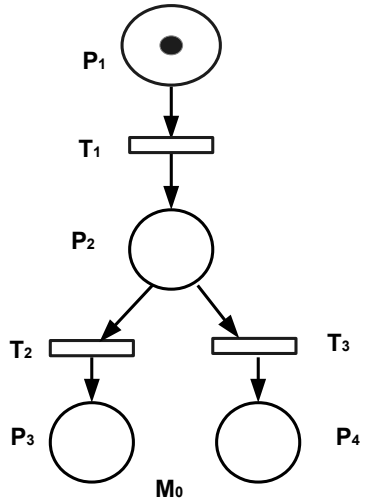


FIGURE II.2 – Exemple de réseau de Petri

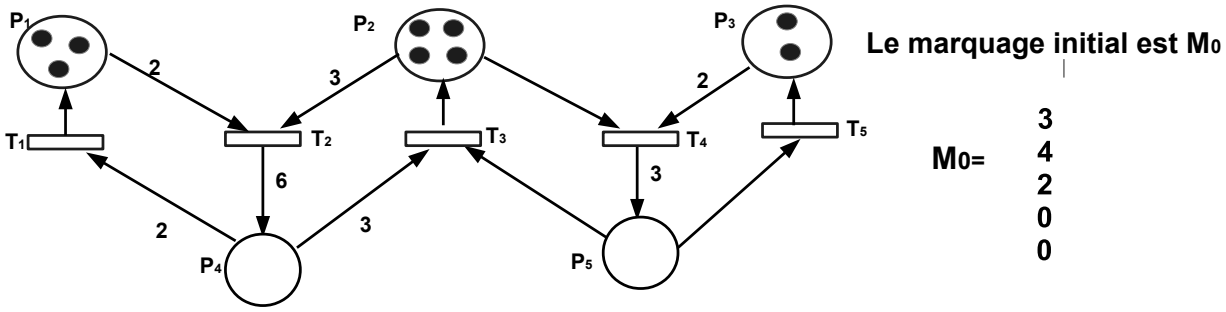


FIGURE II.3 – Exemple de marquage initial

Une transition se compose d'un ou de plusieurs arcs amont ou **arcs d'entrée** et d'un ou de plusieurs arcs aval ou **arcs de sortie**. Un réseau marqué est le couple :(R, M)Le marquage M d'un réseau de Petri R est une application de P dans ℕ. M(p) représente le nombre de jetons contenus dans P. Si $|P| = m$,$M(p)$ est un vecteur à m composantes. On note M_0 le marquage initial. À chaque arc est associé un nombre entier positif appelé poids de l'arc. Dans le cas fréquent, où les arcs sont tous de poids 1, on parle de réseaux de Petri ordinaires. Dans le cas contraire, il s'agit de réseaux de Petri généralisés. Une transition est dite sensibilisée si les places situées en amont(places d'entrée) possèdent chacune un nombre de jetons supérieur ou égal au poids des arcs joignant ces places à la transition(figure II.4)[10].

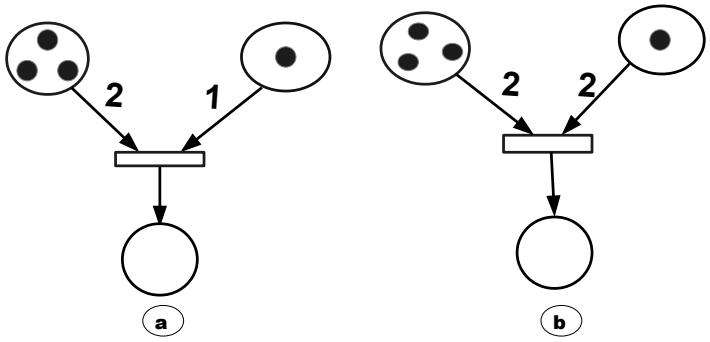


FIGURE II.4 – Transition sensibilisée(a)et transition non sensibilisée(b)

Notation matricielle

Les fonctions Pre et Post peuvent être représentées par des matrices à m lignes (nombre de places), n colonnes (nombre de transitions). Un élément de la matrice Pre, respectivement Post, s'écrira Pre(p, t), respectivement Post (p, t)[10].

2.4 Dynamique des RdP

2.4.1 Franchissement d'une transition

Franchir une transition consiste à prendre, dans chaque place d'entrée, un nombre de jetons égal au poids de l'arc joignant cette place à la transition et à déposer, dans chaque place aval ou place de sortie, un nombre de jetons égal au poids de l'arc joignant la transition à chacune de ces places (figureII.5). Le franchissement des transitions et les modifications de marquage qu'il entraîne permettent d'analyser la dynamique du système modéliser [10]. Une transition t est sensibilisée pour un marquage M si et seulement si $\forall$ $p \in P : P_i$ le franchissement de t conduit au nouveau marquage M' tel que :

$$\forall\ p \in P\ M'(p) = M(p) - Pre(p,t) + Post(p,t)\ .$$

La différence des marquages M et M' est égale à *Post-Pre*. On introduit la matrice W, dite matrice d'incidence telle que :

$$\forall\ p \in P, \forall\ t \in T : W(p,t) = Post(p,t) - Pre(p,t)\ .$$

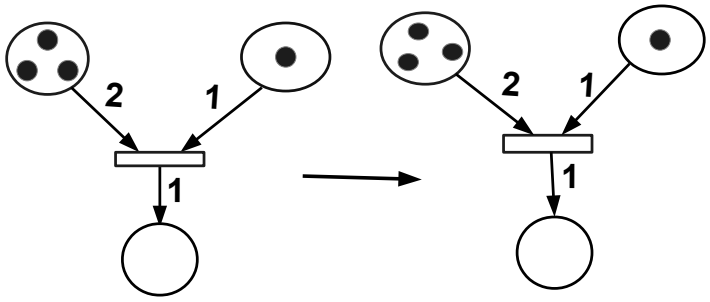


FIGURE II.5 – Franchissement d'une transition

2.4.2 Séquence de franchissements

Une séquence de franchissements à partir d'un marquage M_1 est représentée par la suite des transitions T_i,T_j,... telles que le franchissement de chacune d'elles conduit à un marquage qui sensibilise la suivante [10] :

$$\text{Si } M_1 \xrightarrow{T_i} M_2 \xrightarrow{Tj} .....\xrightarrow{Tr} M_q$$
$$M_1 \xrightarrow{S} M_q$$

On dira que la séquence $S = T_i$, T_j,..., T_r est applicable à partir de M_1.

2.4.3 Marquages accessibles

L'ensemble des marquages accessibles est l'ensemble des marquages M_i qui peuvent être atteint par le franchissement d'une séquence S à partir du marquage initial M_0. On le note ${}^*M_0$.

$$\{\ {}^*M_0 = M_i \text{ telque } M_i\ [S \xrightarrow{S} M_j\ \}$$

la figure II.2 présente un ensemble des marquages accessibles

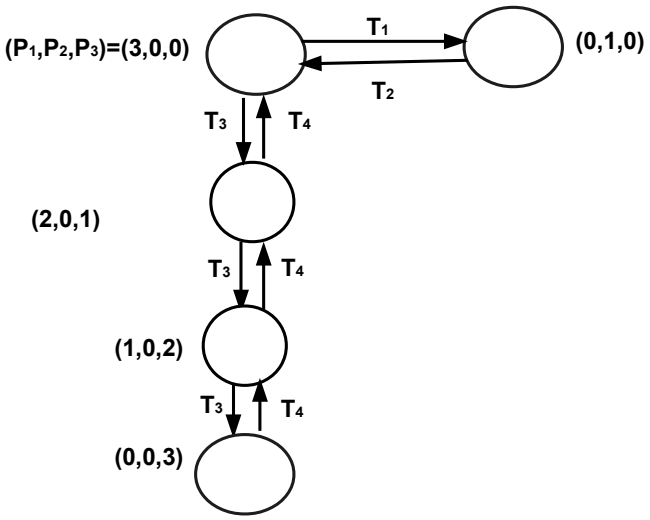


FIGURE II.6 – Ensemble des marquages accessibles

$^*M_0$=M_0,M_1,M_2,M_3Avec M_0=[1 0 0 0], M_1=[1 0 0 0], M_2=[0 1 0 0]et M_3=[0 0 0 1]

Notation : On notera par $^*M$ l'ensemble des marquages accessibles à partir du marquage M. Par s la séquence de transitions t_1,t_2,...,t_n et le passage du marquage M au marquage M' par franchissement de la séquence de transitions s par $M \xrightarrow{s} M'$.

2.4.4 Graphe de marquages

On peut représenter l'ensemble de marquages accessibles par un graphe si ce dernier est fini. Le graphe de marquages a comme sommet l'ensemble de marquages accessibles A(R,M_0) et un arc orienté relie deux sommet M_i et M_j s'il existe une transition t franchissable permettant de passer d'un marquage de M_i à M_j Mi [t - Mj]. Les arcs du graphe sont étiqueté par les transitions correspondantes. la figure II.7 représente le graphe de marquages accessibles correspond au RdP.

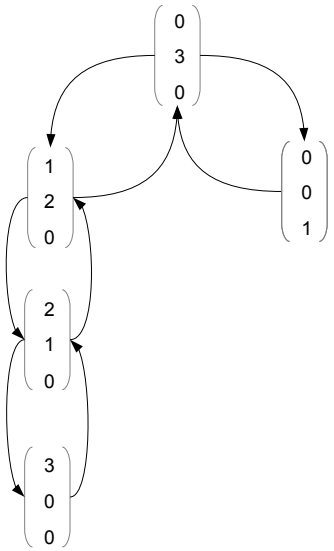


FIGURE II.7 – Graphe de Marquage

2.5 Propriétés des réseaux de Petri

De nombreuses propriétés des RdP ont été mises en évidence. On peut les regrouper en deux catégories, selon qu'elles concernent la structure du réseau (on parle alors de propriétés structurelles) ou bien son comportement (propriétés comportementales). Les propriétés structurelles ne dépendent que de la structure du RdP et non pas de la manière dont les jetons évoluent dans le réseau. On n'en présentera que quatre, à savoir : vivacité et blocage, bornitude et état d'accueil [11]. L'analyse d'une fonction ou d'un système séquentiels passe par l'étude des propriétés du réseau de Petri qui les représente. Parmi ces propriétés, on citera celles qui permettent d'affirmer que les spécifications incluses dans le modèle réseau de Petri sont correctes [12]. C'est ainsi que on pourra démontrer qu'un réseau (donc le système représenté) est sans blocage ou que le nombre d'états pouvant être atteints est fin. On mettra également en évidence les conflits entre plusieurs évolutions possibles ou encore la gestion du partage des ressources.

2.5.1 Réseau de Petri borné

Un réseau de Petri est borné pour un marquage initial donné si, quel que soit le marquage accessible atteint M et quelle que soit la place p considérée, le nombre de jetons contenus dans cette place est inférieur à une borne k :

$$\forall\ M \in P : M(p) \leq k$$

On dira également que le nombre d'états accessibles à partir de l'état initial est fini, le graphe d'états équivalent peut donc être construit [10].

2.5.2 Vivacité

• **Transition vivante :** Une transition T_j est vivante pour un marquage initial M_0 si pour tout marquage accessible M_i (appartenant à ${}^*M_0$), il existe une séquence de franchissement S qui contient la transition T_j, à partir de M_i. Autrement dit, quelque soit l'évolution, il subsistera toujours une possibilité de franchir T_j à nouveau [13].
• **Réseau de Petri vivant :** Un réseau de Petri est vivant pour un marquage initial donné si, pour tout marquage M accessible à partir du marquage initial et pour toute transition t, il existe une séquence s de franchissements qui inclut la transition t [10] :

$$\forall\ M, \forall\ P \in P\ \exists\ S : M \xrightarrow{S} M' \text{ telque } t \subset S$$

• **Quasi-vivacité :** Une transition T_j est quasi vivante pour un marquage initial M_0, s'il existe une séquence de franchissement qui contient T_j à partir de M_0. Un RdP est quasi vivant si toutes ses transition sont quasi vivantes [13].

2.5.3 Blocage

On parle de blocage lorsque à partir d'un marquage aucune transition n'est validée. Un RdP est dit donc sans blocage (pseudo vivant) pour un marquage initial M_0 si aucun marquage accessible M_i n'est un blocage [13].

2.5.4 États d'accueil

Un RdP possède un état d'accueil M_a pour un marquage initial M_0 si pour tout marquage accessible M_i il existe une séquence de franchissement permettant d'atteindre le marquage M_a, M_a est un état d'accueil [14] :

$$\forall\ M_i \in {}^*M_0, \exists\ S \in T^* : M_i \xrightarrow{S} M_a$$

Un RdP est réinitialisable (ou réversible) pour un marquage initial M_0 si M_0 est un état d'accueil.

2.6 Les réseaux de Petri de haut niveau

2.6.1 Réseaux de Petri Colorés

Les RdP colorés sont [15] dérivés des RdP ordinaires. Ils sont des RdP dans lesquels les jetons portent des couleurs. Chaque couleur correspond à une information bien définie associée au jeton. Ce type de réseaux s'utilise pour éviter le problème de taille importante des RdP ordinaires dans le cas ou des entités différentes présentent des comportements

similaires ce qui condense le modèle. L'intérêt des réseaux colorés est de pouvoir décrire des dynamiques relativement complexes avec un petit nombre de places et de transitions.

2.6.1.1 Définition

La couleur est l'identificateur [16] :

- Aux marques de chaque place ou réseau, pour distinguer deux jetons se trouvant dans la même place.
- Aux transitions, pour indiquer la ou les couleurs de franchissement de celles si. C'est à dire préciser la(es)couleur(s) des jetons affectés par le franchissement de la transition.

2.6.1.2 Franchissement d'une transition dans un RdP coloré

Chaque transition peut être franchie de plusieurs manières, selon les couleurs de franchissement qui lui sont associées. Le franchissement d'une transition dans les réseaux de Petri généralisés est réalisé par :

- Le retrait systématique d'un jeton de chacune des places d'entrée de la transition .
- L'ajout systématique d'un jeton dans chacune des places de sortie de la transition.

Dans un RdP coloré, on a besoin d'un outil supplémentaire qui permet de définir la couleur des jetons retirés des places d'entrée, et celle des jetons à ajouter aux places de sortie. Deux types de relations sont utilisés :

- La première entre la couleur du jeton contenu dans la place d'entrée de la transition, et la couleur de franchissement associée à cette dernière.
- La deuxième entre la couleur de franchissement de la transition, et la couleur de jeton que doit contenir la place de sortie de celle-ci

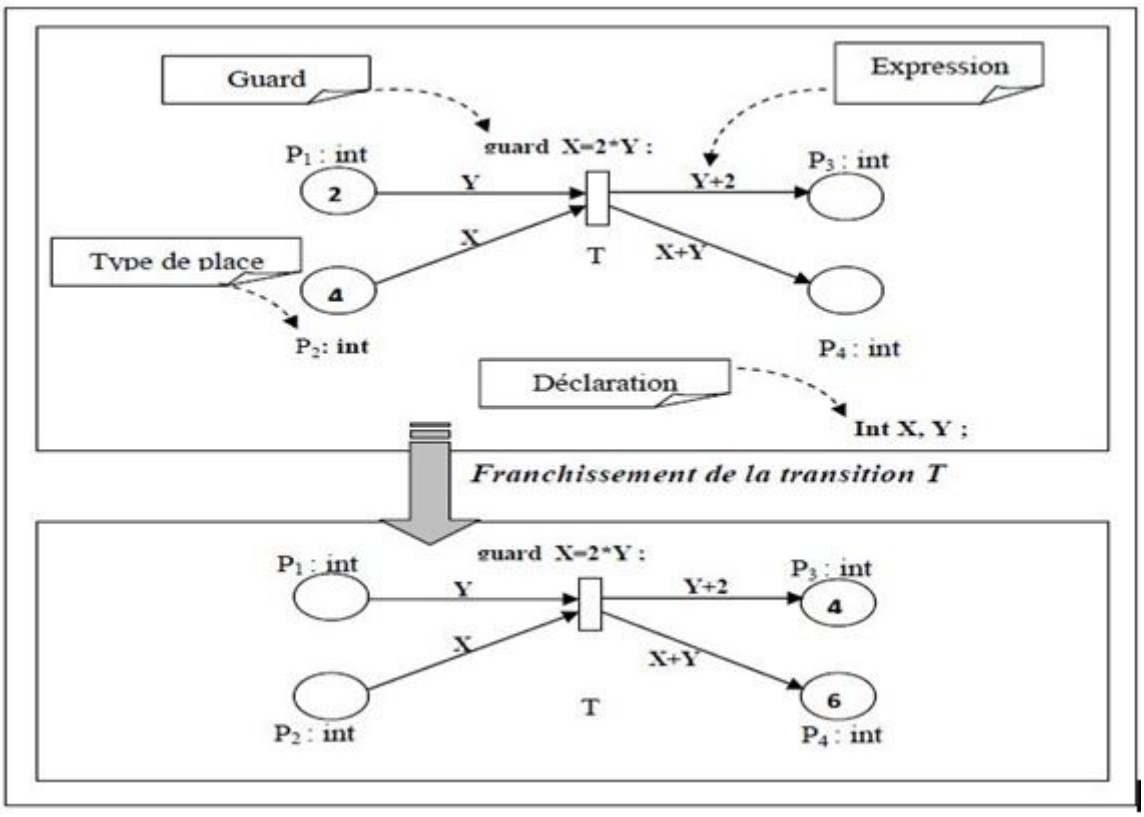


FIGURE II.8 – Réseau de Petri coloré

Dans un RdP colorés, les arcs connectant les places aux transitions sont étiquetés par des constantes, des variables ou des fonctions. Les valeurs affectées aux variables, les valeurs obtenues après évaluation des fonctions et les constantes définissent le nombre et le type des jetons nécessaires pour tirer une transition, et que seront retirés de certaines places ou ajoutés à certaines d'autres. Dans un RdP colorés, on peut associés des expressions booléennes (des gardes) à une transitions. Cette dernière ne sera franchissable que si son expression de garde est vraie [14].

2.6.2 Réseaux de Petri hiérarchiques

Le principe de RdP hiérarchiques [17] , est de construire un modèle en combinant un certain nombre des réseaux appelés "page" en un seul réseau. L'objectif est de réduire la complexité d'un modèle en permettant la modélisation modulaire. Un RdP hiérarchique se compose de plusieurs modules "pages", où chaque module représente un sous réseau. Un module dit principal référence tous les autres modules secondaires. Le référencement d'un module se fait au moyen d'une transition ou une place de substitution portant le nom du module en question. L'utilisation des RDP hiérarchiques est préconisée pour la modélisation des problèmes de grande taille. Avec la notion d'hiérarchie obtenue par la substitution des places ou des transitions il est possible de s'intéresser à un niveau en faisant abstraction des niveaux qui lui sont inférieurs (figure II.9).

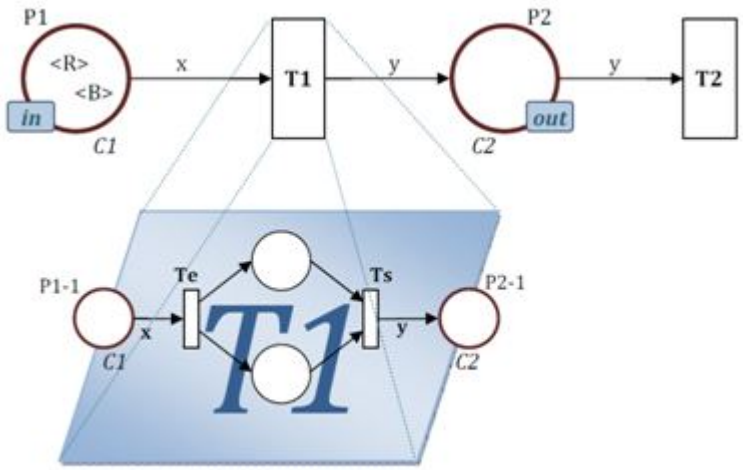


FIGURE II.9 – Réseau de Petri coloré Hiérarchique

2.7 Outils de modélisation des réseaux de Petri

L'aspect formel des RdP a encouragé les développeurs à mettre au point une multitude d'outils de simulation et de vérification des RdP selon la technique de vérification de modèle (CPNTools, CPNAMI, Petri Net Kernel, ... etc). La plupart de ces outils présente un environnement graphique d'édition des RdP avec la possibilité de simuler le modèle et d'analyser des propriétés génériques des RdP [14] tel que CPNTools.

