الجمهورية الجزائرية الديمقراطية الشعبية

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DR. YAHIA FARES DE MEDEA

Faculté des Sciences

Département Mathématiques et Informatique



Projet de Fin D'étude

Présenté pour l'obtention du Diplôme De Master en Informatique

Option : Ingénierie des Systèmes et Technologie du Web

THEME

Vérification formelle des Diagrammes d'Activité SysML

Préparé et présenté par :

M^{elle} : **LAMRI Abir**

M^{elle} : **TOUBAL SEGHIR Wafia Hassina**

Proposé et dirigé par :

Mr.**KHELDOUN Ahmed**

Co-encadré par :

Mr.**RAHIM Massaoud**

Année Universitaire 2016-2017

Remerciements

Avant tout, nous remercions le Dieu tout puissant de nous avoir donné la force et la volonté pour achever ce modeste travail. Nous tenons à remercier particulièrement nos parents pour leur contribution, leur soutien et leur patience.

Nous remercions infiniment nos Parents pour tout ce qu'ils ont fait et ce qu'ils font encore pour nous.

Nous nous devons remercier le corps professoral et administratif de la Faculté des Sciences, pour la richesse et la qualité de leur enseignement et qui déploient de grands efforts pour assurer à leurs étudiants une formation actualisée.

Nous nous adressons tous nos gratitudes et nos remerciements à notre promoteur Mr. Kheldoun Ahmed qui s'est toujours montré à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'il a bien voulu nous consacrer.

Nous tenons aussi à remercier sincèrement Mr. Rahim Messaoud en tant que co- promoteur,qui nous a prodigué de ces conseils, et encore, la patience, la confiance qu'il nous a témoignés ont été déterminants dans la réalisation de notre travail

Nous tenons également à remercier l'ensemble du corps enseignant du département de mathématiques et informatique (UYF-Médéa), pour leurs concours dans la réussite de notre formation.

Nous remercions les membres du jury d'avoir accepter de juger notre travail.
Nous remercions également les personnes nous ont aider

Merci à tous les membres de notre famille et à nos amis et collègues pour leur soutien durant toutes les années d'étude.
En fin, nous tenons à remercier tous ceux qui ont contribué d'une façon ou d'une autre à la réalisation de ce travail.

Dédicace

A la lumière de ma vie, ma très chère mère

Pour son amour, ses prières et ses sacrifices, pour m'avoir donnée la vie et la joie de vivre. Ta bonne éducation, tes conseils et tes bénédictions n'ont jamais fait défaut, que Allah le tout puissant te garde pour moi (amin).

A ma très chère grande mère.

A mon oncle Kamel auquel je tiens la force, la ténacité

A mon frère Anes et sa femme Wissem, mon frère Amine et sa femme Nora, et mon frère Abdenour et mon fiancé abdelkader.

A mon neveu Amir et ma nièce Rafif

Et toute ma famille qui n'a pas cessé de m'encourager dans les moments difficiles que

Allah les protège.

A Latifa, Mounira, Nor, karima, Soumia, Asma, Afaf, et Marwa.

A mon binôme et sœur Abir, pour son amitié sa loyauté et sa franchise, et toute sa famille.

A mes collègues

A touts mes amis.

A toute la promotion 2016-2017.

Wafia

Dédicace



Je dédie ce modeste travail à mes très chers parents qui

veille pour me voir réussir, et qui n'ont jamais cessé de m'encourager,

de me conseiller.

A mes chers frères et et soeurs.

A mon oncle Dr Ali Mohad Abdenor auquel je tiens la force, la ténacité

A ma très cher amie et binôme et sœur Wafia

A toute les personnes qui, de près ou de loin , ont contribué à la réussite

de ce Travail

A mes collègues

A toute la promotion 2016-2017.

Abir

Liste des acronymes

ADT **A**TL **D**evelopement **T**ool

ATL **A**tlas **T**ransformation **L**angauge

API **A**pplication **P**rogramming **I**nterface

BDD **B**lock **D**efinition **D**iagram

CIM **C**omputation **I**ndependant **M**odel

CT **C**reated **T**okens

DA **D**iagramme d'**A**ctivité

DFD **D**ata **F**low **D**iagram

DSIG **D**omains **S**pecial **I**nterest **G**roup

DT **D**estroyed **T**okens

EMF **E**clipse **M**odeling **F**ramework

EMOF **E**ssential **M**eta **O**bject **F**acility

FFBD **F**unctional **F**low **B**lock **D**iagram

GMF **G**raphical **M**odeling **F**ramework

IBD **I**nternal **B**lock **D**iagram

IC **I**nput **C**ondition

ICT **I**ndexed **C**reated **T**okens

IDM **I**ngénierie **D**irigée par le **M**odel

INCOSE **I**nternational **C**ouncil **O**n **S**ystem **E**ngineering

IS **I**ngénierie **S**ystème

KM3 **K**ernel **M**eta **M**eta **M**odel

LTL **L**inear **T**emporal **L**ogic

MDA **M**odel **D**riven **A**rchitecture

MDE **M**odel **D**riven **E**ngineering

MOF **M**eta **O**bject **F**acility

OCL **O**bject **C**onstraint **L**anguage

OMG **O**bject **M**anagement **G**roup

PDM **P**latform **D**escription **M**odel

PIM **P**latform **I**ndependent **M**odel

PSM **P**latform **S**pecific **M**odel

QVT **Q**uery **V**iew **T**ransformation

RFP **R**equest **F**or **P**roposal

RDP **R**eseau **D**e **P**etri

RDPR **R**eseau **D**e **P**etri **R**écursifs

SE **S**ystems **E**ngineering

SysML **S**ystems **M**odeling **L**anguage

TC **T**ransition **C**ondition

UML **U**nifed **M**odeling **L**anguage

XMI **X**ML **M**etadata **I**nterchange

XML e**X**tensible **M**arkup **L**anguage

ملخص

نظام لغة النمذجة (SysML) هي لغة الرسم البياني تستخدم على نطاق واسع لتحديد الأنظمة المعقدة. و مع ذلك، هذه اللغة هي شبه رسمية ومعانيها غير معرفة بشكل رسمي. وبالتالي، فإنه ليس من الممكن أن تحقق المواصفات الشاملة لنظام لغة النمذجة(SysML).

و الهدف من هذا المشروع هو اقتراح نهج التحقق الرسمي من بعض النظرة الحيوية لنظام لغة النمذجة (SysML) , أي مخططات الانظمة.

لتحقيق هذا الهدف , مررنا على نظام لغة النمذجة (SysML) لخصوصيات رسمية مبنية على تمديد شبكات بتري و يسمى RECATNet . هذا التمديد جد معروف لقدرته التعبيرية و دلالاته , حيث يمكن تعريفه بسهولة من ناحية منطق اعادة الكتابة , هذه الاخيرة تسمح باستعمال نظام مود (Maude) اثناء مراحل التحليل و التحقيق.

لتنفيذ هذا المشروع، قمنا بتطوير أداة جافا للتحقق من الخصائص الوظيفية، الموصوفة من قبل المخططات المتطلبات، على مخططات الأنظمةSysML.

الكلمات المفتاحية :

التحقق الرسمي ، لغة نمذجة الأنظمة، مخطط الأنظمة، مخطط المتطلبات، شبكات البتري، نظام المود، جافا.

Résumé

SysML (System Modeling Language) est un langage graphique largement répandu pour la spécification des systèmes complexes. Cependant, ce langage est semi-formel et sa sémantique n'est pas formellement définie. Par conséquent, il n'est pas possible de vérifier de manière exhaustive les spécifications du SysML.