2.7.1 CPN Tools

CPNtools [18] est la nouvelle version de Design/CPN développé à l'université d'Aarhus au Danemark. CPNtools est utilisé par une large communauté d'universitaires et d'industriels ce qui a permis son développement. CPNtools permet de spécifier les systèmes temps réel dans le formalisme des réseaux de Pétri colorés [18]. La spécification peut se faire sous forme de réseaux hiérarchiques, ce qui en facilite la description et la compréhension. Le langage utilisé pour la spécification est propre à l'outil mais le vérificateur syntaxique facilite grandement cette tâche. Le système peut aussi être spécifié de manière visuelle grâce à l'éditeur graphique. Les propriétés peuvent être spécifiées dans la logique CTL. L'interface utilisateur permet une fois encore de spécifier les propriétés de manière conviviale. L'outil offre deux modes de simulations : la simulation pas-à-pas et la simulation automatique. Dans chacun des modes, l'interface graphique assiste l'utilisateur dans la compréhension du comportement spécifié par un système d'activation par coloriages des composants. Des critères d'arrêt peuvent être spécifiés pour contrôler le mode automatique. La vérification automatique génère une trace au format textuel, mais un mécanisme d'extension de l'outil permet d'associer des fonctions utilisateurs à des changements d'états du système dans une trace donnée. Ces extensions doivent être écrites dans le langage SML [19].

2.7.1.1 Couleurs supportées

La déclaration des couleurs et des variables est effectuée à l'aide du langage CPNML basé sur le langage de programmation ML [20]. Grâce au logiciel CPNtools, il est possible d'utiliser des couleurs complexes et des fonctions. Les couleurs simples existantes dans le CPNML sont : unité (couleur noir sans couleur), entier, booléen, string (chaîne de caractères) et énuméré.

3 Vérification

La vérification répond à la question « Construisons-nous correctement le modèle ? La vérification est l'ensemble des actions de revue, inspection, test, preuve automatique ou autres techniques appropriées permettant d'établir et de documenter la conformité des artéfacts du développement vis-à-vis des critères préétablis. Dans la norme(ISO 8402) la vérification est définit comme étant "la confirmation par examen et apport de preuves tangibles (informations dont la véracité peut être démontrée, fondée sur des faits obtenus par observation, mesures, essais ou autres moyens) que les exigences spécifiées ont été satisfaites" [21].

3.1 Techniques de vérification formelle

Classiquement, on distingue deux grandes familles de techniques pour vérifier formellement la correction d'un système. En premier lieu, les techniques de preuve (theorem proving) qui sont des démonstrations mathématiques au sens classique du terme où la vérification des propriétés est effectuée par déduction à partir d'un ensemble d'axiomes et de règles d'inférences. La seconde famille de techniques est appelée vérification de modèles, ou "model-checking", et consiste à construire un modèle à partir d'une description formelle d'un système.

3.1.1 Vérification de modèles (Model-Checking)

Le model checking [22] désigne une famille de techniques de vérification automatique de systèmes dynamiques. Il s'agit de vérifier si un modèle donné, le système lui-même ou une abstraction, satisfait une spécification, telles que l'absence d'interblocage (deadlock). Pour cela, il faut analyser tous les cas d'exécutions possibles du système. A l'issue de cette vérification, si une propriété n'est pas satisfaite, un contre-exemple est fourni permettant ainsi d'identifier un cas d'exécution non conforme et ainsi localiser et corriger la source de l'erreur. Ce contre-exemple peut être obtenu dès l'instant où la propriété à vérifier n'est pas satisfaite durant l'exécution, ce qui permet une vérification plus rapide qu'avec les techniques traditionnelles, au-delà du fait que celle-ci est automatique (figure II.10)[22].

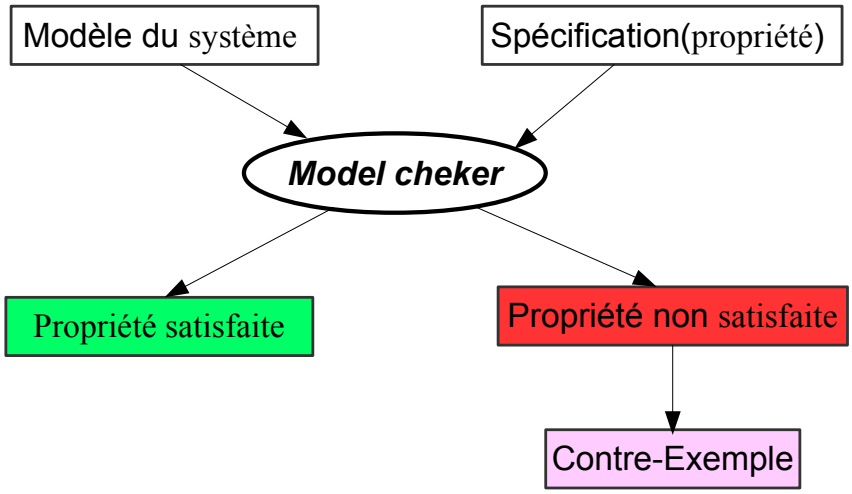


FIGURE II.10 – Principe du model checking

3.1.2 L'explosion combinatoire de l'espace d'états

Lors de la vérification, l'espace d'états calculé a une taille exponentielle par rapport à la taille du modèle, son calcul peut alors être long et consommer beaucoup de mémoire. La vérification basée sur les modèles donc souffre de l'explosion combinatoire liée à la taille du système qui est exponentielle en nombre d'états du système. A plusieurs approches peuvent être suivies pour pallier à ce problème, parmi ces approche la technique de découpage de modèle [23].

4 Découpage de modèles pour la vérification

Le découpage de modèle est une technique, traditionnellement appliquée aux programmes, pour extraire des parties. Au cours des dernières années, les auteurs ont commencé à envisager le découpage au niveau du modèle.
le Programme de découpage est une analyse du code source et manipulation technique, dans lequel un sous- programme est identifié en se basant sur un critère de découpage spécifié par l'utilisateur.
La nécessité pour le découpage de modèle est forte : les modèles véhiculent de nombreux types d'informations mieux que les programmes, mais deviennent difficiles à l'échelle beaucoup plus rapide .
Le découpage de modèle est mis en oeuvre dans le cadre d'un effort de recherche plus large, où on trouve certaines méthodologie tel que la methodologie qui est décrit dans cet

article[24]. L'objectif du découpage de modèle dans la vérification basé sur le modèle vis à pallier au problème d'explosion combinatoire de l'espace d'état en considérant que les parties concernées lors de la vérification d'une propriété.

5 Conclusion

Dans ce chapitre, nous avons présenté le formalisme des réseaux de Petri puis les approches de vérification formelle, en focalisant sur l'approche de model-checking et son application sur les RdP. Nous avons vu que la limite de l'approche model-checking pour la vérification souffre du problème de l'explosion combinatoire de l'espace d'état. Pour pallier à cette limite plusieurs techniques ont été proposées. En particulier, la technique de découpage de modèle vise à réduire l'effet de ce problème en considérant que des parties concernées lors de la vérification d'une propriété. Le présent travail s'appuie sur cette technique pour vérifier des diagrammes d'activité SysML.

Chapitre III

Ingénierie Dirigée par les Modèles

1 Introduction

L'ingénierie dirigée par les modèles (IDM), ou Model Driven Engineering (MDE) est une discipline récente du génie logiciel qui met l'accent sur les modèles au sein du processus de développement logiciel. L'IDM est donc le domaine de l'informatique mettant à disposition des outils, concepts et langages pour créer et transformer des modèles. La transformation des modèles est une tâche complexe qui nécessite la disposition d'outils flexibles permettant la gestion des modèles et des langages durant la transformation et la manipulation des modèles. La sémantique de ces derniers doit être spécifiée, et les langages utilisés doivent être décrit de manière précise. Dans ce chapitre, nous allons présenter les notions et les concepts relatifs à l'ingénierie dirigée par les modelés.

2 Présentation de l'IDM

L'Ingénierie Dirigée par les Modèles (IDM) ou Model-driven engineering (MDE) fait partie des domaines de l'informatique et de l'ingénierie système, elle fournit les outils et concepts nécessaires pour créer ou modifier des langages de modélisation. Ce domaine relativement récent à fait son apparition dans les années 80 avec un premier outil qui supportait les concepts de l'IDM, "the Computer-Aided Software Engineering" (CASE). La complexité grandissante des systèmes à modéliser à conduit les développeurs à manipuler des concepts de plus haut niveau. L'Object Management Group a fini par s'intéresser à l'Ingénierie Dirigée par les Modèles et a proposé en 1997 le standard MOF (Meta-Object Facility). On parle également parfois de MDA (Model Driven Architecture) qui est une marque déposée de OMG qui reprend et restreint les concepts de l'IDM [25].

3 Concept fondamentaux de l'IDM

3.1 Modélisation

L'approche objet est fondée sur les relations essentielles d'instanciation et d'héritage, l'IDM est basée sur un autre jeu de concepts et de relations. Le concept central de

l'IDM est la notion de modèle, pour laquelle il n'existe pas à ce jour de définition universelle. Néanmoins, de nombreux travaux s'accordent à un relatif consensus d'une certaine compréhension. On considère la définition suivante d'un modèle.

Définition (Modèle) : Un modèle est une abstraction d'un système, modélisé sous la forme d'un ensemble de faits construits dans une intention particulière. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé [26]. Les modèles offrent de nombreux avantages. Ceux qui pratiquent UML ou tout autre langage de modélisation les connaissent bien. L'avantage le plus important qu'ils procurent est de spécifier différents niveaux d'abstraction, facilitant la gestion de la complexité inhérente aux applications. Les modèles très abstraits sont utilisés pour présenter l'architecture générale d'une application ou sa place dans une organisation, tandis que les modèles très concrets permettent de spécifier précisément des protocoles de communication réseau ou des algorithmes de synchronisation. Même si les modèles se situent à des niveaux d'abstraction différents, il est possible d'exprimer des relations de raffinement entre eux.

La notion de modèle dans l'IDM fait explicitement référence à la notion de langage bien défini. En effet, pour qu'un modèle soit productif, il doit pouvoir être manipulé par une machine. Le langage dans lequel ce modèle est exprimé doit donc être clairement défini. De manière naturelle, la définition d'un langage de modélisation a pris la forme d'un modèle, appelé métamodèle[27].

3.2 Méta-modélisation

Un méta-modèle est une définition d'un point de vue donne un langage de modélisation, c'est le modèle du modèle. A première vue un modèle ne ressemble à aucun autre modèle. Suivant le système à modéliser, suivant la vision qu'a le concepteur du système, suivant sa maitrise du langage de modélisation, le modèle résultant sera différent. Pourtant tous ces modèles sont regroupes sous une même syntaxe et une même sémantique. Il y a un langage, un formalisme qui peut lui-même être modélisé. C'est le rôle premier de la méta-modélisation. Comme il est dit dans la première partie de ce chapitre l'OMG a définie MDA, qui préconise une vision sur 4 niveaux de modélisation au travers du MOF [28] La figure III.1 montre une représentation graphique du MOF et ses quatre niveaux d'abstraction.

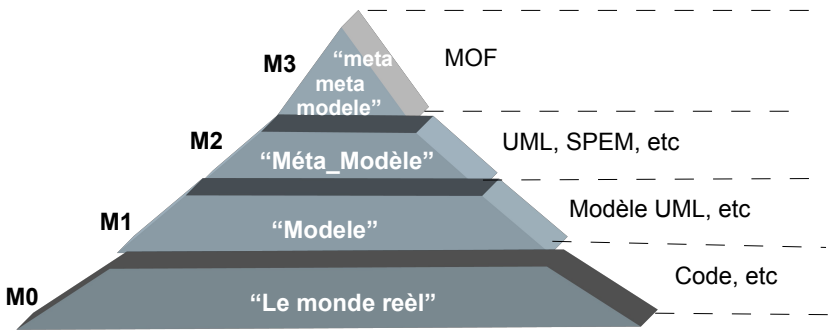


FIGURE III.1 – Modele du MOF.

Définition (Métamétamodèle) : Un méta-méta modèle est un modèle qui décrit un langage de méta-modélisation, c-à-d les éléments de modélisation nécessaires à la définition des langages de modélisation. Il a de plus la capacité de se décrire lui-même. C'est sur ces principes que se base l'organisation de la modélisation de l'OMG généralement décrite sous une forme pyramidale (figure III.1. Le monde réel est représenté à la base de la pyramide (niveau M_0). Les modèles représentant cette réalité constituent le niveau M_1. Les méta-modèles permettant la définition de ces modèles (ex. UML) constituent le niveau M_2. Enfin, le métamétamodèle unique et méta-circulaire, est représenté au sommet de la pyramide (niveauM_3). L'approche consistant à considérer une hiérarchie de méta-modèles n'est pas propre à l'OMG, ni même à l'IDM, puisqu'elle est utilisée depuis longtemps dans de nombreux domaines de l'informatique. Chaque hiérarchie définit un espace technique [29] [30] [31] (III.2).

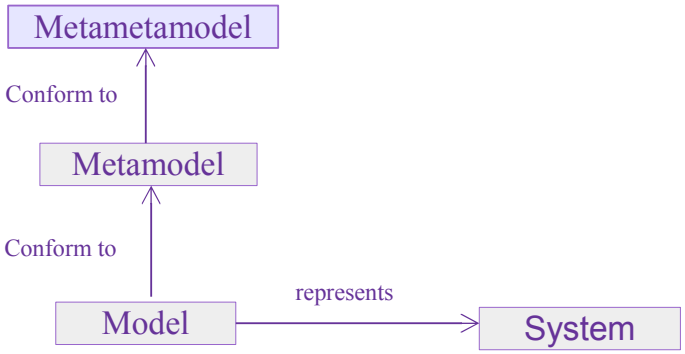


FIGURE III.2 – Les relations de bases dans l'IDM.

3.3 Approche MDA Model Driven Architecture (MDA)

L'OMG a défini MDA (Model Driven Architecture) en 2000 dans cet objectif. L'approche MDA préconise l'utilisation massive des modèles et offre de premières réponses aux comment, quand, quoi et pourquoi modéliser. Sans prétendre être une Bible de la modélisation, répertoriant toutes les bonnes pratiques, elle vise à mettre en valeur les qualités Intrinsèques des modèles, telles que pérennité, productivité et prise en compte des plateformes d'exécution. MDA inclut la définition de plusieurs standards, notamment UML, MOF et XMI. Le principe clé de MDA consiste en l'utilisation de modèles aux différentes phases du cycle de développement d'une application. Plus précisément, MDA préconise l'élaboration de modèles d'exigences (CIM), d'analyse et de conception (PIM) et de code (PSM). L'objectif majeur de MDA est l'élaboration de modèles pérennes, indépendants des détails techniques des plates-formes d'exécution (J2EE, .Net, PHP ou autres), afin de permettre la génération automatique de la totalité du code des applications et d'obtenir un gain significatif de productivité. D'autres modèles, comme les modèles de supervision, de vérification ou d'organisation d'entreprise, ne sont pas encore intégrés dans l'approche MDA mais le seront rapidement sans difficulté, compte tenu de son ouverture. Les quatre principaux types de modèles définis dans l'approche MDA sont les suivants [32] (figureIII.3)

— **CIM (Computation Independant Model) :** Appelé aussi modèle de domaine ou modèle métier, le CIM capture les exigences en termes de besoins et décrit la situation dans laquelle le système sera utilisé. Son but est d'aider à la compréhension du problème mais aussi de fixer un vocabulaire commun pour un domaine particulier. Dans la pratique, l'appellation "CIM" est très peu utilisée.

— **PIM(Platform Independant Model) :** Le PIM décrit le système indépendamment de la plate-forme cible sur laquelle il s'exécutera. Il présente donc une vue fonctionnelle détaillée du système, sans détails techniques. Il peut être raffiné progressivement jusqu'au intégrer des détails d'architecture spécifiques à un type de plate-forme (machine virtuelle, système d'exploitation, etc.) mais il doit rester technologiquement neutre.

— **PDM (Platform Description Model) :** Le PDM est le modèle qui décrit une plateforme d'exécution. Il fournit un ensemble de concepts techniques représentant les différentes parties de la plate-forme et/ou les services qu'elle fournit.

— **PSM (Platform Specific Model) :** Le PSM est le résultat de la combinaison du PIM et du PDM. Il représente une vue technique détaillée du système.

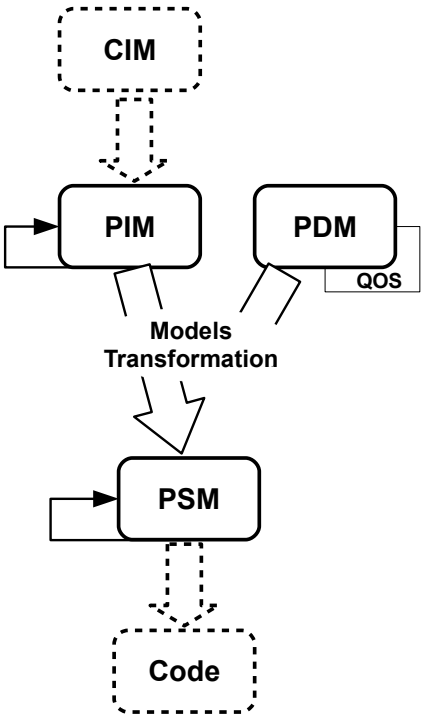


FIGURE III.3 – Le processus en Y de l'approche MDA.

Transformation des modèles du MDA

Comme expliqué au début, MDA représente à l'aide de modèles toutes les informations permettant la construction d'applications informatiques. La mise en œuvre de MDA induit donc nécessairement la création d'un nombre important de modèles. Chacun de ces modèles contient une partie de l'information utile pour la génération des applications informatiques. La mise en production de MDA passe par la mise en relation de ces modèles et plus précisément par leurs transformations respectives. C'est grâce aux transformations de modèles qu'il est possible, par exemple, d'obtenir une traçabilité entre des modèles très abstraits, proches des besoins exprimés par les utilisateurs, et des modèles très concrets, proches des plates-formes d'exécution [27].

— **Transformations de modèles CIM vers PIM :** Permettent de construire des PIM partiels à partir des informations contenues dans les CIM. L'objectif est de s'assurer que les besoins exprimés dans les CIM sont retranscrits dans les PIM. Ces transformations sont essentielles à la pérennité des modèles. Ce sont qui garantissent les liens de traçabilité entre les modèles et les besoins exprimés par le client. La figure III.4 reprend les phases de MDA et illustre les principales transformations de modèles MDA.

— **Transformations de modèles PIM vers PIM :** Permettent de raffiner les PIM afin d'améliorer la précision des informations qu'ils contiennent. En UML, de telles transformations peuvent être par exemple, la création de classes UML à partir d'un ensemble de diagrammes de séquences ou la création de diagrammes d'états à partir de diagrammes de classes. Ces transformations sont omniprésentes dans les outils d'élaboration de modèles PIM. Elles permettent d'accélérer la production des PIM et donc de raccourcir le cycle de développement.

— **Transformations de modèles PIM vers PSM :** Permettent de construire une bonne partie des modèles PSM à partir des modèles PIM. Ces transformations sont les plus importantes de MDA car elles garantissent la pérennité des modèles aussi bien que leur productivité et leur lien avec les plates-formes d'exécution.

— **Transformations de modèles PSM vers du code :**Permettent de générer la totalité du code.

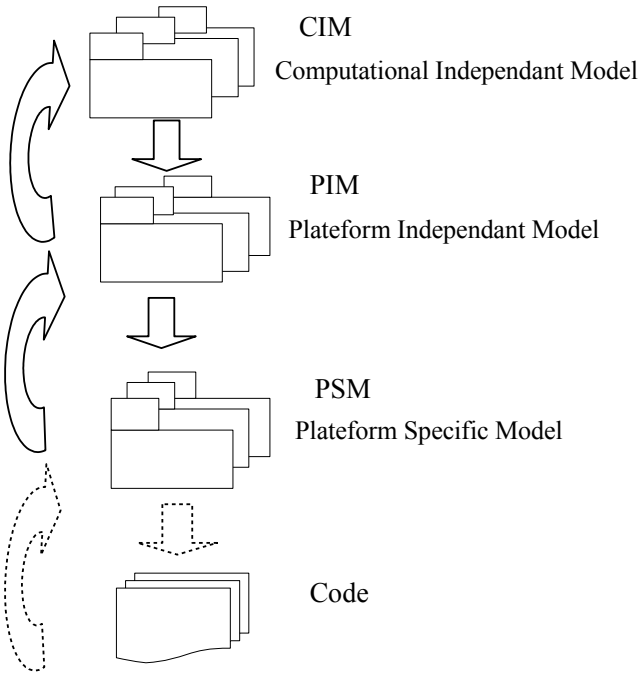


FIGURE III.4 – Transformations principales de MDA

3.4 Transformation de modèles

La définition la plus générale et qui fait l'unanimité au sein de la communauté IDM [30] consiste à dire qu'une transformation de modèles est la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources. Dans l'approche par

modélisation, cette transformation se fait par l'intermédiaire de règles de transformations qui décrivent la correspondance entre les entités du modèle source et celles du modèle cible. En réalité, la transformation se situe entre les métamodèles source et cible qui décrivent la structure des modèles cible et source. Le moteur de transformation de modèles prend en entrée un ou plusieurs modèles sources et crée en sortie un ou plusieurs modèles cibles, une transformation des entités du modèle source met en jeu deux étapes. La première étape permet d'identifier les correspondances entre les concepts des modèles source et cible au niveau de leurs métamodèles, ce qui induit l'existence d'une fonction de transformation applicable à toutes les instances du métamodèle source. La seconde étape consiste à appliquer la transformation du modèle source afin de générer automatiquement le modèle cible par un programme appelé moteur de transformation ou d'exécution. La figure III.5 illustre ces deux étapes d'une transformation de modèles [33] .

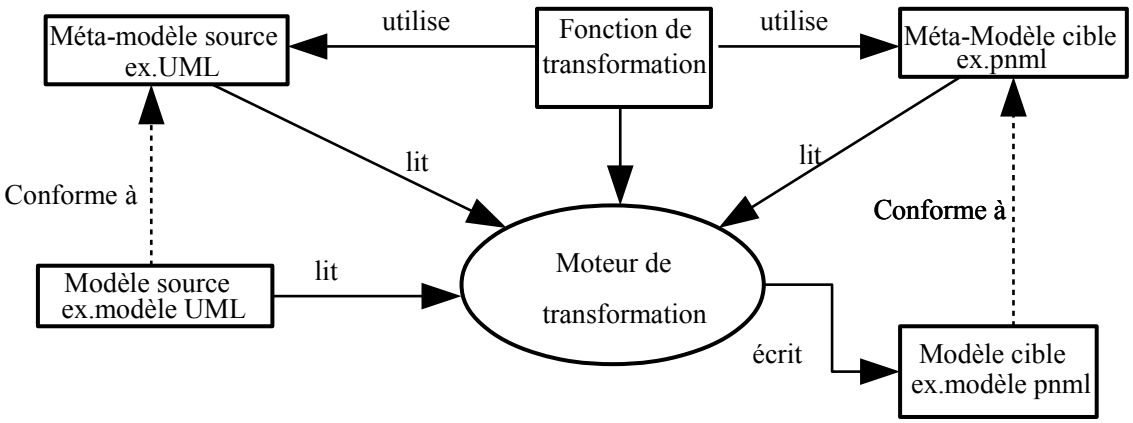


FIGURE III.5 – Schéma de base d'une transformation de modèles

Principales approches de transformation de modèles

Dans [34] trois approches de transformations de modèles sont retenues : l'approche par programmation, l'approche par template et l'approche par modélisation, on détaille l'approche par modélisation [33].

Approche par modélisation

Consiste à appliquer les concepts de l'ingénierie des modèles aux transformations des modèles elles-mêmes. L'objectif est de modéliser les transformations de modèles et de rendre les modèles de transformation pérennes et productifs, en exprimant leur indépendance vis-à-vis des plates-formes d'exécution. Le standard MOF 2.0 QVT de l'OMG a été élaboré

dans ce cadre et à pour but de définir un métamodèle permettant l'élaboration des modèles de transformation de modèles. A titre d'exemple, cette approche a été choisie par le groupe ATLAS à travers son langage de transformation de modèles ATL [33].

Type de Transformation

Il existe deux types de transformation figure III.6

La transformation Endogène : Le modèle source et le modèle cible sont conformes au même méta-modèle (e.g. : d'un diagramme SysML vers un diagramme SysML).

La transformation Exogène : Le modèle source et le modèle cible sont conformes à des méta-modèle différents (e.g. : d'un SysML vers un RdP)[26].

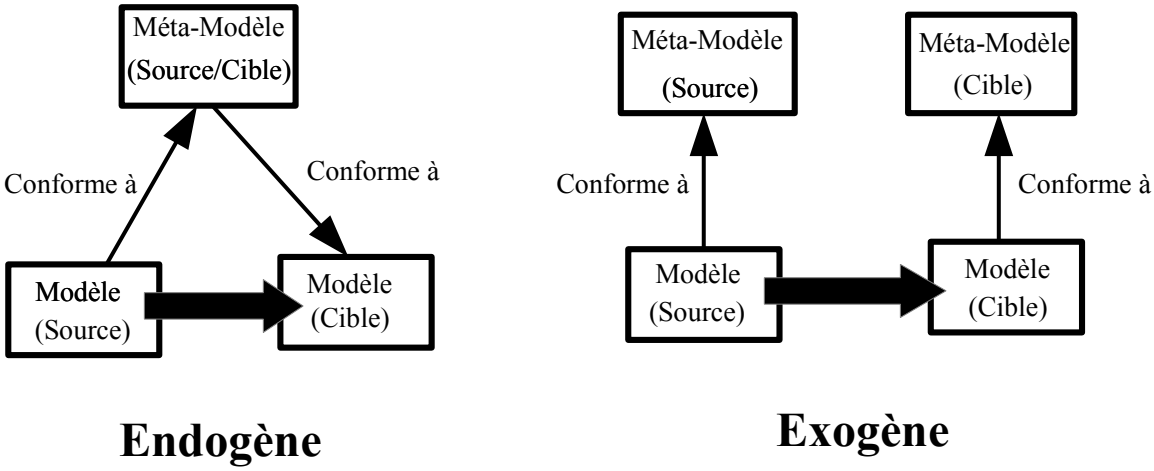


FIGURE III.6 – Type de Transformation

3.5 Eclipse Modeling Framework

L'outil principal utilise durant ces travaux est Eclipse Modeling Framework (EMF), c'est un kit de composants logiciels structurels (plus couramment appelé framework) pour la construction d'outils basées sur une approche modèle. Les modèles y sont décrits en XML, mais peuvent être spécifies dans des documents UML/SysML. L'interopérabilité avec d'autres outils bases sur Eclipse est un des points forts de ce Framework. Un de ces outils appelé Graphical Modeling Framework (GMF) permet le développement d'éditeur graphique de modèle. L'utilisation de GMF apporterait à ces travaux un rayon d'utilisation plus important grâce à un éditeur graphique spécifique aux diagrammes d'activités.

Le langage de transformation de modèle ATL (Atlas Transfor-ming Language), a été intègre à EMF [26].

Langages/outils dédiés à la transformation de modèles

Dans cette catégorie, on retrouve les outils et langages conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standard. Parmi ces outils nous pouvons citer Mia-Transformation de Mia-Software, et le plug-in ADT qui implémente le langage ATL du groupe ATLAS de l'INRIA-LINA. Dans la suite de cette section, nous présentons essentiellement le langage ATL qui est représentatif de cette catégorie des langages dédiés aux transformations, et fait l'objet de nombreuses expérimentations dans la communauté IDM.

3.6 ATL

ATL (ATLAS, 2005), acronyme de ATLAS Transformation Language, est un langage à vocation déclarative, mais en réalité hybride, qui permet de faire des transformations de modèles aussi bien endogènes qu'exogènes. une équipe du laboratoire de l'INRIA de Nantes a développé le langage ATL (Atlas Transformation Language)[33]servant à la transformation de modèle. Ce langage est proche du stan-dard QVT(Querry,View,Transformation), propose par l'OMG. Cette ressemblance est historique puisque ATL est la première tentative d'implémentation de QVT. Les outils de transformation liés à ATL sont intégrés sous forme de plug-in ADT (ATL Development Tool) pour l'environnement de développement Eclipse. Un modèle de transformation ATL se base sur des définitions de métamodèles au format XMI. Sachant qu'il existe des dialectes d'XMI, ADT est adapté pour interpréter des métamodèles décrits à l'aide d'EMF (Eclipse). Afin d'assurer son indépendance par rapport aux autres outils de modélisation, ADT met à disposition le langage KM3 (Kernel MetaMetaModel) qui est une forme textuelle simplifiée d'EMOF permettant de décrire des métamodèles et des modèles. En effet, ADT supporte les modèles décrits dans différentes dialectes d'XMI, qui peuvent par exemple être décrits au format KM3 et transformés au format XMI voulu. ATL est défini par un modèle MOF pour sa syntaxe abstraite et possède une syntaxe concrète textuelle. Pour accéder aux éléments d'un modèle, ATL utilise des requêtes sous forme d'expressions OCL [33].

Règles de transformation

Il existe plusieurs types de règles de transformation selon la manière dont elles sont appelées et le type de résultat qu'elles retournent

CalledRule : Règle appelée explicitement en utilisant son nom et en initialisant ses paramètres.

MatchedRule : Règle exécutée lorsque qu'un schéma type (InPattern) est reconnu dans le modèle source. Le résultat d'une règle est soit un ensemble d'éléments de modèles prédéfini (OutPattern), soit un bloc d'instructions impératives (ActionBlock). Une règle est dite déclarative s'il s'agit d'une MatchedRule et si son résultat est un ensemble d'éléments du modèle cible (OutPattern). Une règle de type CalledRule dont le résultat est un bloc d'instructions est appelée Procédure. Le cas échéant, il s'agit d'une règle hybride (déclarative et impérative).

Etablir une règle de base en ATL : Un programme ATL est compose de règles qui spécifient comment les éléments du modèle cible doivent être crées en fonction des éléments présents dans le modèle source. Ces règles sont toujours établies suivant le schéma suivant :

- **rule**, **from** et **to** sont les instructions du langage.
- **i** (resp. **o**) est le nom de la variable qui dans le corps de la règle représente une instance de la méta-classe source (resp. cible) dans le modèle source (resp. cible).
- **Attribut1,2** (resp. A,B) sont les attributs de la méta-classe cible (resp. source) du méta-modèle cible (resp. source).

```
1 rule Nom de la regle
2     from
3         i: Nom MM Source!Nom Meta Classe Source
4     to
5         o: Nom MM Cible!Meta Classe Cible
6                 Attribut1 <-  i.AttributA,
7                 Attribut2 <- i.AttributB+i.AttributA,
```

Listing III.1 – Etablir une règle de base en *ATL*

Le point d'exclamation permet de spécifier à quel méta-modèle appartient une méta-classe.

Une traduction en langage courant donnerait : La règle Nom de la règle crée pour chaque instance i identifiée dans le modèle source une instance o de la méta-classe cible dans le modèle cible, en donnant à Attribut1 la valeur de AttributA et à *Attribut*2 la valeur de la somme AttributA plus AttributB.

ATL permet de factoriser les règles avec l'utilisation de helper, que l'on peut assimiler à des fonctions [26].

Helpers : Un helper est l'équivalent d'une méthode auxiliaire, Il est défini sur un contexte et pourra être appliquée sur toute expression ayant pour type ce contexte (comme une méthode dans une classe en Java), un helper peut prendre des paramètres et possède un type de retour, le code d'un helper est une expression OCL.

4 Conclusion

Dans ce chapitre, nous avons défini les principaux concepts du domaine de l'ingénierie des modèles en mettant en évidence le rôle central des modèles. Dans une approche IDM, la transformation de modèle joue un rôle très important, elle permit de passer d'un modèle à un autre. d'ailleurs, des langages de transformation ont été développés et implémentés. Parmi ces langages on trouve ATL. Notre travail sert de la transformation de modèles, pour passer des diagrammes d'activité vers les réseaux de Petri.

Chapitre IV

Méthodologie

1 Introduction

L'ingénierie des systèmes basée sur le modèle (MBSE) devient une solution prometteuse pour la conception de systèmes complexes et critiques. Pour vérifier ces dernier on trouve des techniques de vérification basées sur le modèle checking sont fondées sur l'examen automatique de tous les états accessibles pour vérifier si un modèle d'un système donné satisfait les propriétés souhaitées. Cependant, comme cela est connu, le défaut majeur des méthodes basées sur le modèle checking est réside dans le problème de l'explosion combinatoire de l'espace d'états qui apparaît lorsque l'on considère les systèmes de taille réelle. Ceci rend difficile la vérification de toutes les configurations possibles. Beaucoup d'approches ont été proposées pour réduire l'espace d'etats, A travers ce chapitre nous présentons Une méthodologie basée sur le découpage de modèle pour la vérification des diagrammes d'activités SysML. Ce découpage sera déduit à partir de l'exigence à vérifier.

2 Vérification des spécifications SysML

SysML est un Profil UML proposé pour spécifier les systèmes qui incluent des Composants hétérogènes. Il couvre quatre perspectives sur le système à modéliser : la structure, le comportement, les exigences et les paramètres. Le diagramme des exigences est une des plus importantes extensions fournies dans SysML. Grâce au diagramme d'exigences, SysML fournit une meilleure organisation et décrit les exigences à différents niveaux d'abstraction. Malgré les divers avantages de SysML, il reste un langage semi-formel sans possibilités de vérifier formellement les modèles décrits par celle-ci. Pour cette raison, en utilisant SysML conjointement avec des méthodes formelles qu'est suggéré de fournir une validation formelle pour les spécifications SysML.

3 Vérification des exigences sur les diagrammes d'activités

La vérification dans notre approche est orientée par les relations SysML entre les exigences, les blocs et les activités, ce qui permet l'utilisation d'une technique basée sur

le découpage de modèle pour la vérification des diagrammes d'activités SYSML. Pour vérifier une exigence donnée, notre travail exploite deux relations :
La relation «satisfy » est une dépendance entre une exigence et un bloc SysML, cela signifie que l'exigence est satisfaite par le bloc.
La relation « Verify » est une dépendance entre une exigence et un cas de test, Cela indique qu'il est utilisé pour vérifier l'exigence, le cas de test peut être une séquence, une machine d'état ou un diagramme d'activité (le cas de notre travail). Comme représenté sur la figure IV.1

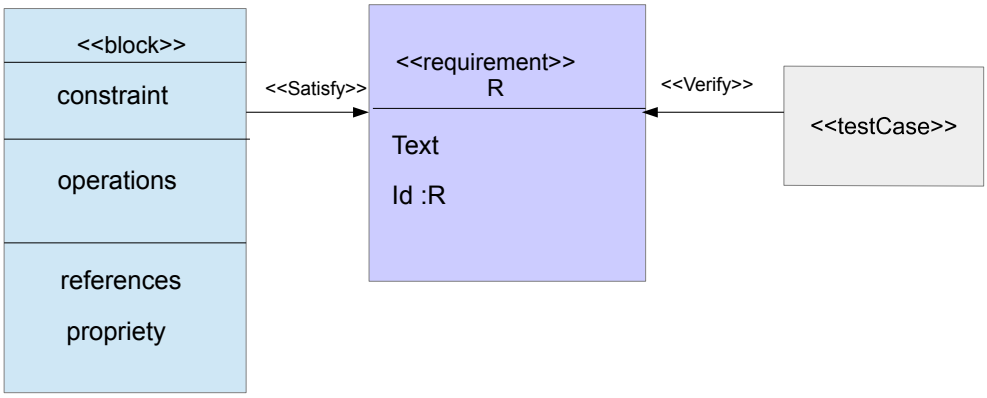


FIGURE IV.1 – «satisfy» et «verify» relations dans les diagrammes d'exigence

3.1 Démarche globale

Pour la vérification des exigences sur les diagrammes d'activité. Dans notre démarche, on se basant sur la transformation de diagramme d'activité aux réseaux de Petri colorés hiérarchiques, pour chaque exigence dans le diagramme SysML, nous extrayons les identifiants du bloc et l'activité qui le satisfont respectivement et vérifient, nous exploitons le diagramme d'exigences avec les relations « satisfy » et «verify» Après cela, on s'intéresse à la proposition d'une approche qui se base sur le découpage de modèle pour vérifier des diagrammes d'activité SySML afin de pallier au problème d'explosion combinatoire de l'espace d'états.

3.2 Idée de découpage

Notre idée de découpage se base sur la transformation des diagrammes d'activité avec des couloirs, la considération des couloirs permet de représenter les aspects structurels du système. Pour vérifier une exigence donnée, le découpage consiste à extraire un fragment

minimisée (Slice) du modèle de réseau de Petri coloré hiérarchique (traduit du diagramme d'activité), qui est solide et suffisante pour réaliser la vérification. Ceci permet d'alléger le processus de vérification des systèmes complexes avec de nombreux composants de tel manière nous extrayons l'identifiant du bloc et l'activité qui sont présentés dans le diagramme d'exigence pour diviser le RDPHC traduit de l'ensemble du diagramme d'activité.

3.3 Méthodologie

La méthodologie que nous avons proposé est bien présentée par la figure IV.2. Elle prend comme entrées un diagramme d'exigence qui comporte une ou plusieurs exigences satisfont des propriétés dans le diagramme d'activités et vérifient le bloc qui lui associe. En suite, nous passons par deux étapes, la première consiste à transformer le diagramme d'activité au réseaux de Petri, puis le format résultant "PNML" sera transformé vers le format "CPNtools" avant le découpage (slicing) pour éviter d'effectuer de nombreuses transformations (une transformation après chaque découpage du module). et la deuxième étape sert à effectuer le découpage de modèle pour vérifier le diagramme d'activité SysML. Enfin, dans la dernière phase.