L'objectif de ce projet est de proposer une approche de vérification formelle d'une partie de la vue dynamique du langage SysML, à savoir, les diagrammes d'activités (SysML-AD).

Afin de réaliser cet objectif, nous avons passé du langage SysML à des spécifications formelles en se basant sur une extension des réseaux de Petri, appelée RECATNet. Cette extension est très connue par son pouvoir d'expression, sa sémantique, et le fait qu'elle peut être définie facilement en termes de la logique de réécriture. Cette dernière permet d'utiliser le système Maude lors des phases d'analyse et de vérification.

Pour implémenter ce projet, nous avons développé un outil JAVA permettant de vérifier les propriétés fonctionnelles, décrites par les diagrammes d'exigences, sur des diagrammes d'activités SysML.

Mots clés:

Vérification formelle, SysML, Diagramme d'activité, Diagramme d'exigence , IDM, Réseaux de Petri, RECATNet, Maude, JAVA.

Abstract

SysML (System Modeling Language) is a graphical language, widely used is widely used graphical language for the specification of complex systems. However, this language is semi-formal and its semantics is not formally defined. Therefore, it is not possible to fully verify the SysML specifications.

The objective of this project is to propose a formal verification approach for one dynamic view of the SysML language, namely, the activity diagrams (SysML-AD).

In order to achieve this objective, we have gone from SysML to formal specifications based on an extension of the Petri nets, called RECATNet. This extension is well known by its power of expression, its semantics, and the fact that it can be defined easily in terms of the rewriting logic. This allows the Maude system to be used during the analysis and verification phases.

To implement this project, we developed a JAVA tool to check the functional properties, described by the requirements diagrams, on SysML activity diagrams.

Keywords :

Formal verification, SysML, Activity Diagram, Requirement Diagram, IDM, Petri nets, RECATNet, Maude, JAVA.

Table des matières

Résumé…… iii

Introduction générale …… … 01

I **SysML**…… 04

1. Introduction…… 05
2. Ingénierie Système …… 05
3. Historique …… 06
4. Présentation SysML…… 07

4.1. Diagrammes SysML…… 08

4.1.1. Modélisation structurelle…… 08

4.1.1.1. Diagramme de définition de blocs…… 09

4.1.1.2. Diagramme de bloc interne…… 09

4.1.1.3. Diagramme paramétrique…… 10

4.1.1.4. Diagramme de paquetage…… 11

4.1.2. Modélisation dynamique …… 12

4.1.2.1 Diagramme de séquences…… 12

4.1.2.2. Diagramme de cas d'utilisations…… 13

4.1.2.3. Diagramme d'état…… 14

4.1.2.4. Diagramme d'activité …… 15

4.1.3. Modélisation fonctionnelle…… 18

4.1.3.1. Diagramme d'exigences…… 18

5. OutilsSysML…… 19
6. Conclusion…… 19

II **L'ingénierie dirigée par les modèles** …… 21

1. Introduction…… 22
2. Présentation de l'IDM…… 22
3. Les principes généraux de l'IDM…… 23

3.1. La Modélisation…… 23

3.2. La Méta-Modélisation…… 23

4. L'approche MDA (Model Driven Architecture)…… 25
4.1. Transformation des modèles du MDA…… 27
5. La transformation de modèles…….. 29
5.1. Type de Transformation…… 31
6. Langage de transformation ATL…… 32
6.1. La plate-forme de modélisation EMF…… 33
6.2. Atlas Transformation Langage…….. 33
6.3. Programmation d'un module ATL…… 34
6.3.1. La section d'en-tête…… 34
6.3.2. La section d'importation…… 34
6.3.3. Les helpers……. 35
6.3.4. Les règles…… 35
6.3.5 Les étapes de la transformation……. 35
7. Conclusion……. 35

III Formalisme de RECATNets …… 36
1. Introduction …….. 37
2. Réseau de Petri…… 38
2.1. Concept et définitions…….. 38
2.1.1. Définition formelle…… …… 39
2.1.2. Marquage…… 39
2.1.3. Représentation matricielle…….. 40
2.2. Dynamique des RdP…….. 41
2.2.1. Franchissement d'une transition……. 41
2.2.2. Séquence de franchissements…….. 42
2.2.3. Marquages accessibles…….. 42
2.2.4. Graphe de marquages……. 43
2.3. Propriétés des réseaux de petri…… 43
2.3.1. Caractère borné…… 43
2.3.1.1. Place k-bornée…… 43
2.3.1.2. Réseau borné…….. 44
2.3.2. Vivacité……. 44
2.3.2.1. Transition quasi-vivante……. 45

2.3.2.2. Transition vivante…….. 45
2.3.2.3. Réseau de Petri vivant…….. 45
2.3.3. Blocage……. 45
3. Réseaux de Petri Récursifs……. 46
3.1. Définition…….. 47
3.2. Marquage étendu…….. 48
3.3. Franchissabilité et franchissement…….. 49
3.3.1. Définition…… 49
3.3.2. Franchissement d'une Transition élémentaire…… 49
3.3.3. Franchissement d'une Transition abstraite…… 50
3.3.4. Franchissement d'une étape de coupure…… 50
4. Les RECATNets……. 51
4.1. Syntaxe des RECATNets…….. 52
4.2. Définition formelle…… 53
4.3. Marquage étendu…… 54
4.4. Exemple illustratif…….. 55
5. Méta-modèle RECATNet…… 57
5.1. Composition de Méta-Modèle ……. 57
6. Sémantique de RECATNet en Maude…… 60
6.1 Maude et logique de réécriture……... 61
7. Conclusion…….. 61

IV Translation des diagrammes d'activités SysML en RECATNet…… 62
1. Introduction…… 63
2. Processus de transformation …….. 63
2.1. Syntaxe abstraite du diagramme d'activité SysML…… 64
2.2. Les règles de transformation…….. 65
3. Exemple illustrarif……. 75
4. Conclusion ..…… 77

V Implémentation et Réalisation…….. 78
1. Introduction…… 79
2. Processus de développement ……. 79
3. Etude de cas ……..…… 80
3.1. Description de cas …….. 81
3.1.1. Diagramme d'activité ……..... 82
3.1.2. Diagramme d'exigence ……..... 83
4. Transformation des spécifications SysML en langages Formelles …… 85
4.1. Transformation de diagramme d'activité en RECATNet ….……. 85
4.2. Transformation de diagramme d'exigence en LTL ….…… 91
5. Génération de code Maude à partir un modèle formelle RECATNet… ...…… …… 92
6. Vérification avec Maude ……. 94
6.1. Vérification des propriétés ……. 94
7. Conclusion…….. 96

Conclusion générale …… 97
Bibliographie…….. 98
Annexe ……. 102

Table de figures

1 La méthodologie proposée……... 2

I.1 Processus de l'ingénierie Système (IS)……..... 6
I.2 La relation entre UML et SysML…….. 7
I.3 Les différents diagrammes du langage SysML…….. 8
I.4 Diagramme de définition des blocs et leur diagramme interne d'une roue de véhicule……. 10
I.5 Diagramme interne paramétrique…….. 11
I.6 Un exemple de diagramme de paquetage……... 11
I.7 Diagramme de séquence d'un conducteur de véhicule…….. 12
I.8 Diagramme de cas d'utilisation d'un conducteur de véhicule…….……... 13
I.9 Diagramme d'état d'un moteur de véhicule……...…… 14
I.10 Un exemple de diagramme d'activité……... 17
I.11 Diagramme d'exigence d'un véhicule…….. 19