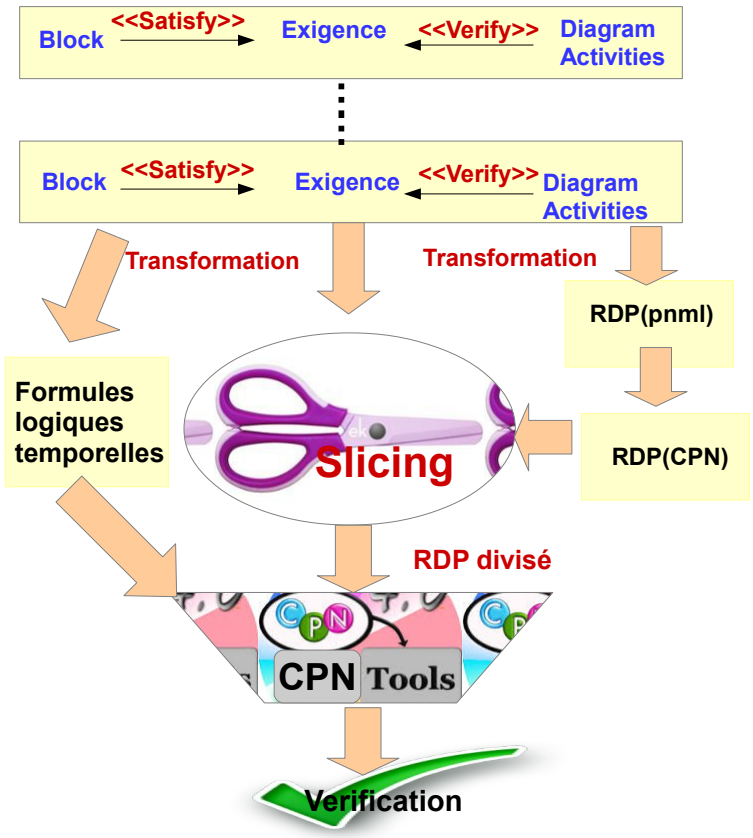


FIGURE IV.2 – Démarche globale

4 Conclusion

Ce chapitre présente d'une façon générale l'intérêt des méthodes formelles pour la vérification des spécifications SysML et particulièrement le passage vers des réseaux de Petri hiérarchiques colorés à partir des diagrammes d'activité. Nous avons évoqué dans ce chapitre, le problème d'explosion combinatoire d'espace d'états. Ainsi, visant à proposer une solution pour réduire ce problème, nous avons présenté un aperçu global sur la méthodologie de vérification se basant sur le découpage de modèle et son application pour vérifier des exigences sur les diagrammes SysML.

Chapitre V

Translation des diagrammes d'activité SysML en Réseaux de Petri

1 Introduction

Dans ce chapitre, nous tenterons d'établir les concepts nécessaires à une transformation automatique des diagrammes d'activités SysML conforme à la spécification définie par l'OMG vers les réseaux de Petri, de vérifier formellement cette transformation avant d'utiliser les réseaux de Petri ainsi nous présenterons la transformation de modèle pnml obtenu à un fichier xml qui est compréhensible par l'outil CPNTools.

2 Transformation des DA vers les RdP

Cette section présente les méta-modèles source et cible, dans un premier temps le formalisme sera mis en avant, avant de modéliser celui-ci en faisant apparaitre la démarche de méta-modélisation (figure V.1).

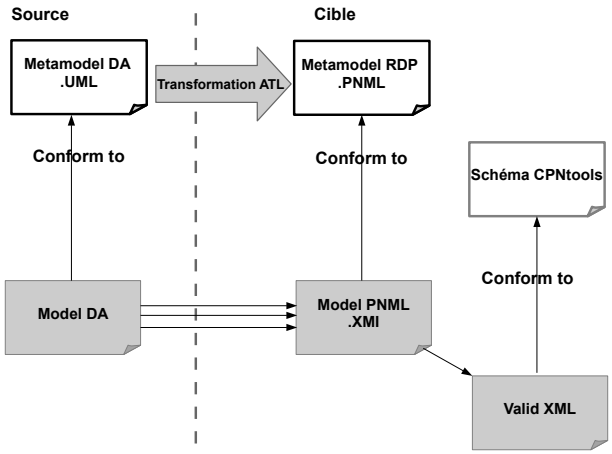


Figure V.1 – Transformation de DA vers RDP

2.1 Méta-modèles

La première étape dans la transformation que nous proposons consiste à définir un méta-modèle pour les RdP et un méta-modèle pour les DAs. Pour arriver a implémenter les règles de transformations d'un modèle vers un autre, il faut d'abord bien citer les méta modèles cibles et sources.

2.1.1 Méta-modèle Diagramme d'activité (MMDA)

Le diagramme d'activité est un des quatre diagrammes comportementaux de SysML. Il est utilisé pour décrire le comportement du flux de contrôle et de données des systèmes complexes. Il est donc parfaitement adapté pour décrire le déroulement d'un cas d'utilisation. Il est composé des nœuds et d'arcs. Ces nœuds sont divisés en deux parties, les nœuds de contrôle et les nœuds d'activités. Chacun de ces nœuds est ensuite spécialisé et peut grâce aux arguments associés avoir un comportement sensiblement différent, c'est pourquoi nous ne présenterons pas en détail chacune des possibilités offertes par SysML et préférons renvoyer à la spécification OMG. Il sera tout de même présenté dans ce paragraphe les principaux nœuds du diagramme. Nous avons extrait le méta-modèle du diagramme d'activité de la spécification SysML et UML de l'OMG. En effet, l'OMG spécifie le diagramme d'activité SysML en ce basant sur celui d'UML. La figure V.2 présente une traduction en diagramme de classe UML du MMDA présenté dans la spécification OMG. On ne présentera qu'une petite partie de ce méta-modèle, Comme dit précédemment, les diagrammes d'activité sont composés de nœuds, nous allons nous intéresser ici à ActivityNode, Activity, ActivityEdge, ActivityPartition.

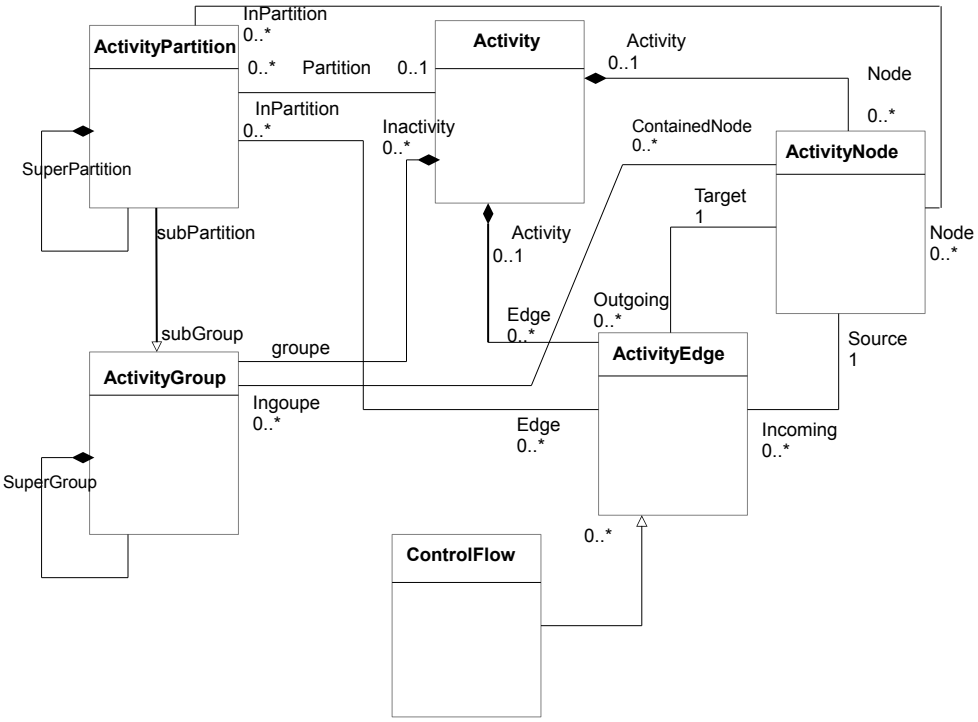


FIGURE V.2 – La partie supérieure de méta-modèle DA

2.1.1.1 Description de Méta-Modèle

Dans cette partie nous allons présenter seulement les parties qui permettent de mettre l'accent sur la translation ATL. Une activité définit un comportement décrit par un séquencèrent organisé d'unités dont les éléments simples sont les actions. Cette dernière a une relation avec la Méta-classe Activiy Partition.

Les partitions, souvent appelées couloirs (swimlane) du fait de leur notation, permettent d'organiser les nœuds d'activités dans un diagramme d'activités en opérant des regroupements. La figure V.3 présente la méta-classe "ActivityPartition" une méta-classe possédant deux liens d'héritage avec les méta-classes "Activitygroupe" et "NamedElment". ainsi, elle a des relations avec les méta-classes Activity ,ActivityNode et Activityedge.

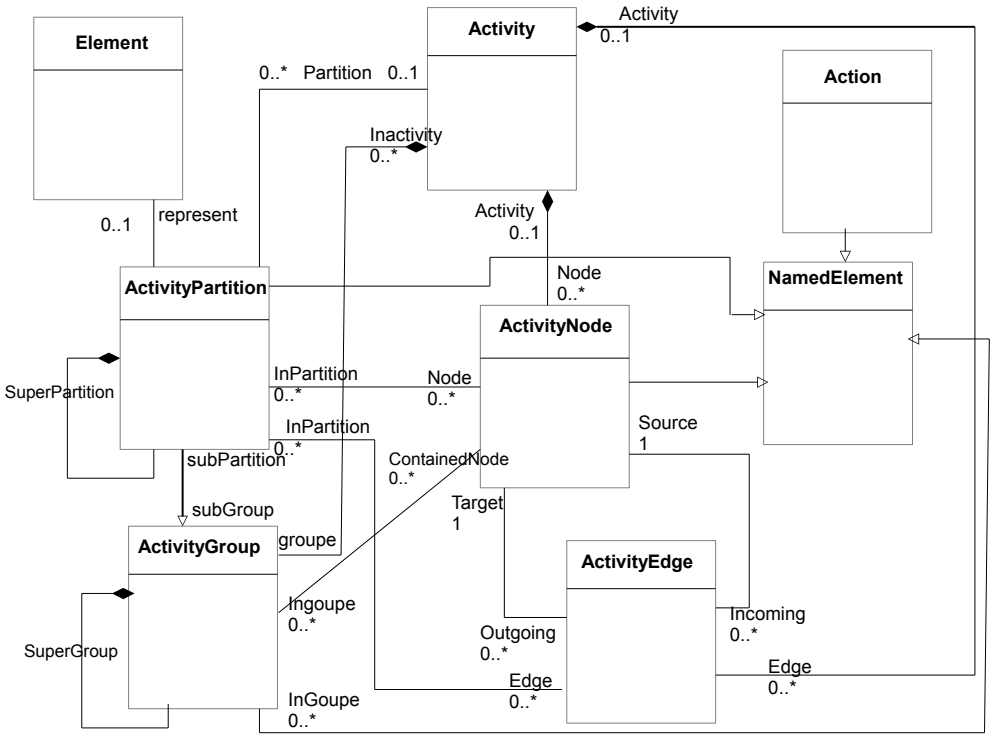


FIGURE V.3 – Les partitions de l'activité

Une ActivityPartition peut avoir des liens entrants et sortants. La figure V.4 présente la méta-classe "ActivityEdge" qui possède deux liens d'héritage avec les méta-classes "ObjectFlow" et "ControlFlow". Un flux d'objet est un activityedge qui peut avoir des objets ou des données passant le long de lui.

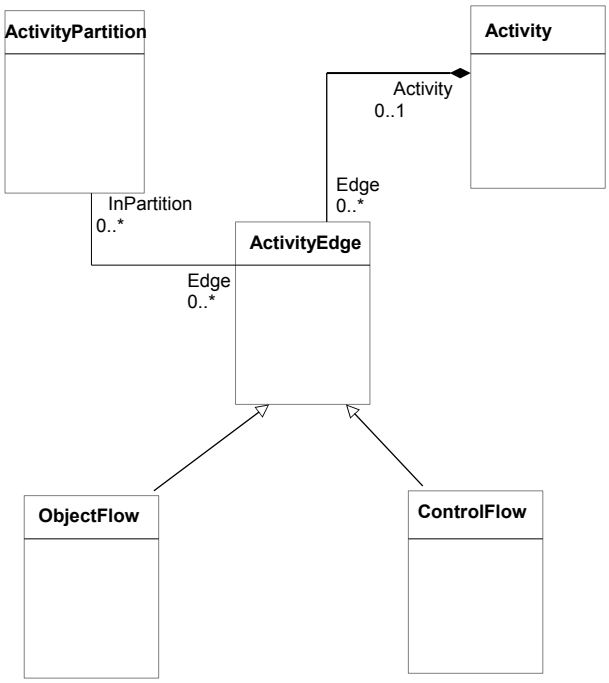


FIGURE V.4 – Les flux d'activité(ActivityEdge)

2.1.2 Méta-Modèle RDP

Maintenant que le méta-modèle source a été crée il faut définir le méta-modèle cible. Dans l'ingénierie dirigée par les modèles on pourrait dire : il faut modéliser le formalisme des RdP, ou bien encore méta-modéliser les RdP. Avant tout, il est important de redéfinir clairement les réseaux de Petri voulu pour cette transformation afin d'établir un méta-modèle conforme à nos attentes, et compatible avec l'outil CPNtools. Avant d'établir le méta-modèle des RdP, il est intéressant de montrer le principe de création du méta-modèle RdP. La figure V.5 présente un méta-modèle RdP simplifié.

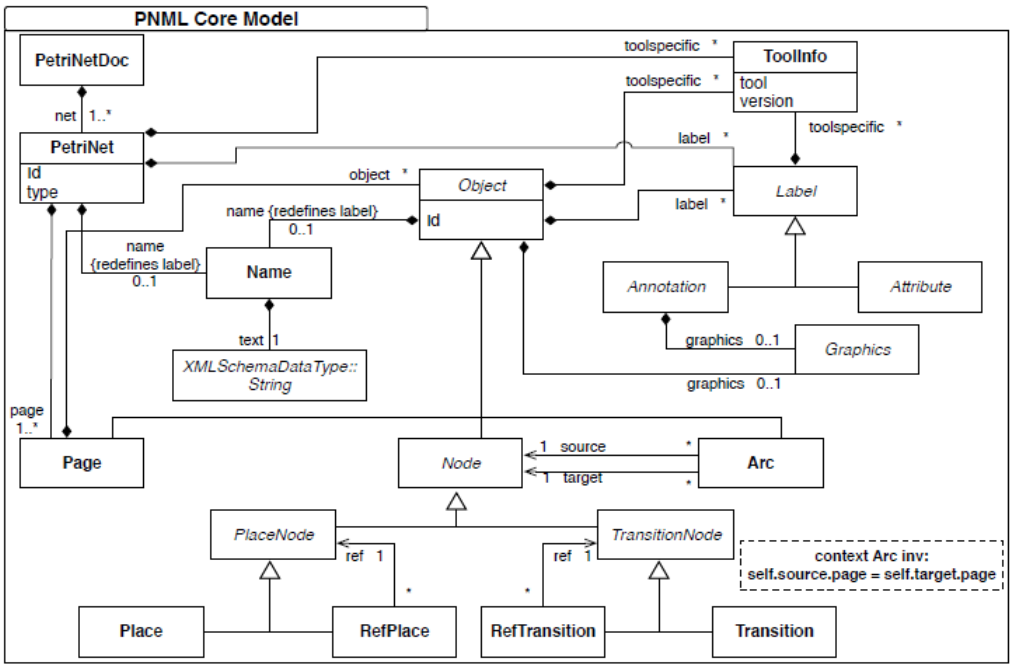


FIGURE V.5 – méta-modèle RdP

2.1.2.1 Description de méta-modèle

Un PetriNet (réseau de Petri) est composé de Node (noeud) et d'Arc (arc). Node peut être une Transition ou une Place reliés par des Arcs. Un Arc ne peut avoir qu'une seule Source et qu'un seul Target (cible). On voit également qu'un PetriNet, un Node, une Transition, une Place reçoit par héritage un attribut de type Name permettant de nommer explicitement les instances de ces méta-classes. pour faire la différence entre une transition et une transition de substitution on a ajouté au niveau de la méta-classe Transition l'attribut type et l'attribut condition pour préciser des conditions de franchissement, on a ajouté également la relation RefPage qui permet à une transition de substitution de référencer vers une et une seul Page (figure V.6).

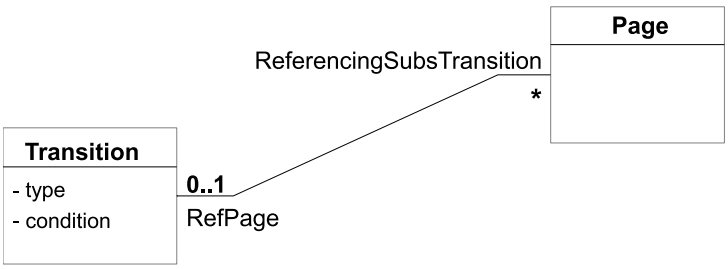


FIGURE V.6 – Les modifications effectué sur MMPNML

La figure suivante V.7. Représente le Méta-Modèle cible sous son extension Ecore

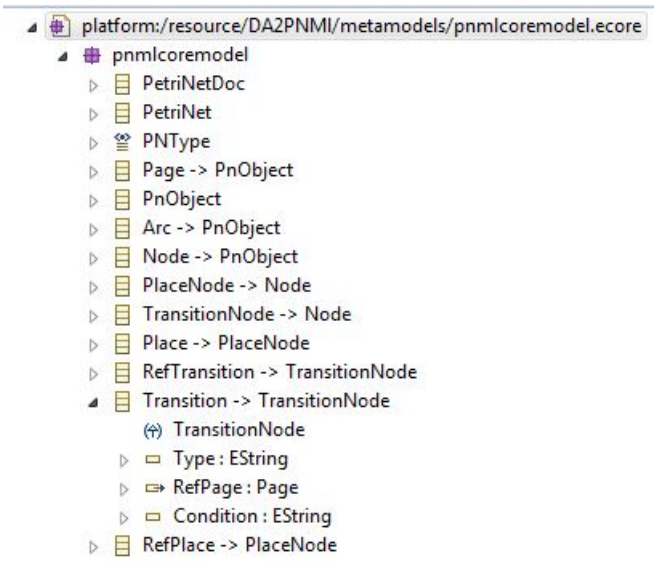


FIGURE V.7 – Extension Ecore

2.2 Processus de transformation

Il existe de nombreux langages pour effectuer les transformations de modèle. Notre choix s'est porté sur le langage ATL (ATLAS Transformation Language) qui est intégré à la boite d'outils PolarSys d'Eclipse. La figure V.8 montre plus en détail le fonctionnement de la transformation dans l'environnement de travail choisi en vue d'une transformation ActivityDiagram2PetriNet. Pour effectuer une transformation de modèle, il faut :

— Le méta-méta-modèle : Ecore.

— Le méta-modèle source conforme au méta-méta-modèle Ecore qui décrit le langage de source : Les Diagrammes d'activité SysML.

— Le méta-modèle cible conforme au méta-méta-modèle Ecore qui décrit le langage de modélisation cible : Les réseaux de Pétri.

— Le modèle source qui décrit un système en respectant la sémantique et la syntaxe des Diagrammes d'activité.

— Les règles de transformation en ATL.