II.1 Relation entre monde réel,modele, meta-modèle……..... 24
II.2 Les niveaux d'abstraction……. 25
II.3 MDA : un processus en Y dirigé par les modèles……. 27
II.4 Transformation principales de MDA…… 29
II.5 Scénario d'une transformation de modèle……. 30
II.6 Type de transformation …… 32
II.7 La structure de langage ATL…… 34

III.1 Méthode générale de modélisation et d'analyse basée sur les réseaux de Petri….. 37
III.2 Les composants graphiques d'un RDP……. 38
III.3 Représentation des éléments du réseau de Petri…….…… 39
III.4 Exemple de réseau de Petri…….. 40
III.5 Franchissement d'une transition……... 42
III.6 Graphe de marquages accessibles……. 43
III.7 Exemple d'un RdP non-bornée……. 44

III.8 Exemple d'un RdP 3-bornée…… 44
III.9 Représentation des transitions vivantes et quasi-vivantes…… 45
III.10 Exemple d'un RdP avec blocage…… 46
III.11 Représentation des Transitions de RdPR……. 46
III.12 Exemple d'un RdPR …….. 47
III.13 Graphe d'accessibilité du RdPR…….. 48
III.14 Franchissement d'une transition élémentaire……. 49
III.15 Franchissement d'une transition abstraite…….. 50
III.16 Franchissement d'une étape de coupure…… 51
III.17 Représentation graphique d'une transition élémentaire de RECATNets……... 52
III.18 Représentation graphique d'une transition abstraite de RECATNets…… 53
III.19 Exemple de RECATNets …….. 55
III.20 Séquence de franchissement de l'exemple précédent……. 56
III.21 Le méta-modèle de RECATNet……. 57
III.22 RECATNet sous extension (ecore)……. 60

IV.1 Démarche générale de transformation de DA vers RECATNet……. 64
IV.2 Transformation de flux de séquence……. 65
IV.3 Transformation le nœud initial…….. 66
IV.4 Transformation le nœud de fin d'activité…….. 67
IV.5 Transformation une action simple……. 68
IV.6 Transformation un flux d'objet…….. 69
IV.7 Transformation le nœud d'union (join node)……... 70
IV.8 Transformation de le noeud de bifurcation (fork node)…….. 71
IV.9 Transformation le nœud de décision ……... 72
IV.10 Transformation le noeud de fusion (merge node)…….. 73
IV.11 Transformation le comportement d'appel (call behavior)……... 74
IV.12 Processus de financement et de lancement d'un projet…….. 76
IV.13 RECATNet pour le processus de financement et de lancement d'un projet…….. 77

V.1 Démarche générale……... 80
V.2 L'activité principale de la machine qui vend les tickets…….. 82

V.3 L'activité appelée procédure de paiement…….. 83
V.4 Le diagramme d'exigence pour le système de la machine qui vend les tickets….. 84
V.5 L'interface principale……... 85
V.6 Interface graphique de transformation DA2RECATNet ……...… 86
V.7 Interface graphique de model.uml ……...…….. 87
V.8 Interface graphique résultat de transformation DA2RECATNet…… 88
V.9 Résultat graphique de transformation…….. 89
V.10 Résultat graphique de transformation (call behavior)…….. 89
V.11 Résultat de transformation sous format GMF…….. 90
V.12 Le DE après l'intégration des expressions extrait de l'éditeur ActRL…… 91
V.13 Interface graphique de transformation RECATNet2Maude…… 92
V.14 Interface graphique de modèle .XMI……... 93
V.15 Interface graphique résultat de transformation RECATNet2Maude……... 94
V.16 Module RECATNET-CHECK…… 95
V.17 Module RECATNET-PRED…… 95
V.78 Résultats de la vérification……. 96

Liste des tableaux

4.1 Les éléments du diagramme d'activité conforme aux spécifications OMG…… 15
4.1 Les différentes formes de l'expression $\mathbf{IC}(\mathbf{p},\mathbf{t})$ ……. 54
5.1 Concepts organisationnels du méta-modèle pour le RECATNet…….. 57

Introduction générale

Motivation :

Le développement des systèmes informatique permet, en général, d'obtenir des résultats satisfaisants dans différents domaines. Cependant, ces systèmes ne sont pas forcément sans erreurs. La vérification formelle est une technique qui vise à évaluer le système conçu afin d'éviter ce type d'erreurs, dont les conséquences peuvent être fatales. par exemple, le système d'atterrissage des avions dans les aéroports.

L'ingénierie dirigée par les Modèles (IDM) est une technologie qui permet de réutiliser des modèles de formalisme appelés méta-modèles ceux-ci sont adaptables à toutes les plateformes. Certains organismes de l'OMG (Object Management Group) s'intéressent de plus en plus à l'IDM et cherchent à la normaliser. Cette association qui est à l'origine de langage de modélisation comme UML, a défini le langage SysML **[1]** (System Modeling Language) qui se veut orienter 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.

Récemment, le formalisme de RECATNet (Récursive ECATNet) a été introduit dans le but de modéliser et proposé comme un modèle alternatif pour spécifier de très grands systèmes à structure dynamique, voir infinie. Les RECATNets font partie des réseaux de Pétri algébriques de haut niveau combinant la puissance expressive des types de données abstraites et des réseaux récurrents de Petri.

Description du problème :

SysML est un langage semi-formel, il est donc impossible de faire directement une vérification formelle sur des spécifications décrites par ce langage. C'est pourquoi nous nous intéressons ici à la transformation du langage SysML et plus particulièrement des diagrammes d'activités vers le formalisme de RECATNet.

A la fin de ce projet, nous allons développer un outil JAVA permettant de vérifier formellement des propriétés fonctionnelles, spécifiées par les diagrammes d'exigences, sur les diagrammes d'activités SysML.

Méthodologie proposée :

Ce schéma montre les processus suivi pour vérifier les exigences sur un système modélisé par un diagramme d'activité SysML.

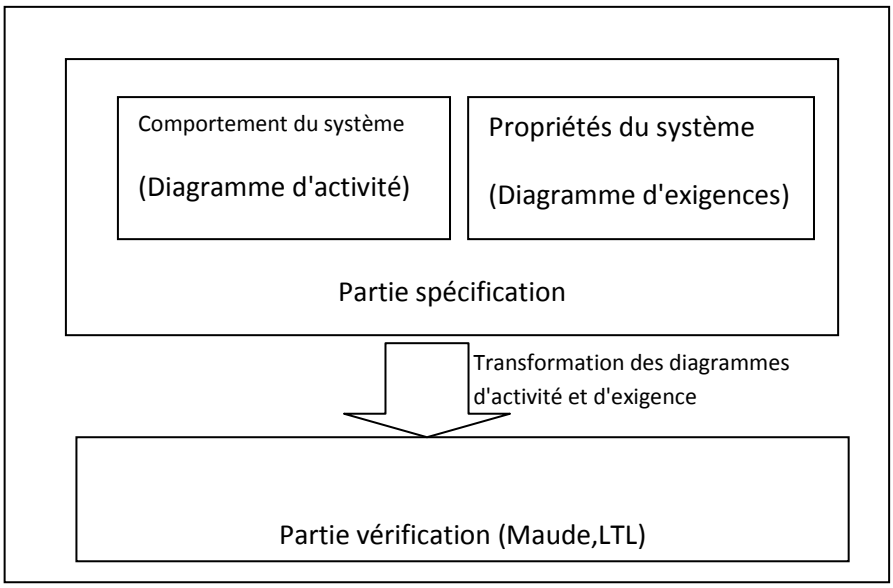


Figure 1 La méthodologie proposée

Organisation du mémoire :

Ce travail sera organisé en cinq chapitres :

— **Chapitre I :** Ce chapitre présente le langage SysML et ses différents diagrammes. Particulièrement, nous focalisons sur la présentation du digramme d'activités et du digramme d'exigences.

— **Chapitre II :** Ce chapitre présentera le domaine d'ingénierie dirigée par les Modèles (IDM) et les différentes notions qui sert pour la transformation de modèle.

— **Chapitre III** : Ce chapitre présente les notions de base de Réseau de Petri (RdP) et leur extension RdP récursif. Enfin, nous présentons le formalisme des RECATNets

— **Chapitre IV :** Ce chapitre présente une transformation des diagrammes d'Activité vers de RECATNet et en détailler les règles de transformation

— **Chapitre V :** Dans ce chapitre, nous exposons d'une manière détaillée un outil JAVA permettant de vérifier des propriétés sur les diagrammes d'activités SysML.

Enfin, le travail ce termine par une conclusion générale.

Chapitre I

SysML

1. Introduction

L'ingénierie système (IS) est une démarche méthodologique pour répondre à des problèmes complexes par la réalisation de solutions logicielles et matérielles. L'essor d'UML[1] est un langage de modélisation très répandu pour les développements logiciels, Il a été utilisé et adapté pour définir un langage de modélisation des systèmes : SysML**[1]** ou Systems Modeling Language. Ce dernier 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. Dans ce chapitre nous présentons l'Ingénierie Système, un bref historique sur SysML puis une description des différents diagrammes en commençant par la modélisation structurelle du système suivi par la modélisation dynamique, en concentrant sur le diagramme d'activité, puis les diagrammes des exigences.

2. Ingénierie Système

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 était utilisée pour l'IS[2], que ce soit pour des représentations concrètes avec des plans ou modèles réduits, ou plus abstraites avec des systèmes d'équations**[2]**.

La démarche de l'IS fait appel à trois grands points de vue complémentaires :

- Fonctionnelle : qui consiste à décrire les actions effectues par le système pour répondre à la question 'A quoi sert-il ?'
- Structurelle : permet de décrire la structure du système et ses composants ainsi que les relations entre ces composants, pour répondre aux questions 'De quoi est-il composé ? ' et 'Comment est-il organisé ? '
- Comportementale : permet de décrire le comportement du système afin de répondre, à la question 'Comment il se comporte ? '

[1] Unified modeling language
[2] Ingénierie système

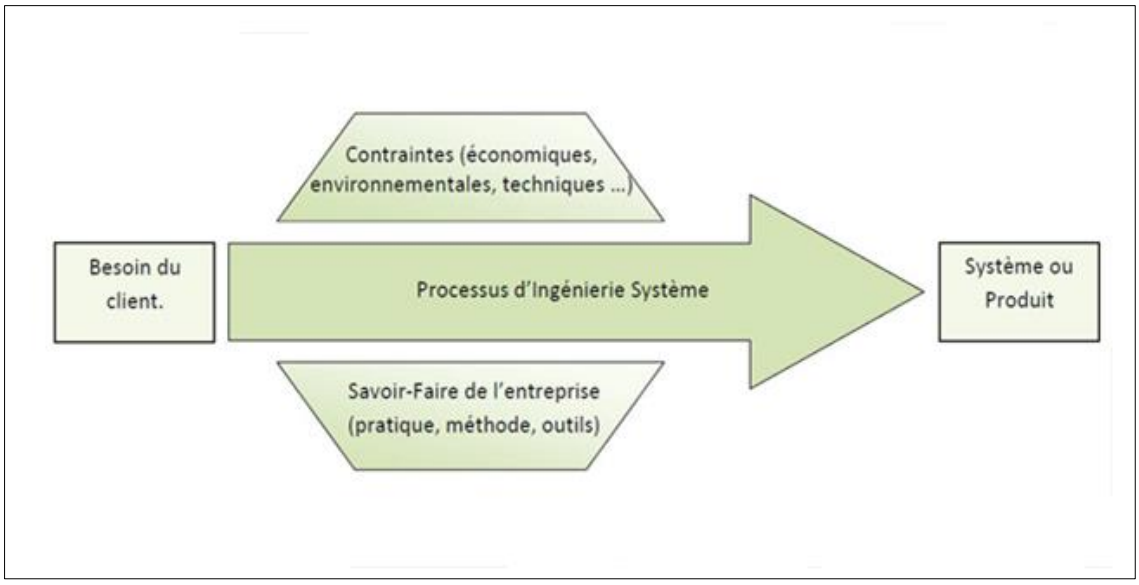


*Figure I.1 Processus de l'ingénierie Système (IS)***[3].**

3. Historique

La modélisation avec le langage UML est une pratique bien établie dans l'industrie logicielle. Bien que le langage UML permette par son caractère à usage général d'adresser de nombreux besoins pour l'IS, il est nécessaire d'adapter ce langage de modélisation par la définition de « profils UML ».

Le besoin de définir un langage basé sur UML pour l'IS a été initié en 2001 par l'organisation internationale de l'ingénierie système INCOSE (International Council on Systems Engineering), qui s'est mise en relation avec l'OMG (Object Management Group), organisme responsable d'UML, pour créer le groupe d'intérêt spécialisé dans les domaines de l'IS ou « SE DSIG » (Systems Engineering Domains Special Interest Group).Suite à de nombreux ateliers, l'INCOSE[3] et l'OMG[4] ont défini et publié la demande de proposition « UML pour l'IS » en 2003 (UML for SE RFP - Request For Proposal).

Plusieurs membres de l'industrie (BAE, Motorola, oose, Boeing.), éditeurs d'outils (IBM, Sparx Systems.), universités et organisations (INCOSE, AP233) ont travaillés sur la définition du langage de modélisation système SysML (Systems Modeling Language).

[3] International Council on Systems Engineering
[4] Object Management Group

- Juillet 2006 : OMG annonce l'adoption de SysML
- Septembre 2007 : spécifications de la version 1.0 rendues officielles
- Décembre 2008 : SysML v1.1
- Juin 2010 : SysML v1.2 (version actuelle)**[2].**

4. Présentation SysML

SysML (Systems Modeling Language) est basé sur UML et remplace la modélisation de classes et d'objets par la modélisation de blocs pour un vocabulaire plus adapté à l'Ingénierie Système. Un bloc englobe tout concept logiciel, matériel, données, processus, et même la gestion des personnes**[4].**

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 la relation entre SysML et UML

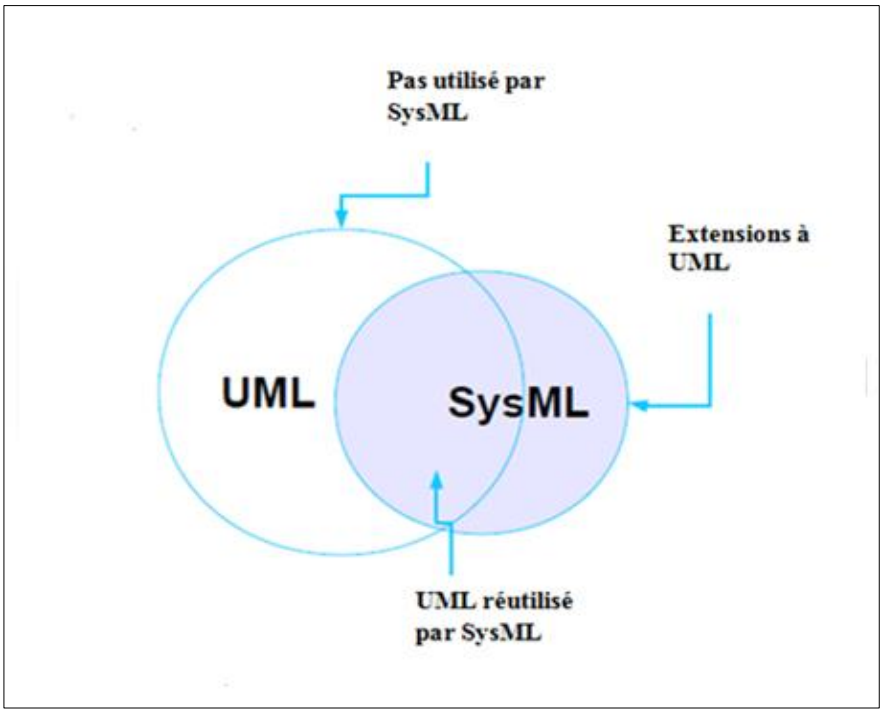


Figure I.2 La relation entre UML et SysML.