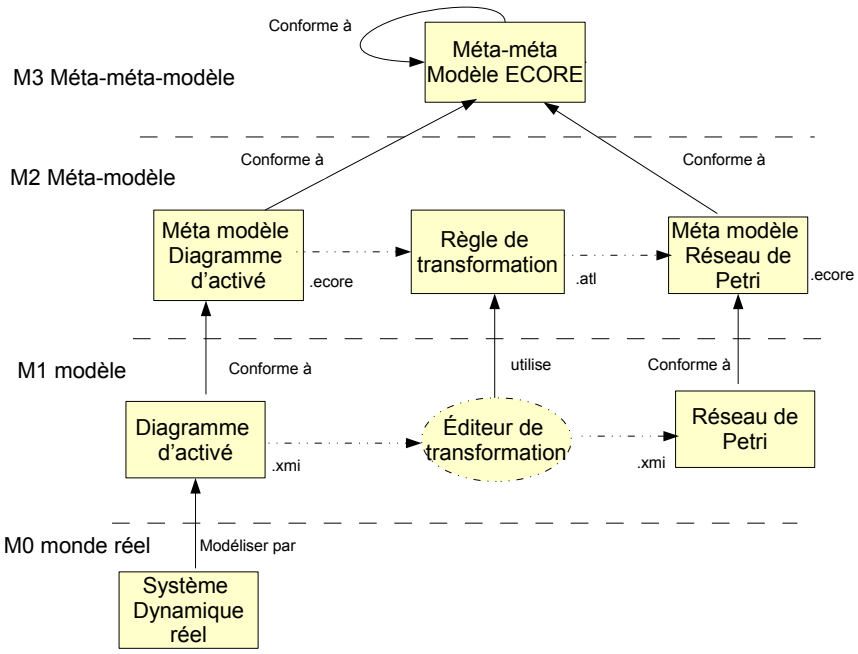


FIGURE V.8 – Concept de la transformation

2.2.1 Règles de transformation

Il faut établir des règles de transformation pour passer du modèle source au modèle cible. Classiquement, il suffit d'élaborer les règles à partir des éléments du modèle source pour les transformer en éléments désirés du modèle cible. La transformation naturelle des éléments de modèle du diagramme d'activités consiste à transformer chaque élément en ensemble des éléments places arcs et transitions dans le RDP.

2.2.1.1 Transformation des entités de diagramme d'activité

Ce paragraphe va établir les concepts de la transformation entre les entités atomiques du DA(noeud/arc) et un bloc équivalent RdP. Ces blocs devront respecter le comportement (sémantique) du DA préconise par l'OMG. A première vue, un réseau de Petri peut sembler proche d'un diagramme d'activité. Leurs représentations graphiques et leurs comportements (déplacement de jetons dans le "réseau", composition des graphes : nœud, arc) nous a permet de générer un réseau de Petri sous la forme de modèle pnml nous présentons dans la suite la transformation de chaque élément et sa règle de transformation pour les éléments du DA avec partitions vers RDP.

— **Modèle et Activité :**
Un *DA* est un modèle, ce modèle va être transformé en un *RdP* qui va avoir le même nom que le modèle. Le *DA* peut être composé de plusieurs activités, chaque activité sera transformé en une page du *RdP*. Cette transformation est représentée dans la

figure V.9. Comme résultat, nous obtenons un *RdP* hiérarchique dont chaque page représente une sous activité du *DA*.

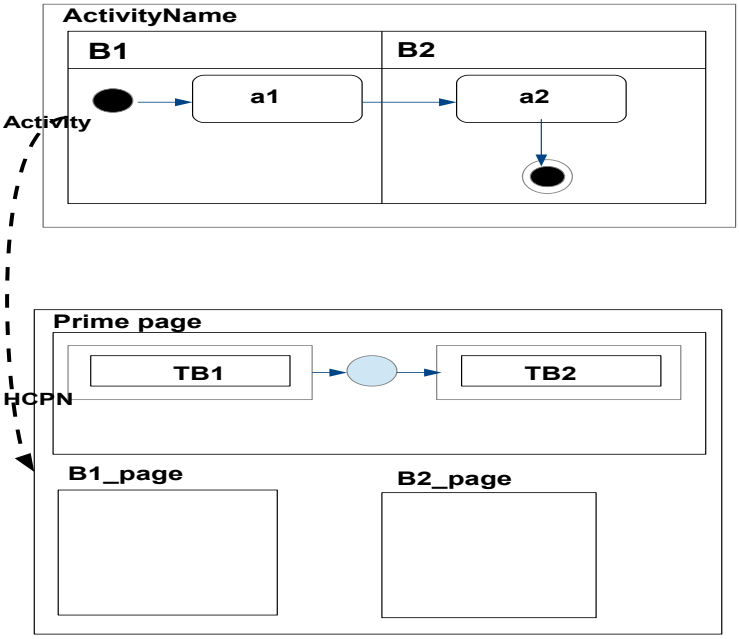


FIGURE V.9 – Transformation de "Model"

Dans ce qui suit, le code ATL de la règle "*PetriNet*" est présenté par le listing ,V.1, et celui de la règle "*Activity*" se situe dans la partie annexe 5 par le listing 4 :

PetriNet

```
1 rule PetriNet {
2   from
3     model: MMuml!Model
4   to
5     pn: MMpn!PetriNet (
6       id <- model.name,
7       pages <- model.packagedElement -> collect(e | thisModule.
    resolveTemp(e,'page'))
8
9     )
10 }
```

Listing V.1 – Le code *ATL* de la règle "*PetriNet*"

— **Partition d'activité (ActivityPartition)**

Dans le DA qui contient des Partitions, chaque partition d'activité est transformée en transition de substitution et une page en RDP qui regroupe touts les éléments de la partition. Comme la figure V.10 présente.

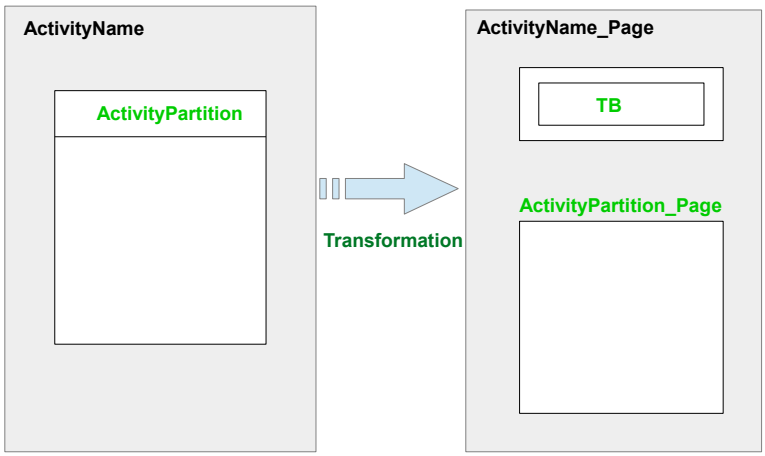


FIGURE V.10 – Transformation de partition d'activité

Le listing V.2 décrit l'implémentation de la règle de transformation d'une partition d'activité comme suit : **ActivityPartition**

```
1  rule ActivityPartition {
2   from
3   ActPart: MMuml!ActivityPartition
4
5   to
6      substrns:MMpn!Transition (
7         id <- 'TB_' + ActPart.name,
8       Type <- 'SubsTransition',
9         RefPage <- thisModule.resolveTemp(ActPart, 'page'),
10       containerPage <- thisModule.resolveTemp(ActPart.inActivity, '
    page'),
11       InArcs <-  thisModule.resolveTemp(ActPart.containedEdge, '
    outarc2'),
12         OutArcs <- thisModule.resolveTemp(ActPart.containedEdge, '
    inarc2')
13       ),
14     page: MMpn!Page (
15       id <- 'Page_' + ActPart.name,
16       containerPage <- thisModule.resolveTemp(ActPart.inActivity, '
    page')
17     )
18
19 }
```

Listing V.2 – Le code *ATL* de la règle "*ActivityPartition*"

— **Flux de contrôle (Controlflow)**

Dans notre transformation il faut faire la différence entre les arcs dans les partitions

qui consistent à relier les nœuds de la partition et les arcs hors les partitions qui consistent à relier les partitions d'une activités. Pour cela, on a implémenté un helper qu'est représenté par le Le listing V.5.

Helper de Controlflow

```
helper context MMuml!ControlFlow def: typecntflow():Boolean =
  if self.inPartition->isEmpty() then
    true
  else
    false
  endif;
```

Listing V.3 – Le code *ATL* de la règle "*HelperdeControlflow*"

Chaque arc est transformé en un arc qui se dirige vers une place, une place et un autre arc sort de cette place. Comme la figure V.11 présente.

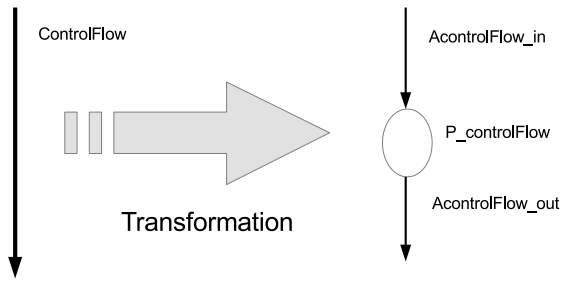


FIGURE V.11 – Transformation de DA vers RDP DA

Le code ATL des ControlFlow dans les partitions

```
rule ctlflw {
  from
    ctl: MMuml!ControlFlow(not ctl.typecntflow())
  to
    inarc: MMpn!Arc (
      id <- 'A_' + ctl.name+ctl.activity.name + '_in',
            containerPage <- thisModule.resolveTemp(ctl.inPartition->
    first()  , 'page'),
          source <- Sequence{thisModule.resolveTemp(ctl.source, 'trst
    '),
                thisModule.resolveTemp(ctl.source, 'trst1'),
                thisModule.resolveTemp(ctl.source, 'EndTrst'),
                  thisModule.resolveTemp(ctl.source, 'SubsTrns')
```

```
12              },
13      target <- ArcPlc
14    ),
15    ArcPlc: MMpn!Place (
16      id <- 'P_in_out_' + ctl.name+ctl.activity.name,
17      containerPage <- thisModule.resolveTemp(ctl.inPartition->first()
    , 'page'),
18      referencingPlaces <- Sequence{thisModule.resolveTemp(ctl.target,
     'refPlacein'),
19              thisModule.resolveTemp(ctl.source, 'refPlaceout')
20              }
21    ),
22    outarc: MMpn!Arc (
23      id <- 'A_' + ctl.name+ctl.activity.name + '_out',
24      containerPage <- thisModule.resolveTemp(ctl.inPartition->first()
    , 'page'),
25      source <- ArcPlc,
26      target <- Sequence{thisModule.resolveTemp(ctl.target, 'ActStart
    '),
27              thisModule.resolveTemp(ctl.target, 'start'),
28              thisModule.resolveTemp(ctl.target, 'substrns')
29              }
30    )
31}
```

Listing V.4 – La règle "Controlflow dans la parttion"

Le code ATL des ControlFlow hors les partitions

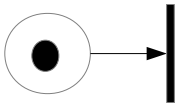
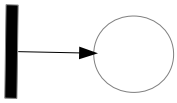
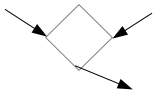
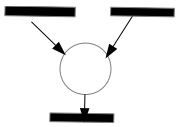
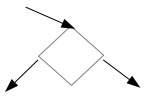
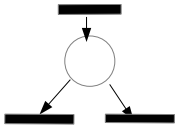
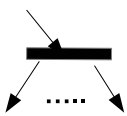
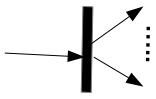
```
1rule ctlflw2 {
2  from
3    ctl: MMuml!ControlFlow(ctl.typecntflow())
4  to
5    inarc2: MMpn!Arc (
6      id <- 'A_' + ctl.name+ctl.activity.name + '_in',
7          containerPage <- thisModule.resolveTemp(ctl.activity, 'page
    '),
8            source <-thisModule.resolveTemp(ctl.source.inPartition->
    first(), 'substrns'),
9      target <- ArcPlc2
10    ),
11    ArcPlc2: MMpn!Place (
12      id <- 'P_in_out_' + ctl.name+ctl.activity.name,
13      containerPage <- thisModule.resolveTemp(ctl.activity, 'page'),
14      referencingPlaces <- Sequence{thisModule.resolveTemp(ctl.target,
     'refPlacein'),
15                                thisModule.resolveTemp(ctl.source, '
    refPlaceout')
16              }
17    ),
```

```
18      outarc2: MMpn!Arc (
19        id <- 'A_' + ctl.name+ctl.activity.name + '_out',
20        containerPage <- thisModule.resolveTemp(ctl.activity, 'page'),
21        source <- ArcPlc2,
22        target <- thisModule.resolveTemp(ctl.target.inPartition->first()
      , 'substrns')
23      )
24  }
```

Listing V.5 – La règle "Controlflow hors la parttion"

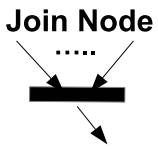
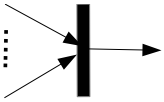
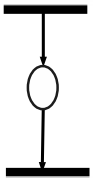
— **Quelques transformations des autres objets de DA :**
Le reste des règles de transformation des autres objets de base de DA sont décrit par la table 2.1 :

TABLE 2.1: Table contient le reste des règles de transformations

Objets SysML	Blocs *RdP*	Solution
Initiale Node 		Création d'une place marqué avec un marquage initial avec une transition permet de fournir à chaque arc sortant un jeton.
Finale Node 		Création d'une transition pour chaque arc entrant, afin d'activer la place finale, donc démarqué l'ensemble d'activités.
Merge Node 		Création d'une transition pour chaque entrée du Merge et l'ajout d'une place qui distribue directement chaque jeton en sortie.
Decision Node 		Création des transitions de sortie en fonction du nombre de sortie de décision.
Fork Node 		Création d'une transition qui permet de redistribuer autant de jetons que d'arcs sortant de façon synchrone.

à suivre dans la page suivante

TABLE 2.1 – *Suivant*

Objets SysML	Blocs *RdP*	Solution
		Création d'une transition qui a les même propriétés de synchronisation dans les RDP.
		Création de deux transitions, deux arcs et une place .

Le code ATL correspond au reste des règles de transformation sont décrites dans la partie annexe 5.

Pour relier chaque transition de substitution avec leur page, on a ajouté des objets supplémentaires (transition, arc et RefPlace) dont on a implémenté deux helpers :
Un helper pour les nœuds qui ont au moins un ControlFlow hors les partitions en entrée (incoming).

Helper arc (incoming)

```
1 helper context MMuml!OpaqueAction def:element() :Boolean=
2  if self.incoming-> select(e | e.inPartition->isEmpty())->notEmpty()
3   then true
4    else
5    false
6   endif;
```

Listing V.6 – Le code ATL de helper de arc incoming

Et un autre pour les nœuds qui ont au moins un ControlFlow hors les partitions en sorties (outgoing).

Helper arc (outgoing)

```
1 helper context MMuml!OpaqueAction def:element1() :Boolean=
2  if self.outgoing-> select(e | e.inPartition->isEmpty())->notEmpty()
3   then true
4    else
5    false
6   endif;
```

Listing V.7 – Le code *ATL* de helper arc outgoing

-Dans le cas d'un arc entrant (incoming) se situe dans la partie annexe 5 par le listing 8.
-Dans le cas d'un arc sortant (outgoing) : présenté aussi dans l'annexe 5 par le listing 9.
-Dans le cas d'un arc entrant et autre sortant d'un même nœud qui est présenté dans la

partie annexe 5 par le listing 10.

3 Transformation modèle-texte

Le résultat de la transformation produit un modèle du système entier qui peut être ensuite être simulé par l'outil CPNtools. Grâce à la similarité de la sémantique de notre méta-modèle et de l'outil CPNtools, une fois la transformation modèle–modèle est Réalisée (DA vers RDP), une transformation modèle vers texte est ensuite exécutée pour construire un fichier .cpn qui est conforme au fichier DTD de l'outil CPNtools. Ce dernier comporte un model-checker qui aide à la vérification, puis le résultat de cette transformation devient comme entrée dans notre outil de découpage (slicing).

3.1 Format CPN

Le format XML pour les fichiers .cpn est décrit en utilisant une DTD, et pour un fichier soit valide par CPNtools il doit comporter des pages, des places, arcs et des transitions, la structure d'un fichier .cpn est comme ceci :

```
1<workspaceElements>
2   <generator .../>
3   <cpnet>
4     <globbox>
5       ...
6     </globbox>
7     <page id="...">
8       <pageattr name="...">
9       <place id="...">
10        <posattr .../> <!-- position -->
11        <textattr .../> <!-- font and text color -->
12        <text>...</text> <!-- name -->
13        <ellipse .../> <!-- sets width and height -->
14        <token .../> <!-- position of round no of tokens -->
15        <marking .../> <!-- position (and whether hidden) of rectangular
    description of current marking -->
16        <type> <!-- color set -->
17          <posattr .../> <!-- position -->
18          <text>...</text>
19        </type>
20        <initmark> <!-- initial marking -->
21          <!-- like type -->
22        </initmark>
23        <fusioninfo id="..." name="..."> <!-- omit if not fusion  place --
    >
24          <posattr .../> <!-- position -->
25        </fusioninfo>
```

```
26      </place>
27      <trans id="..." explicit="false">
28        <posattr .../> <!-- position -->
29        <text>...</text> <!-- name -->
30        <box .../> <!-- sets width and height -->
31        <subst subpage="..." portsock="..."> <!-- omit if not
    substitution transition; check example for exact format -->
32          <subpageinfo id="..." name="..."> <!-- position of small blue
    hierarchy tag-->
33            <posattr .../> <!-- position -->
34          </subpageinfo>
35        </subst>
36        <binding x="..." y="..." /> <!-- position of binding index -->
37        <cond> <!-- guard -->
38          <!-- like type in place -->
39        </cond>
40        <time> <!-- time stamp -->
41          <!-- like type in place -->
42        </time>
43        <code> <!-- action part -->
44          <!-- like type in place -->
45        </code>
46        <channel> <!-- for transition fusion, not used, just copy from an
     example -->
47          ..
48        </channel>
49      </trans>
50      <arc id="..." orientation="TtoP|PtoT|BOTHDIR" order="..."> <!-- arc
    , order can probably just be set to 0 -->
51        <posattr .../> <!-- position -->
52        <arrowattr headsize="1.2" currentcycle="2"/>
53        <transend idref="..."/> <!-- id of transition of this arc -->
54        <placeend idref="..."/> <!-- if of place of this arc -->
55        <bendpoint id="..." serial="..."> <!-- zero or more of these,
    numbered from 1 (using serial), only posattr is used -->
56          <posattr .../> <!-- position -->
57        </bendpoint>
58        <annot id="..."> <!-- arc expression -->
59          <!-- like type in place -->
60        </code>
61      </arc>
62    </page>
63    <fusion id="..." name="..."> <!-- description for each fusion group
    -->
64      <fusion_elm idref="..."> <!-- one for each member, id of place -->
65    </fusion>
66  </cpnet>
```

```
67 </workspaceElements>
```

Listing V.8 – schéma *CPN*

3.1.1 Représentation des entités de PNML en CPN

— **PetriNet**

La racine en pnml est représentée par la balise < pnml....> dont la racine en CPN qui est representé par la balise <cpnet > </cpnet>. la figure V.12 montre les deux racines.

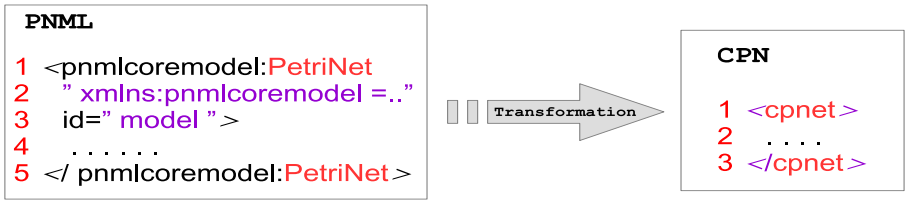


FIGURE V.12 – Transformation de "PetriNet"

— **Page principale**

La page en pnml reste toujours une page lors de la transformation, mais on trouve plus de détail en CPNtools comme l'id de la page, le nom , ...etc. ce qui montre la figure V.13.

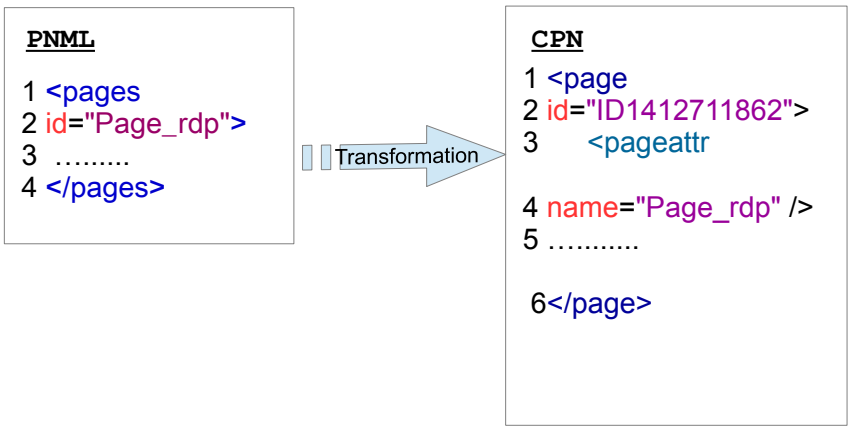


FIGURE V.13 – Transformation de "Page"

— **Transition**

La transition en pnml est plus détaillée lors de la transformation en CPNtools comme

décrit la figure V.14

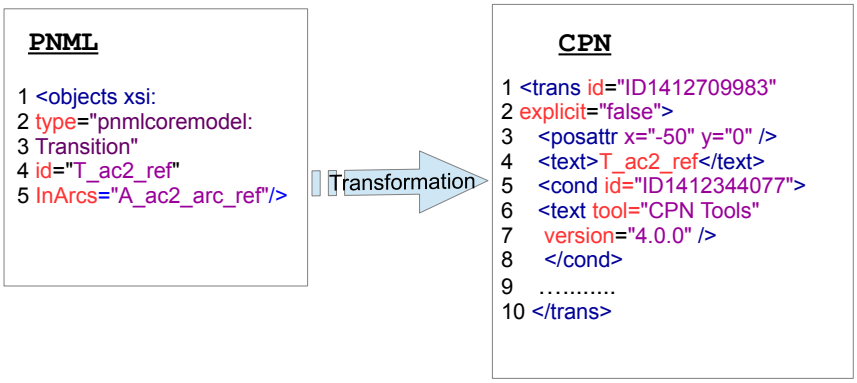


FIGURE V.14 – Transformation de "Transition"

— **Transition de subtitution**

La transformation d'une transition de substitution se diffère de la transformation d'une simple transition en ajoutant la balise <Subst.... > qui contient les deux attributs SubPage et portsock, ce dernier rassemble l'id de la place de la page principale et l'ID de la place de <subpage....> dans un couple pour faire le référence entre une transition de substitutions et sa page en CPNtools comme il est montré dans la figure V.15.

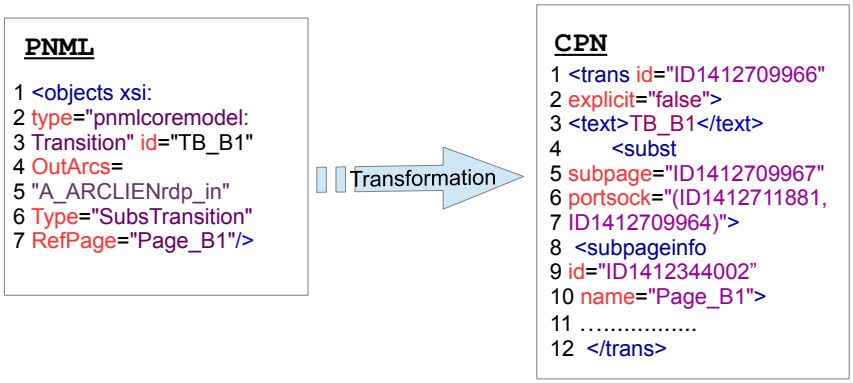


FIGURE V.15 – Transformation de "Transition de substitution"

— **Place**

La figure V.16 décrit la transformation d'une place en CPNTools.

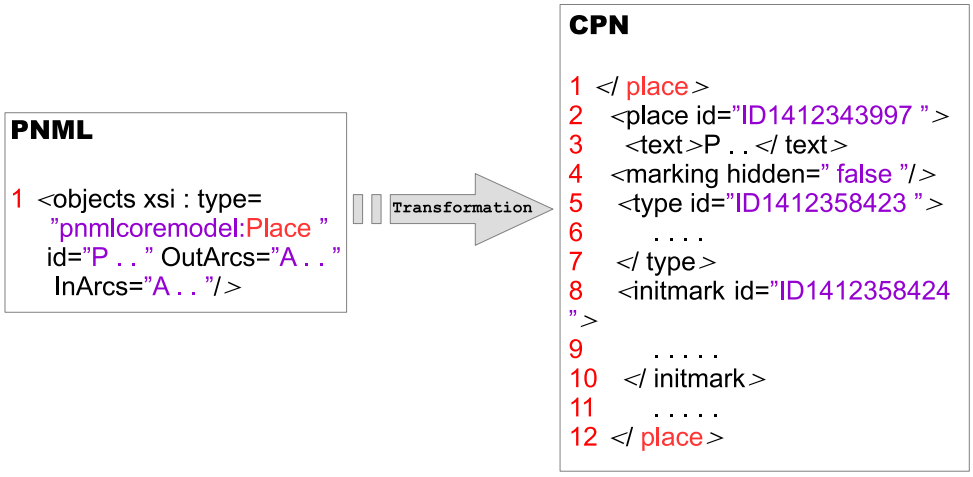


FIGURE V.16 – Transformation de "Place"

— **Place Ref**

La PlaceRef est transformée comme une place sauf que la balise <Port....> est ajoutée .

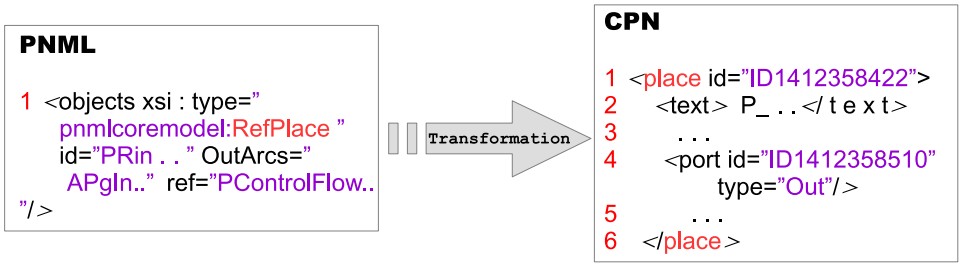


FIGURE V.17 – Transformation de "RefPlace"

— **Arc**

La transformation de l'arc varie selon l'orientation de ce dernier. Si l'arc reliant une place à une transition on trouve l'attribut orientation "PtoT". Si l'arc reliant une transition à une place on trouve l'attribut orientation "TtoP". La figure V.18 donne exactement la transformation en CPNtools.

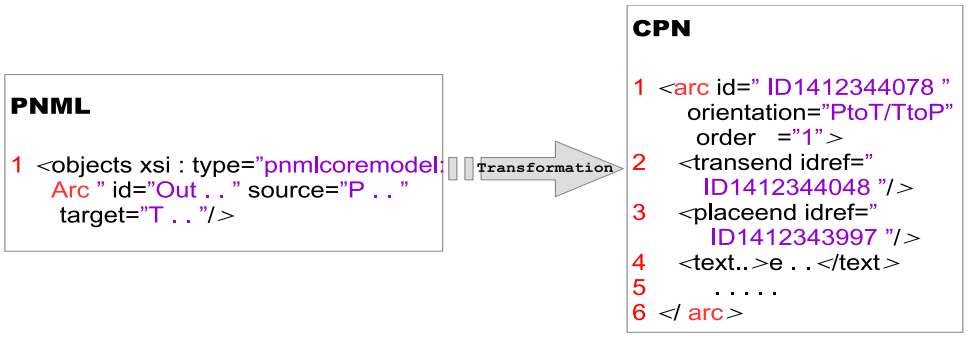


FIGURE V.18 – Transformation de "Arc"

— **Sous-Page**

En Pnml la sous-page est toujours dans la page principale, lors de la transformation elle devient comme une page hors la page principale.

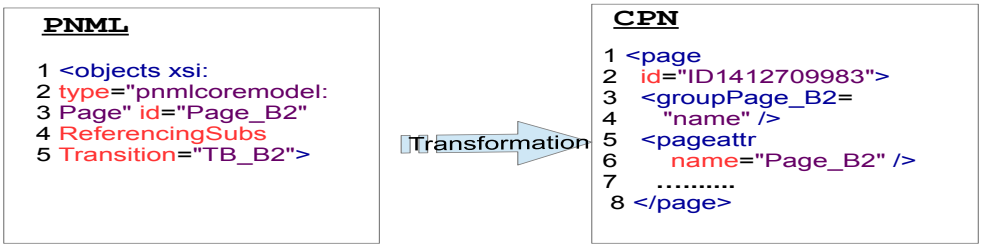


FIGURE V.19 – Transformation de "Sous page"

4 Conclusion

L'objectif de ce chapitre a été de proposer une transformation des diagrammes d'activités avec les partitions (swimlanes) vers les réseaux de Petri dans un cadres IDM en respectant les spécifications définies par l'OMG. Notre approche s'est voulu la plus généraliste possible afin de transformer en réseaux de Pétri la majorité des diagrammes d'activités dont il reste quelque cas à traiter. On a présenté également la transformation modèle-texte qui est une transformation de modèle CPN généré dans l'étape précédente du système entier à un fichier XML qui est compréhensible par l'outil CPNtools.

Chapitre VI

Découpage du Modèle RDPHC

1 Introduction

Dans ce chapitre, nous présentons l'étape de slicing ou de découpage qu'on a déjà vue dans le chapitre IV qui permet de faire le découpage de modèle d'un système complexe, En préservant la validation des fichiers par rapport au schéma CPNtools. A l'objectif de développer et valider un nouveau modèle de vérification basé sur le découpage par bloc avec dépendances directes et indirectes.

2 Conception du découpage

L'idée de découpage de modèles proposée consiste à déterminer pour l'exigence R les partitions d'activité qui sont nécessaires à sa vérification dans une activité act, pour cela nous considérons une exigence R à vérifier dans une activité act qu'est partitionnée aux Partitions d'activité $part = \{B_1 \ldots\ldots B_n\}$ et R doit être satisfaite par un bloc $Bi \in Part$. Étant donné que l'exigence R a une relation «satisfy» avec Bi donc Bi est l'une des partitions d'activité nécessaires. Toutefois, dans certains cas, les actions attribuées à Bi ne sont pas suffisantes pour vérifier R, en raison de leurs liens avec d'autres actions allouées aux d'autres partitions d'activité. À savoir lorsque les actions attribuées au B_i nécessitent en leurs entrées, des actions affectées aux d'autres partitions d'activité. Pour cela, nous devons calculer l'ensemble des partitions d'activité que Bi a une dépendance directe ou indirecte. Soit $Calcule_dep(Bi\,;\,act)$ la fonction utilisée pour effectuer ce calcul, chaque partition d'activité $Bj \notin Calcule_dep(Bi\,;\,act)$ ne sont pas nécessaires à la vérification de l'exigence R. Le découpage est effectué sur le modèle de HCPN en supprimant, pour chaque $Bj \notin Calcule_dep(Bi\,;\,act)$, sa transition de substitution correspondante TB_J. Cela réduit l'espace d'états lors l'étape de la vérification du modèle. L'algorithme suivant est proposé pour calculer les dépendances directes et indirectes d'une partition Bi.

Algorithm 1 *Calcule_dep*(Bi;$DGBAP$) Entrée : (Bi : partition d'activité, DGBAP = <Part ;Ai>) Sortie : (DL :Liste des partitions d'activité)

1: $DL \leftarrow \{\}$
2: $pileP$; $empiler(P, Bi)$;
3: **while** $((P)$ *non vide*$)$ **do**
4: $Bj \leftarrow depiler(P)$;
5: $DL \leftarrow DL \cup \{Bj\}$;
6: **for all** $((Bk; Bj) \in A)$ **do**
7: **if** $(Bk \notin DL)$ **then** empiler(P, Bk) ;
8: **end if**
9: **end for**
10: **end while**

Pour calculer les dépendances directes et indirectes entre les partitions d'activité par la fonction $Calcule_dep(Bi;DGBAP)$, on a proposé de construire un graphe de dépendance entre les partitions d'activité, Où $DGBAP = < Part, A >$ est un graphe qui comporte les éléments de $Part$, l'ensemble des partitions de l'activité, comme les sommets et les éléments de $A \subset Part * Part$, comme indique $edges(Bi;Bj) \in A$, si la partition de l'activité Bj possède au moins un lien d'entrée à partir de la partition de l'activité Bi, donc le calcul $Calcule_dep(Bi;DGBAP)$consiste à effectuer la fonction $Calcule_dep(Bi;DGBAP)$comme indiqué dans l'algorithme 1.

3 Implémentation

3.1 Manipulation des fichiers XML

XML est un format de sortie de CPNtools, pour faire le découpage (slicing) de notre modèle, on doit manipuler le fichier XML. Il s'agit d'un parser qui lit le fichier xml et le rend disponible à un programme. Cependant, on a utilisé JDOM comme une API open source Java dont le but est de représenter et manipuler un document XML de manière intuitive pour un développeur Java sans requérir une connaissance pointue de XML. Avec cette API, nous récupérons le parser et chargeons le document avec le code qui suit :

```
1 SAXBuilder builder =new SAXBuilder("org.apache.xerces.parsers.SAXParser");
2 Document doc = builder.build("monDocument.xml");
```

Listing VI.1 – code JDOM pour charger le document

3.2 Suppression

À partir d'un fichier XML (le résultat du parser), on doit supprimer les parties du réseaux de Petri qui correspondent à la translation des parties du diagrammes d'activité qui ne sont pas nécessaire à la vérification de l'exigence que l'on souhaite vérifier(voir algorithme 1). Ce dernier est décrit par la figure VI.1 et la figure VI.2.

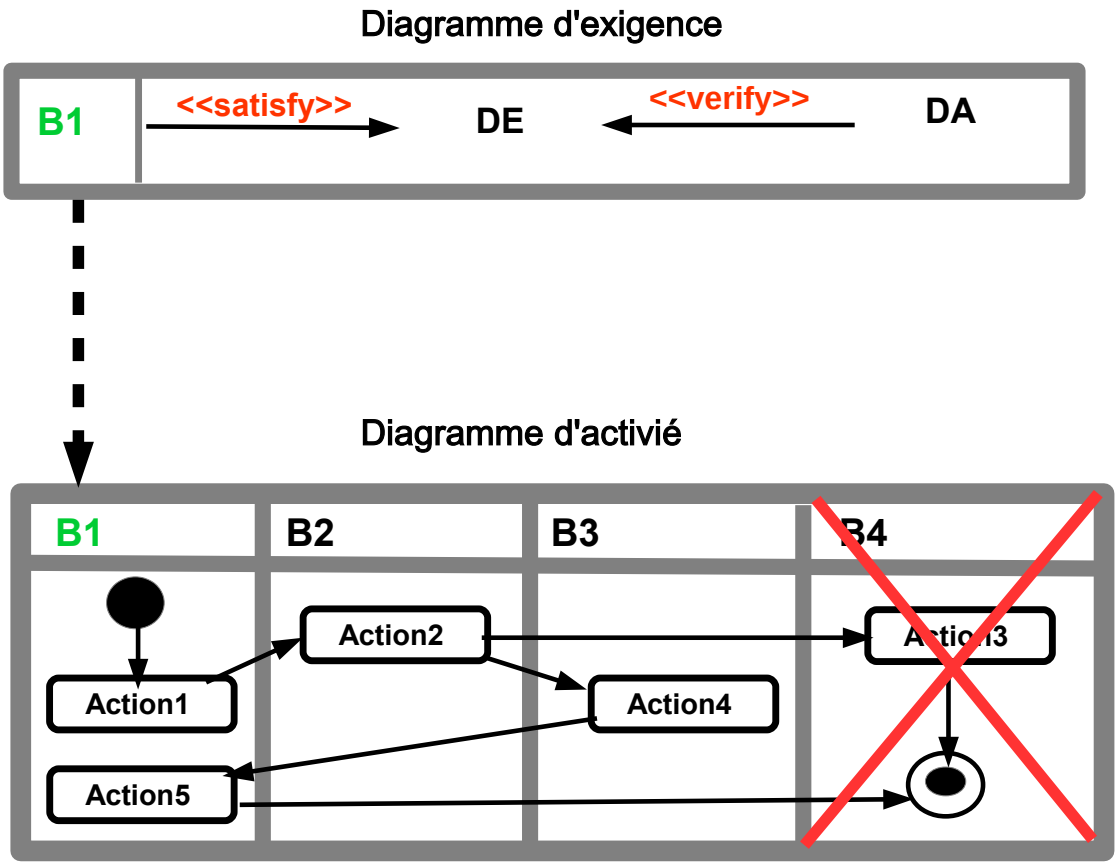


FIGURE VI.1 – Idée de découpage

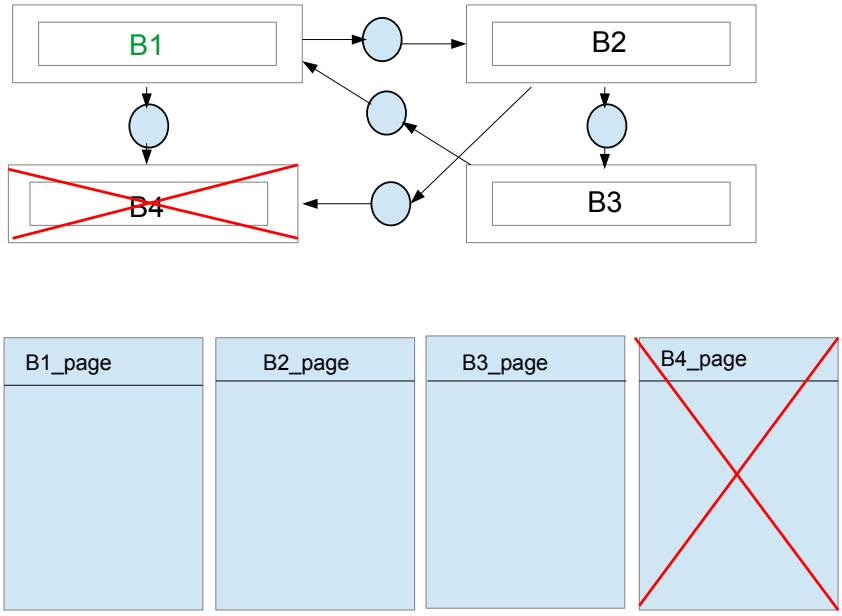


FIGURE VI.2 – Découpage représenté par RDP

Après l'opération de suppression des transitions de substitution qui ne sont pas nécessaires à la vérification, des problèmes concernant la validité du fichier RDP en format CPNtools ont été rencontrés comme la montre la figure VI.3.