4.1. Diagrammes SysML

À partir des informations recueillies dans la spécification OMG d'UML/SysML **[1],** les diagrammes SysML se divise en trois catégories : les diagrammes de modélisation structurelle, le diagramme d'exigence et des diagrammes de modélisation de comportement, c'est-à-dire dynamique. La figure I.3 présente les neuf diagrammes proposés par SysML.

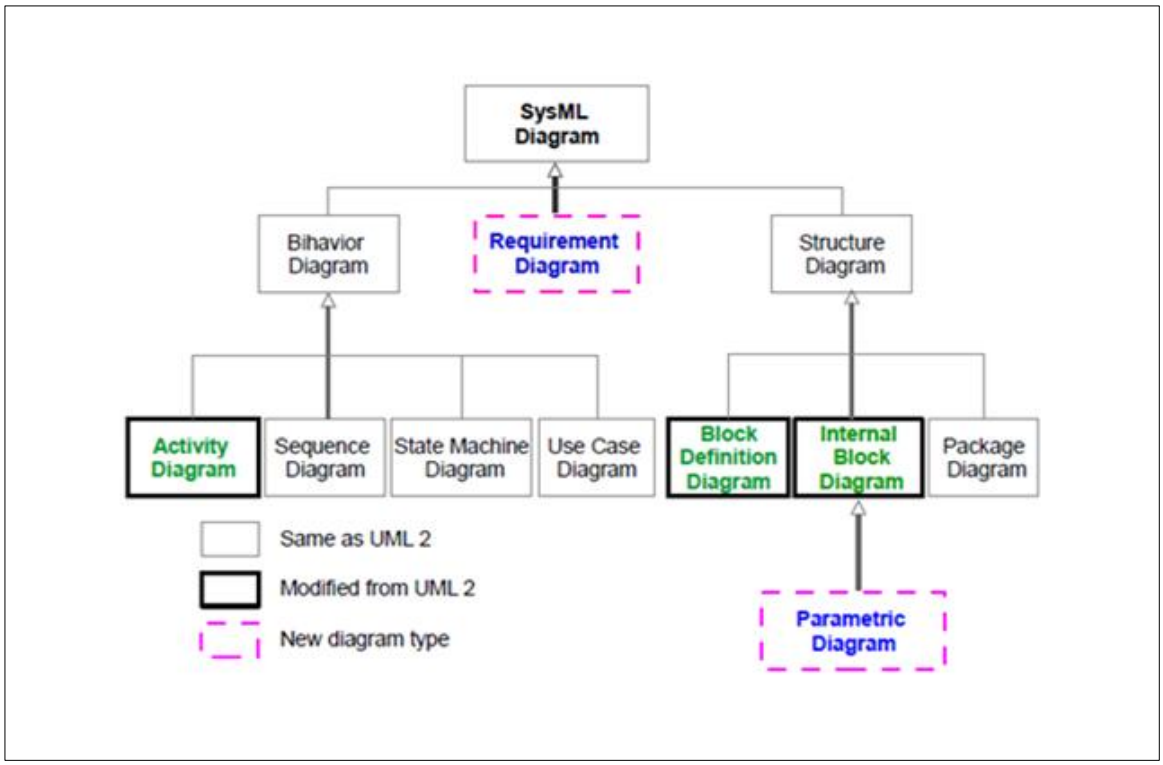


Figure I.3 Les différents diagrammes du langage SysML[2].

4.1.1. Modélisation structurelle

SysML propose pour la modélisation structurelle quatre diagrammes : Diagramme de définition de blocs, diagramme de bloc interne, diagramme paramétrique et diagramme de paquetage.

4.1.1.1. Diagramme de définition de blocs

Le diagramme de définition de bloc (BDD, ou Block Definition Diagram en anglais) 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 logiciel**[5].**

Par rapport à UML, le BDD[5] de SysML redéfinit le diagramme de classe en remplaçant les classes par des blocs**[4]**. La Figure I.4 ci-dessous du diagramme de définition de bloc pour un véhicule.

4.1.1.2. Diagramme de bloc interne

Le diagramme de bloc interne (IBD, ou Internal Block Diagram) décrit la vue interne d'un bloc, il se base sur le BDD pour représenter l'assemblage final des blocs qui composent un bloc composite. Il est surtout utile pour montrer les connexions contextuelles entre les parties d'un bloc, ce qui ne peut pas être représenté sur un bdd.

Les blocs composant le bloc principal et définit dans le BDD. Ils sont instanciés en parties. Ces parties sont assemblées par des connecteurs qui relient leurs ports**[2]**.

Les ports peuvent être de deux natures :

- **flux** (flow port) : ce type de port autorise la circulation de flux physiques entre les blocs. La nature de ce qui peut circuler va des fluides aux données, en passant par l'énergie.
- **standard** : ce type de port autorise la description de services logiques entre les blocs, au moyen d'interfaces regroupant des opérations**[6]**.

[5] Block Definition Diagram

La Figure I.4 montre l'IBD du bloc véhicule dans le diagramme de définition des blocs.

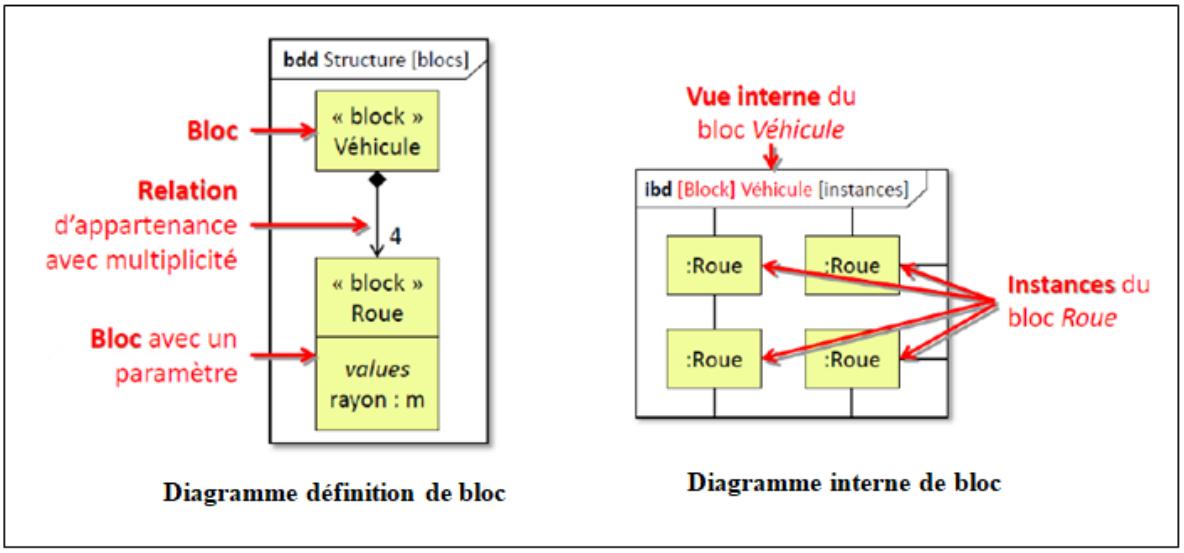


Figure I.4 Diagramme de définition de bloc et leur diagramme interne d'une roue de véhicule

4.1.1.3. Diagramme paramétrique

Le diagramme paramétrique est un diagramme structurel (Paramétric Diagram) dans le langage SysML. Ce diagramme est une extension du diagramme de définition de blocs et il partage donc les mêmes éléments graphiques.

Il présente la particularité de pouvoir connecter entre elles des contraintes ajoutées au diagramme de blocs par le biais d'un bloc particulier, dit "de contraintes" qui contient des paramètres et une relation (mathématique) les reliant**[7].**

La figure I.5 montre le diagramme paramétrique basé sur l'exemple de véhicule précédent :

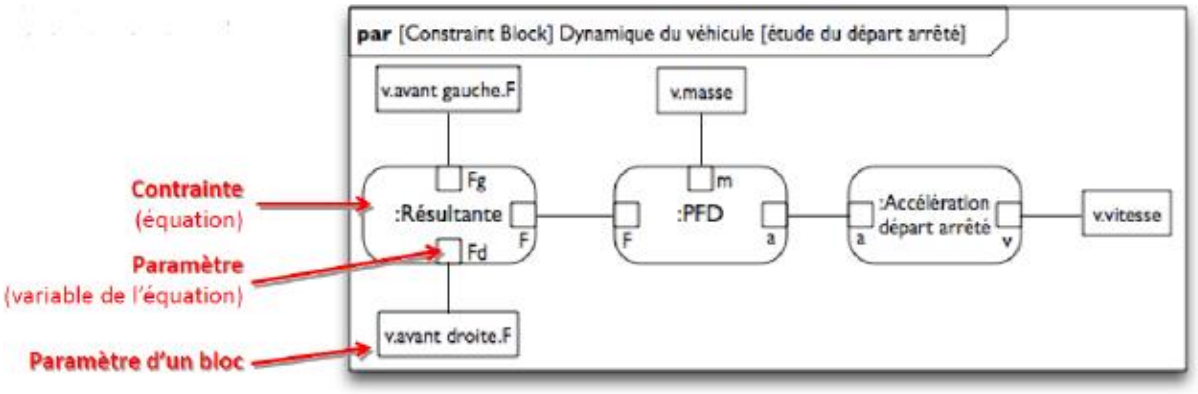


Figure I.5 Diagramme interne paramétrique

4.1.1.4. Diagramme de paquetage

Le diagramme de paquetages est un diagramme structurel (Package Diagram) dans le langage SysML.

L'objectif de ce diagramme est de créer des "macros-associations" de concepts ou d'éléments afin de relier différentes parties d'un même modèle. Un paquetage est un élément "conteneur" permettant le regroupement de tout élément graphique (blocs, interfaces, acteurs...) et peuvent être imbriqués les uns dans les autres**[7].**

La figure I.6 présente un simple exemple de diagramme de paquetage.

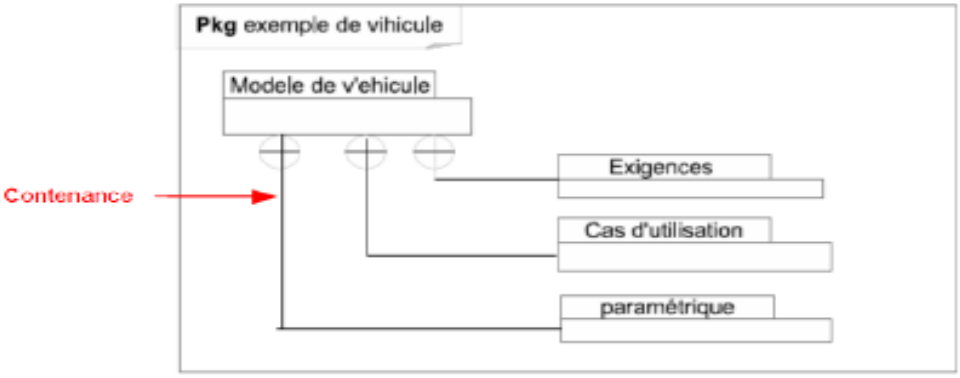


Figure I.6 Un exemple du diagramme de paquetage

4.1.2. Modélisation dynamique

La modélisation de l'aspect dynamique du système avec SysML repose sur une sélection de quatre diagrammes UML2 : diagrammes de cas de séquence, d'utilisations, d'états et d'activité.

Parmi ces diagrammes, seul le diagramme d'activité comporte quelques modifications pour SysML.

4.1.2.1. Diagramme de séquences

Le diagramme de séquence est un diagramme comportemental (Sequence Diagram) dans le langage SysML.

Il représente les éléments intervenant dans un scénario ou un processus, ainsi que les messages échangés dans un ordre chronologique.

Les éléments intervenant sont représentés par des lignes de vie (lifetime en anglais) pouvant être des instances de blocs du modèle. Cette instanciation de blocs établit un lien avec la modélisation structurelle du système donc une cohérence dans le modèle :

- On peut accéder aux propriétés du bloc d'une ligne de vie
- Chaque message échangé peut être utilisé pour l'identification des opérations de blocs

L'ensemble des propriétés du diagramme de séquence utilisées en UML sont également disponibles avec SysML : messages synchrones ou asynchrones, opérateurs (ex : alt, loop, opt, par), références vers d'autres diagrammes de séquence, etc**[4].**

La figure I.7 représente un exemple de diagramme de séquence d'un conducteur de véhicule.

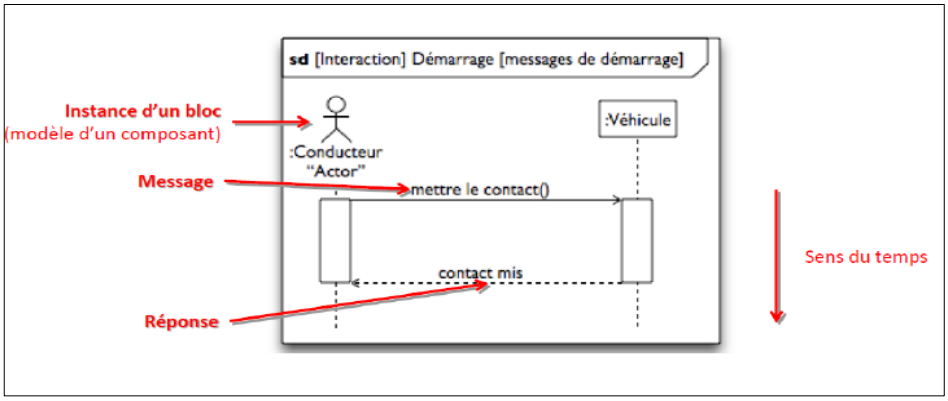


Figure I.7 Diagramme de séquence d'un conducteur de véhicule

4.1.2.2. Diagramme de cas d'utilisations

Tout comme avec UML, on utilise les diagrammes de cas d'utilisations pour identifier les acteurs et les cas d'utilisations d'un point de vue utilisation du système, interactions acteurs/système.