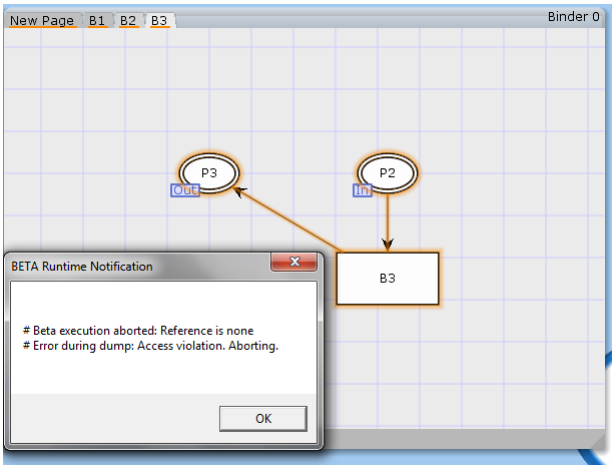


FIGURE VI.3 – Fichier invalide par CPNtools

Pour préserver la validation de fichier .xml après le découpage de modèle (Slicing) on doit supprimer :

— Les arcs entrants et sortants des transitions de substitution qui ont le même identificateurs idref avec l'une des transitions de substitution supprimées.

```
<arc id="ID1412328726" orientation="PtoT" order="1">
  <posattr x="0.000000" y="0.000000"/>
  <fillattr colour="White" pattern="" filled="false"/>
  <lineattr colour="Black" thick="1" type="Solid"/>
  <textattr colour="Black" bold="false"/>
  <arrowattr headsize="1.200000" currentcyckle="2"/>
  <transend idref="ID1412328720"/>
  <placeend idref="ID1412325775"/>
  </arc>
```

FIGURE VI.4 – Balise XML de l'arc

— Les places qui possèdent le même identificateur avec idref des arcs supprimés dans la page principale.

```
      <place id="ID1412325775">
        <posattr x="-168.000000" y="42.000000"/>
        <fillattr colour="White" pattern=""
filled="false"/>
        <lineattr colour="Black"
                  thick="1"
                  type="Solid"/>
        <textattr colour="Black"
                  bold="false"/>
        <text>P5</text>
        </marking>
       </place>
```

FIGURE VI.5 – Balise *XML* de Place

— Les subpages ou bien sous-pages liées aux transitions de substitution supprimées dont on peut extraire l'identificateur ID de subpage qu'on doit supprimer (figure : VI.6).

```
<page id="ID1412328730">
  <pageattr name="B4"/>
  <place id="ID1412328738">
    <posattr x="-168.000000"
             y="42.000000"/>
    <text>P5</text>
    </marking>

  </place>
```

FIGURE VI.6 – Balise XML de Sub-Page

— Les portsocks qui représentent la relation entre les transitions de substitution et les subpages, dont on doit arriver à supprimer le couple de portsock qui contient le même identificateur avec l'une des places supprimées avant VI.7.

```
<subst subpage="ID1412328034"
  portsock="(ID1412328042,ID1412325775)
  (ID1412328047,ID1412325571)
  (ID1412328052,ID1412323849)">
```

FIGURE VI.7 – Balise XML de Portsock

— Lors de la suppression des transitions de substitution on doit encore supprimer toutes les balises crées lors de la création des transitions de substitution concernées VI.8 VI.9 VI.10.

```
<instances>
  <instance id="ID2149"
            page="ID6">
    <instance id="ID1412328072"
              trans="ID1412328022"/>
    <instance id="ID1412328307"
              trans="ID1412328257"/>
    <instance id="ID1412328536"
              trans="ID1412328495"/>
    <instance id="ID1412328761"
              trans="ID1412328720"/>
  </instance>
</instances>
```

FIGURE VI.8 – Balise XML d'instance

```
<cpnsheet id="ID1412329831"
          panx="119.000000"
          pany="5.000000"
          zoom="1.000000"
          instance="ID1412328761">
  <zorder>
    <position value="0"/>
  </zorder>
</cpnsheet>
```

FIGURE VI.9 – Balise XML de cpnsheet

```
<zorder>
  <position value="2"/>
  <position value="3"/>
  <position value="4"/>
  <position value="1"/>
  <position value="0"/>
</zorder>
```

FIGURE VI.10 – Balise XML de position

Où on garde seulement les balises de position qui sont égales ou inférieur au nombre des transitions de substitution restées après le découpage(slicing).

4 Conclusion

Dans ce chapitre, nous avons présenté l'idée de découpage de modèle (slicing) et son implémentation. Ensuite, nous avons détaillé l'opération de suppression tout en pensant à la validité du fichier CPNtools résultant. A l'objectif de réduire le problème d'explosion combinatoire de l'espace d'état en considérant que les parties concernées lors de la vérification d'une propriété.

Chapitre VII

Réalisation et étude de cas

1 Introduction

Dans cette section nous allons illustrer notre méthodologie avec une étude de cas concernant un système de protection contre les incendies dans une salle informatique. Le système de protection contre les incendies se base sur des détecteurs de fumée ou des détecteur de chaleur qui détecte le feu et notifie les informations à la détection centrale pour confirmer une incendie. Si une incendie est confirmée, des alarmes internes et externes seront activées permet au système de suppression l'extinction du feu automatiquement.

2 Spécifications SysML

On a développé un outil qui nous a aidé de faire en premier temps la transformation de DA vers RDPHC. Puis, il permet de faire le découpage de modèle d'un système en préservant la validation des fichiers par rapport au schéma CPNtools, où il est présenté par une interface qui facilite ces fonctions. Pour cela on illustre notre méthodologie qui est présentée dans le chapitre IV par la création du diagramme d'activité avec des partitions pour le système de protection contre les incendies.

2.1 Diagramme d'activité

On utilise l'éditeur des diagrammes *PAPYRUS* intégré dans *POLARSYS* pour modéliser le diagramme d'activité qui présente le comportement de système de protection contre les incendies, la figure VII.1 illustre ce dernier.
Dans ce système, il nécessaire lorsque une incendie se produit, un détecteur détectera la chaleur ou la fumée, pour cela deux cas sont modélisés, le feu détectée et confirmée ou bien le feu n'est pas détecté. Dans le cas de détection de feu, la confirmation de l'incendie se déroulera au niveau de la détection centrale pour déterminer si une incendie est survenu, puis le système déclenche les alarmes internes et externes. En même temps, l'action extinction est activée. Et Pour le second cas, aucune action ne va être exécutée.

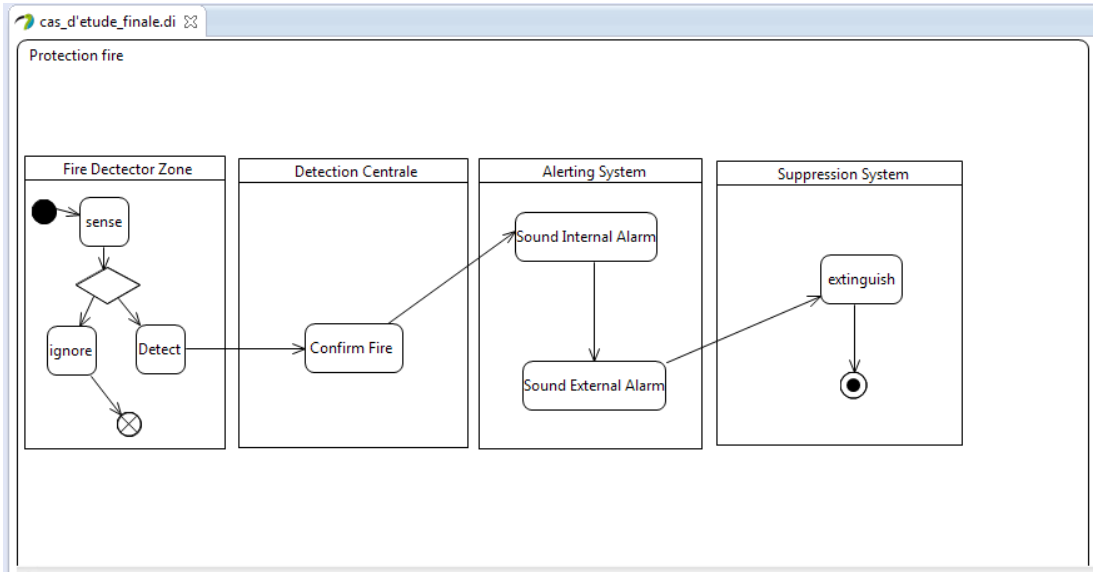


FIGURE VII.1 – L'activité principale *proteger_du_feu*

2.2 Diagramme d'exigence

Les exigences du système de protection contre les incendies sont modélisées dans un diagramme d'exigence SysML. Dans ce contexte, si on considère un bloc de SysML "Détection centrale" qui modélise la détection centrale dans le système de protection contre l'incendie, une exigence R : "si une incendie se produit, il sera Confirmé» et une activité "Protection Fire" qui modélise la détection et la protection contre les incendies ce qui signifie que le bloc "Détection centrale" satisfaire à l'exigence R et l'activité "Détecter Fire" vérifier l'exigence R. Comme montre la figure VII.2.

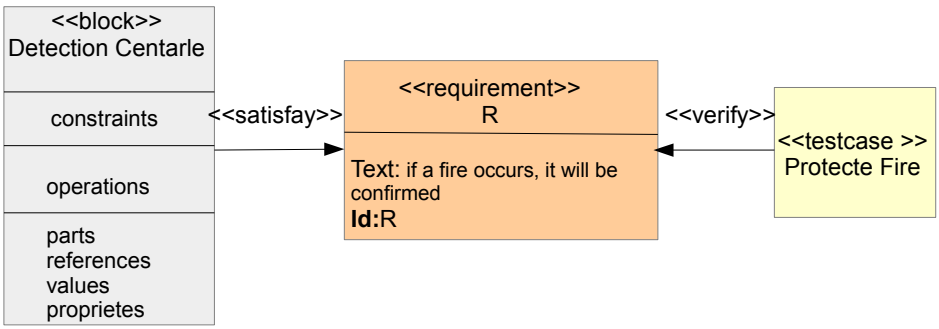


FIGURE VII.2 – Les reslations «satisfay» et «verify» dans le diagramme d'exigence

3 Transformation des spécifications SysML en RDPHC

Cette étape consiste a transformé le modèle précédents avec partitions en RDPHC.
On va présenter la transformation du diagramme d'activité avec partitions en RDPHC, pour cela on a crée une interface qui facilite la procédure de transformation. La figure VII.3 et VII.4 présente l'interface.
Premièrement, l'utilisateur doit créer un nouveau projet numéroté par **1**, qui regroupe toute les fichiers sortants au cours de l'utilisation de l'interface. Ensuite, il choisit le fichier du diagramme d'activité, son chemin est apparu dans le champ numéroté par **2**, ces derniers champs sont obligatoires pour passer à l'interface présenté par la figure VII.4.
Cela permet à l'utilisateur de faire la transformation de diagramme d'activité vers RDPHC numéroté par **3**, qui consiste à générer à partir d'un modèle source choisi précédemment le modèle PNML sous l'extension .xmi et le bouton présenté par numéro **4**, permet de visualiser le résultat de transformation et de l'exporter.
A partir du fichier PNML résultant depuis la première transformation, on commence une transformation de ce fichier en format CPN où on lance la transformation par le bouton **5**.
Le résultat est visualisé par l'exécution de bouton **6**, il s'agit d'un RdPCH en format CPNtools présenté dans la figure VII.6.



FIGURE VII.3 – les interfaces graphiques 1

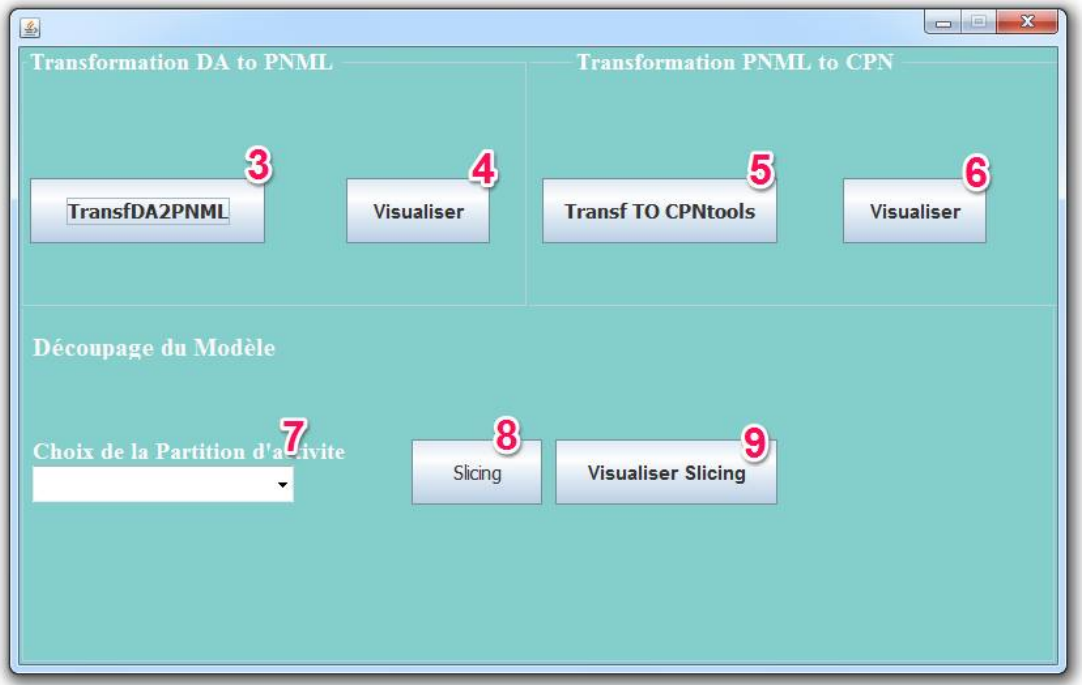


FIGURE VII.4 – les interfaces graphiques 2

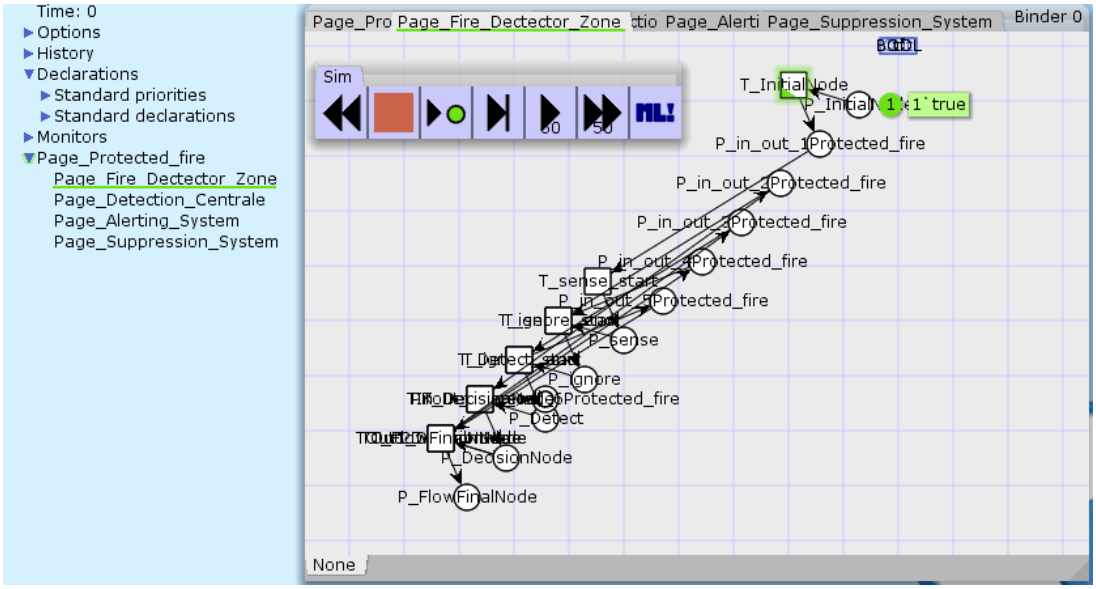


FIGURE VII.5 – le résultat de transformation 1

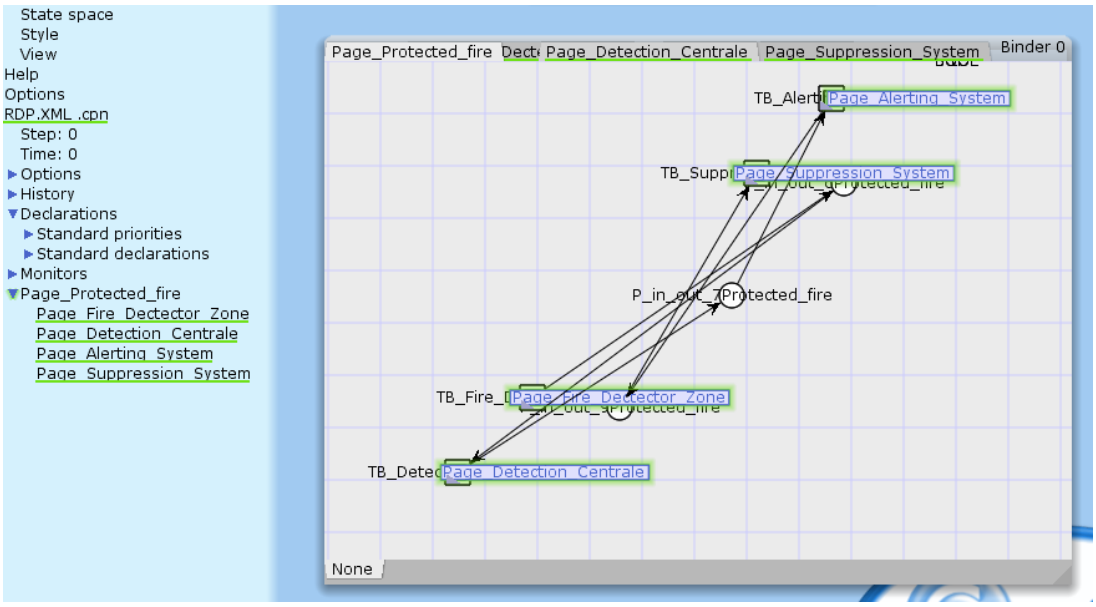


FIGURE VII.6 – le résultat de transformation 2

4 Découpage du modèle

Après l'étape de transformation, on peut commencer la technique de découpage (figure VII.4) dont l'utilisateur doit choisir la partition qui satisfait l'exigence R précédente à partir d'une liste de choix numéroté par **7**. Ensuite, on lance le découpage par le bouton **8** et on peut également visualiser le résultat de découpage qui est supporté par le schéma CPNtools en cliquant sur le bouton **9**. La figure VII.7 présente le résultat de découpage en choisissant la partition Détection centrale qui satisfait l'exigence R.

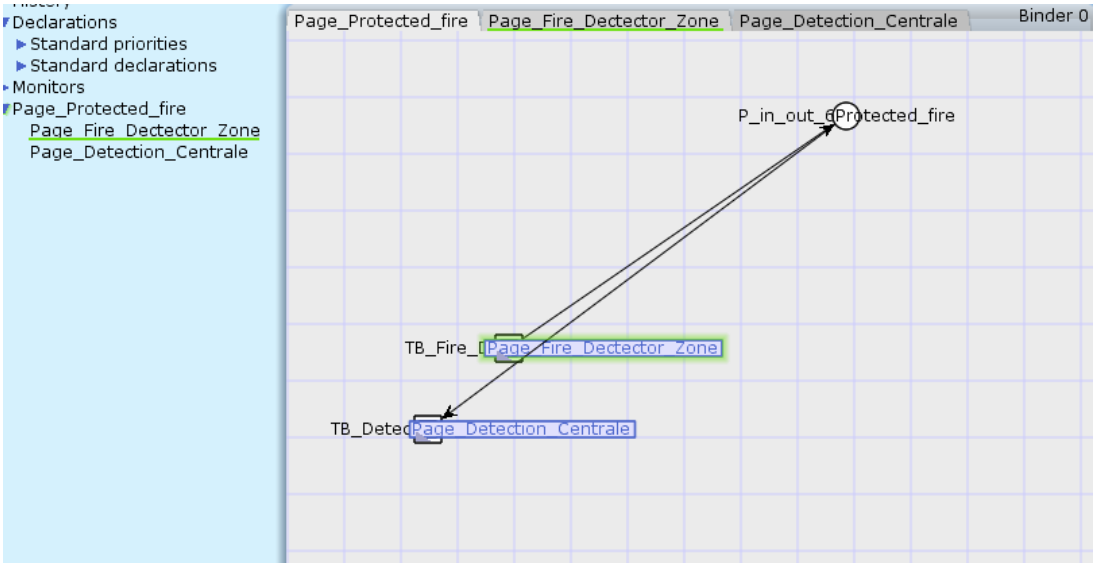


FIGURE VII.7 – Le résultat de découpage en choisissant la partition Détection centrale qui satisfait l'exigence R

TABLE 4.1 – Comparaison entre espace d'états avant et aprés le découpage

Marquage initiale N	Approche classique		Notre approche modulaire	
	Nbr noeuds	Nbr arcs	Nbr noeuds	Nbr arcs
1	21	20	14	13
2	231	420	105	182
3	1771	4620	560	1365
4	10626	35420	2380	7280
5	53130	212520	8568	30940
6	34210	139447	27132	111384
7	30431	123474	38268	167191

On remarque que l'espace d'état est réduit après le découpage comme le tableau 4.1 montre.