Les cas d'utilisations permettent de définir les frontières, le périmètre fonctionnel du système. 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**[8].**

La figure I.8 présente représente un diagramme de cas d'utilisation d'un conducteur de véhicule

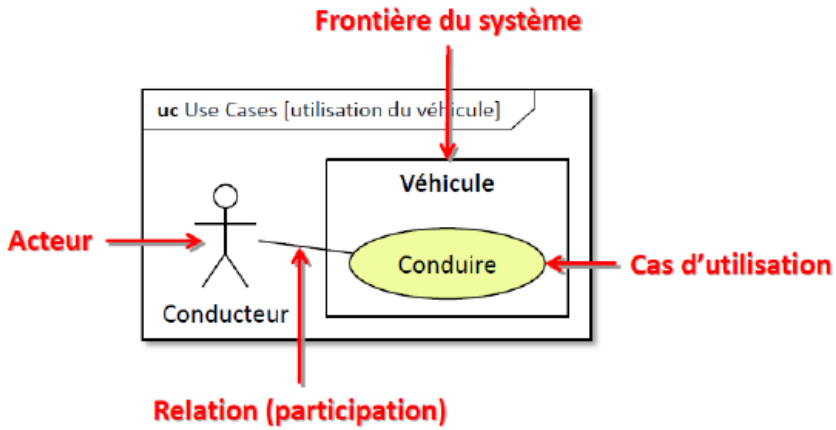


Figure I.8 Diagramme de cas d'utilisation d'un conducteur de véhicule

4.1.2.3. Diagramme d'état

Le diagramme d'états est un diagramme comportemental appelé State Machine Diagram dans le langage SysML.

Le diagramme d'états est rattaché à un bloc qui peut être le système, un sous-système ou un composant. Le comportement décrit par ce type de diagramme sert à montrer les différents états pris par le bloc en fonction des événements qui lui arrivent. Un état représente une situation d'une durée finie durant laquelle un système exécute une activité, satisfait à une certaine condition ou bien est en attente d'un événement. Le passage d'un état à un autre se fait en franchissant une transition.

Les éléments graphiques utilisés sont principalement des rectangles. Les transitions sont représentées par des flèches orientées ayant un état de départ et un état cible. La transition est franchie lors d'une occurrence de l'événement rattaché à la transition**[7].**

La figure I.9 représente un exemple d'un diagramme d'état d'un moteur de véhicule

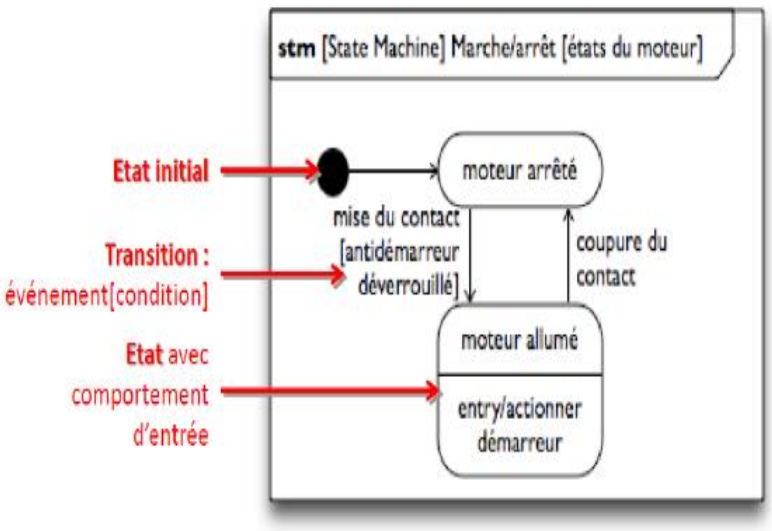


Figure I.9 Diagramme d'état d'un moteur de véhicule

4.1.2.4. Diagramme d'activité

Le diagramme d'activité est un diagramme comportemental (DA[6] appelé Activity Diagram en anglais), et il est utilisé pour représenter les étapes d'un processus, impliquant 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.

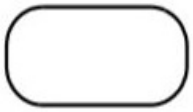
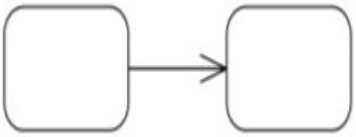
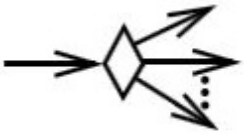
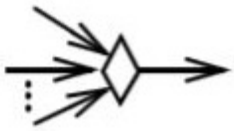
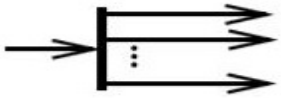
SysML a rajouté quelques spécificités :

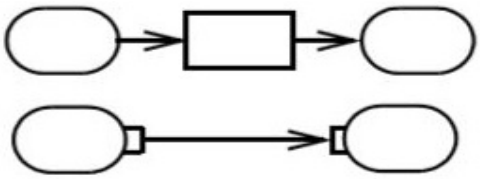
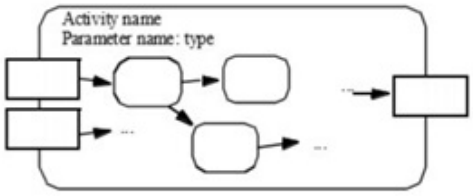
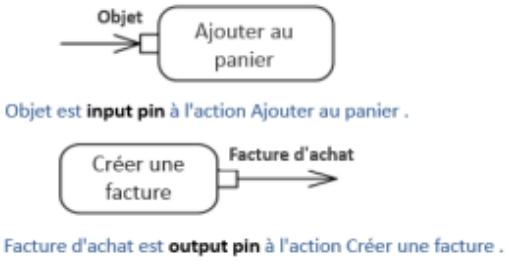
- Notion de contrôle pour activer ou désactiver les actions en cours (Control Value).
- Spécification de la nature du débit sur le flot : système continu ou discret.
- Définition de taux et de probabilité sur les flux de contrôle ou d'objets**[4].**

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 montrée par un symbole de boudine, également appelé comme

[6] Diagramme d'activité

		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 À suivre dans la page 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 nœud 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.

Object Flow		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.
Pins d'entrée et de sortie		À suivre dans la page suivante et pour exprimer les valeurs passées en argument à une activité ainsi que les valeurs de retour.

Un exemple d'un tel diagramme est présenté sur la Figure I.10.