5 Conclusion

Dans ce chapitre, nous avons présenté une étude de cas où nous avons utilisé les spécifications SysML pour modéliser le modèle de digramme d'activité avec des partitions. Puis, nous avons appliqué automatiquement la transformation des diagrammes d'activité en RDPHC compatible avec le CPNtools. Enfin, nous sommes arrivées au découpage de modèle qui permet d'extraire seulement les partitions concernantes une exigence spécifiée. Pour faciliter ces procédures, nous avons développé une application qui automatise les étapes de l'approche proposée.

Conclusion générale

Les systèmes critiques peuvent avoir des pannes avec des conséquences très graves. Pour spécifier et modéliser ces systèmes, le langage SysML a été proposé comme un langage dédié à l'ingénierie des systèmes. Malgré les avantages de SysML, il reste un langage semi formel et nécessite une formalisation pour pouvoir effectuer des vérifications formelles sur les spécifications produites avec. Une des approches proposées pour résoudre ce problème consiste à co-utiliser SysML avec des méthodes formelles. Pour faire la vérification, on doit passer alors à partir des spécifications SysML aux des spécifications formelles.

Dans ce travail, nous avons proposé une démarche pour vérifier des diagrammes d'activité SysML via un passage automatique vers des réseaux de Pétri colorés et hiérarchiques(RDPHC). Le langage ATL a été utilisé pour implémenter cette transformation. Une transformation modèle-texte a été également effectuée pour adapter le modèle CPN généré dans la transformation précédente à l'outil CPNtools, l'outil à utiliser pour la vérification.

Cependant, comme la vérification basée sur les modèles souffre du problème de l'explosion combinatoire de l'espace d'états, nous avons proposé une démarche basée sur le découpage de modèles pour vérifier les diagrammes d'activité. Le découpage est appliqué sur les modèles transformés (Les RDPHC) en se basant sur les relations entre exigences, blocs et activités SysML. Le découpage consiste à ne garder que les parties nécessaires à la vérification d'une exigence donnée. Afin d'outiller l'approche proposée, nous avons développé un outil qui permet d'automatiser la transformation des diagrammes d'activité vers les RDPHC et supporter le découpage de ces derniers selon une exigence précise.

L'objectif de cette recherche était de proposer une démarche de vérification pour les diagrammes d'activité SysML tout en pensant au problème d'explosion combinatoire de l'espace d'états. Ceci a été développé en proposant une démarche de vérification basée sur le découpage (slicing) de modèles. Comme perspective à ce travail, il est important de poser la question suivante : est-il possible de faire le découpage au niveau des actions au lieu des partitions d'activité ? Il est aussi important d'appliquer la démarche proposée sur des cas réels et d'évaluer son efficacité. La réponse à ces questions reste en attente des orientations que pourraient prendre les recherches futures.

Bibliographie

[1] Dominique LUZEAUX, Jean-Rene RUAULT, and Jean-Luc WIPPLER. Introduction a l'ingénierie systéme les concepts généraux.

[2] *o*mg. – omg unified modeling language tm(omg uml). version 2.5.sept. 2013 http ://www.omg.org/spec/uml/2.5/beta2/pdf.

[3] Omg.–omg systems modeling language(omg sysml^{TM}) version 1.3.(2012) http ://www.omg.org/spec/sysml/1.3/pdf.

[4] Agusti CANALS. Référence :h3885 sysml : une notation pour spécifier et concevoir des systèmes.

[5] BRUEL(J.M.). http ://jmbhome.herokuapp.com/sysml.overview-1.3.fr.pdf carte de référence sysml 1.3 en français.

[6] Denis-Papin. Respectivement professeur agrégé de génie mécanique au lycée chaptal de paris (75008) et professeur de construction mécanique au lycée denis-papin de la courneuve (93)technologie avril 2012.

[7] Pascal Roques. Modélisation de systémes complexe savec sysml. groupe eyrolles, isbn :978-2-212-13641-8 ;. 2015.

[8] Object Management Group. *s*ystems modeling language (sysml) specification. sysml partners (www.sysml.org). 2012.

[9] J.P. Chemla. *A-Processus déterministes :Les Réseaux de Petri, Modélisation et analyse des systèmes de production Polytech'Tours département Productique 2ème année.*

[10] Pierre LADET. *r*éseaux de petricet article est issu de : Archives.

[11] Univ Nantes. La vérification de propriétés.

[12] Robert Valette. "les réseaux de petri" support de cours,laas-cnrs-toulouse -checking. 2000.

[13] Salim Ben Saud. Les réseaux de petri, avril. 2015.

[14] Houda HAIOUNI. *Mémoire Magister en informatique, Approche mixte de modélisation par Réseaux de Petri et SMA, Ecole Doctorale en Informatique de l'Est Pôle de CONSTANTINE, Option : Intelligence Artificielle ,novmebre 2010.*

[15] Kurt Jensen. *coloured Petri nets Basic concepts, Analysis methods and practical use", volume1, second Edition, springer, 1992.* ,.

[16] Rachida Hadibyghoul. *les réseaux de petri : outil de modélisation et de conduite des systémes de production outomatisé.*

[17] P. Hubert., K.Jensen., and R.M. Shapiro. *Hierarchies in coloured Petri Nets", Advances in Petri Nets, Lecture Notes in Computer Science, vol 483, Springer Verlag, Berlin Heidelberg, New York, 1990.* ,.

[18] Kurt Jensen . and Lars Michael. *h*coloured petri nets - modelling and validation of concurrent systems. springer, 2009.

[19] Robin Milner. "mads tofte, and robert harper. definition of standard ml.mit press. 1990.

[20] Jeffery-D Ullman . and Lars Michael. *Elements of ML programming", Pearson Education, 2nd edition, 1998. (ISBN : 9780137903870).*,.

[21] Raida ElMansouri. Modélisation et vérification des processus métiers dans les entreprises virtuelles : Une approche basée sur la transformation de graphes, université mentouri constantine.

[22] E.M Clarke . and O. Grumberg.and D. Peled. *Model Checking. 2000.*,.

[23] Lukasz FRONC. *c*ompilation de réseaux de petri modeles haut niveau et symétries de processus. thése de doctorat de l'université d'evry-val d'essonne, spécialité informatique soutenue le 28 novembre 2013.

[24] Davide Falessi1.and Shiva Nejati1.and Mehrdad Sabetzadeh1., Lionel Briand1 ., and Antonio Messina2. *s*afeslice : A model slicing and design safety inspection tool for sysml 1simula research laboratory oslo, norway 2 university of rome (tor vergata)rome, italy.

[25] Jean Bézivin. and Olivier Gerbé. *Towards a Precise Definition of the OMG/MDA Framework. In Proceedings of the 16th IEEE international conference on Automated Software* ,.

[26] *d*amien foures. transformation des diagrammes d'activites sysml1.2 vers les reseaux de petri dans un cadre mde. rapport laas n 11864. rapport de stage. 2012, 55p. <hal-00761053> hal id : hal-00761053,https ://hal.archives-ouvertes.fr/hal-00761053 ,submitted on 4 dec 2012.

[27] xavier blanc. *mda en action ingénierie guidée par les modéles.*

[28] *m*eta object facility (mof) 2.0 core specification,"version 2. 2006.

[29] Ivan Kurtev. and Jean Bézivin.and Mehmet Aksit. *i*van kurtev, jean bézivin, and mehmet aksit. technological spaces : An initial appraisal. in coopis, doa'2002 federated conferences, industrial track, irvine. 2002.

[30] Jean Bézivin., Frédéric Jouault., Peter Rosenthal., and Patrick Valduriez. *m*odeling in the large and modeling in the small. in u. aßmann, m. aksit, and a. rensink, editors, model driven architecture, european mda workshops : Foundations and applications, mdafa 2003 and mdafa 2004, , june 26-27, 2003 and linköping, sweden, june 10-11, 2004, revised selected papers, volume 3599 of lecture notes in computer science, pages 33–46, twente, the netherlands, 2005. springer.

[31] Jean Bezivin . and Ivan Kurtev. *m*odel-based technology integration with the technical space concept. in metainformatics symposium, esbjerg, denmark, 2005. springer-verlag.

[32] Jean Bézivin. and Xavier Blanc. *p*romesses et interrogations de l'approche mda. journal développeur référence, septembre 2002.

[33] Samba Diaw., Rédouane Lbath., Rédouane Lbath., and Bernard Coulette. *e*tat de l'art sur le développement logiciel dirigé par les modèles date de cette version :12 janvier 2009.

[34] Xavier Blanc. *m*da en action. eyrolles, march 2005.

Annexe 1

Le code source ATL des règles de transformation DA en RdP proposé

typecntflow()

```
helper context MMuml!ControlFlow def: typecntflow():Boolean =
  if self.inPartition->isEmpty() then
    true
  else
    false
  endif;
```

Listing 1 – Le code *ATL* de *helper typecntflow*()

Helper arc (incoming)

```
helper context MMuml!OpaqueAction  def:element() :Boolean=
 if  self.incoming-> select(e | e.inPartition->isEmpty())->notEmpty()
  then true
   else
   false
  endif;
```

Listing 2 – Le code *ATL* de *helper Helperdearcincoming*

Helper arc (outgoing)

```
helper context MMuml!OpaqueAction  def:element1() :Boolean=
 if  self.outgoing-> select(e | e.inPartition->isEmpty())->notEmpty()
  then true
   else
   false
  endif;
```

Listing 3 – Le code *ATL* de *helper Helperarcoutgoing*

PetriNet

```
rule PetriNet {
  from
    model: MMuml!Model
  to
    pn: MMpn!PetriNet (
      id <- model.name,
      pages <- model.packagedElement -> collect(e | thisModule.resolveTemp(
    e,'page'))

    )
}
```

Listing 4 – Le code *ATL* de la règle "*PetriNet*"

ActivityPartition

```
rule ActivityPartition {
 from
 ActPart: MMuml!ActivityPartition

 to
    substrns:MMpn!Transition (
       id <- 'TB_' + ActPart.name,
     Type <- 'SubsTransition',
       RefPage <- thisModule.resolveTemp(ActPart, 'page'),
     containerPage <- thisModule.resolveTemp(ActPart.inActivity, 'page'),
     InArcs <-  thisModule.resolveTemp(ActPart.containedEdge, 'outarc2'),
       OutArcs <- thisModule.resolveTemp(ActPart.containedEdge, 'inarc2')
      ),
    page: MMpn!Page (
     id <- 'Page_' + ActPart.name,
     containerPage <- thisModule.resolveTemp(ActPart.inActivity, 'page')
   )

}
```

Listing 5 – Le code *ATL* de la règle "*ActivityPartition*"

Controlflow dans la partition

```
rule ctlflw {
 from
   ctl: MMuml!ControlFlow(not ctl.typecntflow())
 to
   inarc: MMpn!Arc (
     id <- 'A_' + ctl.name+ctl.activity.name + '_in',
           containerPage <- thisModule.resolveTemp(ctl.inPartition->first
  ()  , 'page'),
        source <- Sequence{thisModule.resolveTemp(ctl.source, 'trst'),
              thisModule.resolveTemp(ctl.source, 'trst1'),
              thisModule.resolveTemp(ctl.source, 'EndTrst'),
                thisModule.resolveTemp(ctl.source, 'SubsTrns')
            },
     target <- ArcPlc
   ),
   ArcPlc: MMpn!Place (
     id <- 'P_in_out_' + ctl.name+ctl.activity.name,
     containerPage <- thisModule.resolveTemp(ctl.inPartition->first(), '
  page'),
     referencingPlaces <- Sequence{thisModule.resolveTemp(ctl.target, '
  refPlacein'),
           thisModule.resolveTemp(ctl.source, 'refPlaceout')
           }
   ),
   outarc: MMpn!Arc (
```

```
23        id <- 'A_' + ctl.name+ctl.activity.name + '_out',
24        containerPage <- thisModule.resolveTemp(ctl.inPartition->first(), '
    page'),
25        source <- ArcPlc,
26        target <- Sequence{thisModule.resolveTemp(ctl.target, 'ActStart'),
27              thisModule.resolveTemp(ctl.target, 'start'),
28              thisModule.resolveTemp(ctl.target, 'substrns')
29              }
30     )
31 }
```

Listing 6 – La règle "*Controlflowdanslaparttion*"

Controlflow hors la partition

```
1 rule ctlflw2 {
2   from
3     ctl: MMuml!ControlFlow(ctl.typecntflow())
4   to
5     inarc2: MMpn!Arc (
6       id <- 'A_' + ctl.name+ctl.activity.name + '_in',
7           containerPage <- thisModule.resolveTemp(ctl.activity, 'page'),
8             source <-thisModule.resolveTemp(ctl.source.inPartition->first()
    , 'substrns'),
9       target <- ArcPlc2
10    ),
11    ArcPlc2: MMpn!Place (
12      id <- 'P_in_out_' + ctl.name+ctl.activity.name,
13      containerPage <- thisModule.resolveTemp(ctl.activity, 'page'),
14      referencingPlaces <- Sequence{thisModule.resolveTemp(ctl.target, '
    refPlacein'),
15                                thisModule.resolveTemp(ctl.source, '
    refPlaceout')
16              }
17    ),
18    outarc2: MMpn!Arc (
19      id <- 'A_' + ctl.name+ctl.activity.name + '_out',
20      containerPage <- thisModule.resolveTemp(ctl.activity, 'page'),
21      source <- ArcPlc2,
22      target <- thisModule.resolveTemp(ctl.target.inPartition->first(), '
    substrns')
23    )
24 }
```

Listing 7 – La règle "*Controlflowhorslaparttion*"

Dans le cas d'un arc entrant(incoaming)

```
1 rule Action2 {
2   from
3     actn: MMuml!OpaqueAction ((actn.element()) and (not  actn.element1()))
4   to
```

```
5           refPlacein: MMpn!RefPlace (
6       id <- 'PRin_' + refPlacein.ref.id,
7       containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page')
8     ),
9     arc_end1: MMpn!Arc (
10      id <- 'A_' + actn.name + '_arc_ref',
11      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
12      source <- refPlacein,
13      target <- ActStart
14    ),
15
16    ActStart: MMpn!Transition (
17      id <- 'T_' + actn.name + '_start',
18      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
19      InArcs <- thisModule.resolveTemp(actn.incoming, 'outarc')
20    ),
21    arc_start: MMpn!Arc (
22      id <- 'A_' + actn.name + '_arc_start',
23      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
24      source <- ActStart,
25      target <- plc_on
26    ),
27    plc_on: MMpn!Place (
28      id <- 'P_' + actn.name,
29      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page')
30    ),
31    arc_end: MMpn!Arc (
32      id <- 'A_' + actn.name + '_arc_end',
33      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
34      source <- plc_on,
35      target <- EndTrst
36    ),
37    EndTrst: MMpn!Transition (
38      id <- 'T_' + actn.name + '_end',
39      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
40      OutArcs <- thisModule.resolveTemp(actn.outgoing, 'inarc')
41    )
42
43 }
```

Listing 8 – Dans le cas d'un arc entrant(incoaming de l'action)

Dans le cas d'un arc sortant (outgoing)

```
1 rule Action3 {
2   from
3     actn: MMuml!OpaqueAction ( (not actn.element() ) and (actn.element1()))
4   to
5
6     ActStart: MMpn!Transition (
7       id <- 'T_' + actn.name + '_start',
8       containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
9       InArcs <- thisModule.resolveTemp(actn.incoming, 'outarc')
10    ),
11    arc_start: MMpn!Arc (
12      id <- 'A_' + actn.name + '_arc_start',
13      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
14      source <- ActStart,
15      target <- plc_on
16    ),
17    plc_on: MMpn!Place (
18      id <- 'P_' + actn.name,
19      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page')
20    ),
21    arc_end: MMpn!Arc (
22      id <- 'A_' + actn.name + '_arc_end',
23      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
24      source <- plc_on,
25      target <- EndTrst
26    ),
27    EndTrst: MMpn!Transition (
28      id <- 'T_' + actn.name + '_end',
29      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
30      OutArcs <- arc1
31    ),
32
33    arc1: MMpn!Arc (
34      id <- 'A_' + actn.name+'_ref' ,
35      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
36      source <- EndTrst,
37      target <-  refPlaceout
38    ),
39      refPlaceout: MMpn!RefPlace (
40      id <- 'PRout_' + refPlaceout.ref.id,
41      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
```

```
    page')
42   )
43 }
```

Listing 9 – Dans le cas d'un arc sortant(outgoing) de l'action

Dans le cas d'un arc entrant et autre sortant d'un même nœud

```
1 rule Action4 {
2   from
3     actn: MMuml!OpaqueAction (actn.element()and actn.element1())
4   to
5   refPlacein: MMpn!RefPlace (
6       id <- 'PRin_' + refPlacein.ref.id,
7       containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page')
8     ),
9     arc1: MMpn!Arc (
10      id <- 'A_' + actn.name ,
11      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
12      source <- refPlacein,
13      target <- ActStart
14    ),
15
16  ActStart: MMpn!Transition (
17      id <- 'T_' + actn.name + '_start',
18      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
19      InArcs <- thisModule.resolveTemp(actn.incoming, 'outarc')
20    ),
21    arc_start: MMpn!Arc (
22      id <- 'A_' + actn.name + '_arc_start',
23      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
24      source <- ActStart,
25      target <- plc_on
26    ),
27    plc_on: MMpn!Place (
28      id <- 'P_' + actn.name,
29      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page')
30    ),
31    arc_end: MMpn!Arc (
32      id <- 'A_' + actn.name + '_arc_end',
33      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
34      source <- plc_on,
35      target <- EndTrst
36    ),
```

```
37      EndTrst: MMpn!Transition (
38        id <- 'T_' + actn.name + '_end',
39        containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
40        OutArcs <- thisModule.resolveTemp(actn.outgoing, 'inarc')
41      ),
42      arc_end1: MMpn!Arc (
43        id <- 'A_' + actn.name + '_arc_ref',
44        containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
45        source <- EndTrst,
46        target <- refPlaceout
47      ),
48    refPlaceout: MMpn!RefPlace (
49        id <- 'PRout_' + refPlaceout.ref.id,
50        containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page')
51      )
52
53
54  }
```

Listing 10 – Dans le cas d'un arc entrant et autre sortant d'un même nœud

Les autres règles

```
1  rule Action {
2    from
3      actn: MMuml!OpaqueAction ((not actn.element()) and (not actn.element1()
     ) )
4    to
5      ActStart: MMpn!Transition (
6        id <- 'T_' + actn.name + '_start',
7        containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
8        InArcs <- thisModule.resolveTemp(actn.incoming, 'outarc')
9      ),
10     arc_start: MMpn!Arc (
11       id <- 'A_' + actn.name + '_arc_start',
12       containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
13       source <- ActStart,
14       target <- plc_on
15     ),
16     plc_on: MMpn!Place (
17       id <- 'P_' + actn.name,
18       containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page')
19     ),
20     arc_end: MMpn!Arc (
```

```
21      id <- 'A_' + actn.name + '_arc_end',
22      containerPage <- thisModule.resolveTemp(actn.inPartition->first(), '
    page'),
23      source <- plc_on,
24      target <- EndTrst
25    ),
26    EndTrst: MMpn!Transition (
```

Listing 11 – Les autres règle