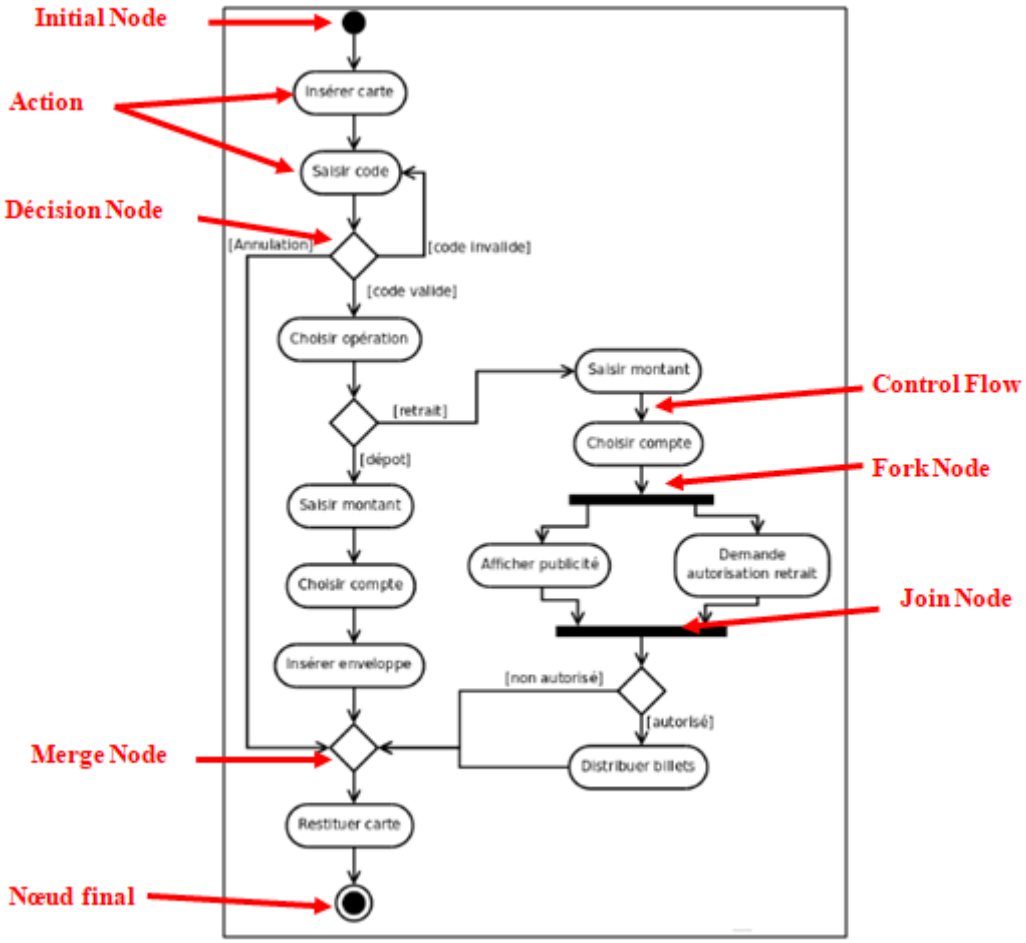


Figure I.10 Un exemple de diagramme d'activité

4.1.3. Modélisation fonctionnelle

4.1.3.1. Diagramme d'exigences

Le diagramme des exigences, appelé Requirement Diagram dans le langage SysML, est le seul diagramme transversal du langage SysML.

Une exigence permet de spécifier une capacité ou une contrainte qui doit être satisfaite par un système. Elle peut spécifier une fonction que le système devra réaliser ou une condition de performance, de fiabilité, de sécurité, etc. Les exigences servent à établir un contrat entre le client et les réalisateurs du futur système**[6].**

Les éléments graphiques sont principalement des rectangles avec un titre représentant les exigences, un identifiant sous forme de numéro et une description textuelle. Il est possible de relier les exigences entre elles.

SysML définit de nouveaux types d'associations (liens de dépendance stéréotypés) **[2]**:

- Derive : une ou plusieurs exigences sont dérivées d'une exigence
- Satisfy: un ou plusieurs éléments du modèle (par exemple un bloc) permettent de satisfaire une exigence
- Verify: un ou plusieurs éléments du modèle (par exemple un « test case ») permettent de vérifier et valider une exigence
- Refine: un ou plusieurs éléments du modèle, par exemple un cas d'utilisation, redéfinit une exigence

SysML définit de nouveaux commentaires stéréotypés permettant d'associer une explication à des associations ou éléments du modèle :

- Problem: commentaire dont la description pose le problème ou le besoin qui a donné lieu à la création de l'association ou de l'élément associé
- Rationale: commentaire dont la description indique la raison ou la justification par rapport à l'élément ou l'association associé.

Dans la figure I.11 représente l'exemple des exigences du véhicule.

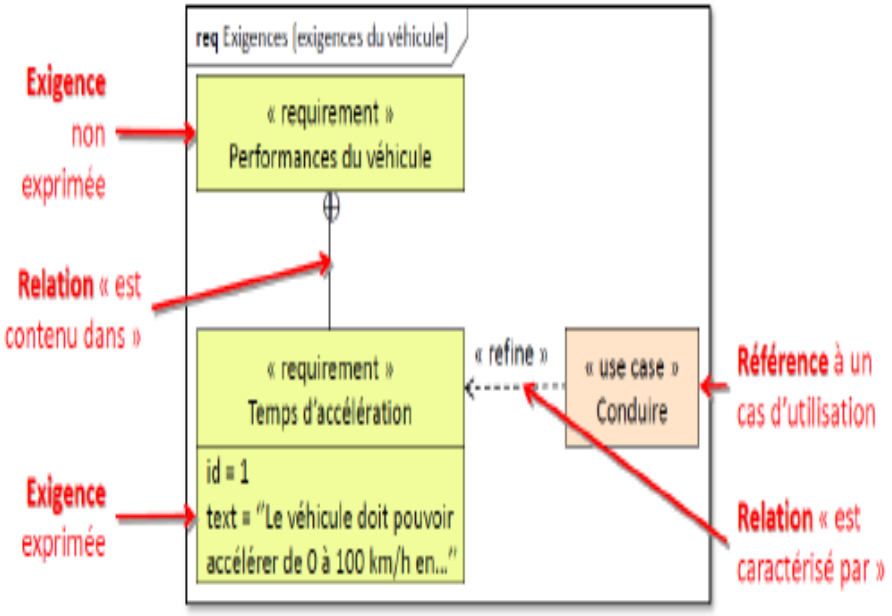


Figure I.11 Diagramme d'exigence d'un véhicule

5. Outils SysML

Le langage *SysML* a été intégré par de nombreux éditeurs d'outils commerciaux ou open source :

- Sparx Systems Enterprise Architect (plugin *SysML* ou version Ultimate requise).
- IBM Rational Software Modeler (plugin d'une société tierce disponible).
- MagicDraw (plugin *SysML* requis)
- Open source : POLARSYS, TOPCASED. **[2]**

Dans ce travail, nous avons choisi d'utiliser l'environnement POLARSYS, qui est une boite à outils open source (environnement Eclipse).

6. Conclusion

SysML permet la spécification, l'analyse, la conception, la vérification et la validation de nombreux systèmes .Il se définit comme une extension d'un sous-ensemble d'UML via l'utilisation du mécanisme de profil défini par UML.

IL est donc un langage suffisamment flexible pour permettre son utilisation dans divers cas de développement, cet aspect positif est constamment mis en valeur. C'est sa grande force, mais en même temps sa plus grande faiblesse, il souffre principalement d'une faible

sémantique formelle. Dans ce chapitre nous avons vu les objectifs du langage de modélisation SysML, après une brève explication de la problématique d'ingénierie système. Nous avons expliqués pourquoi l'OMG a proposé un nouveau langage de modélisation dédié à ingénierie système en présenter également les différents diagrammes SysML, et détailler chacun avec exemple.

Chapitre II

L’ingénierie dirigée par les modèles

1. Introduction

Suite à l'approche objet des années 80, l'ingénierie du logiciel s'oriente aujourd'hui vers l'ingénierie dirigée par les modèles (IDM) ou Model-driven engineering (MDE).
L'Ingénierie Dirigée par les Modèles (IDM) est une discipline récente du génie logiciel qui met les modèles au premier plan au sein du processus du développement logiciel. Elle a apporté plusieurs améliorations significatives dans le développement des systèmes logiciels complexe en fournissant des moyens permettant de passer d'un niveau d'abstraction à un autre ou d'un espace technologique à un autre. Cependant, la gestion des modèles peut s'avérer lourde et coûteuse. Pour pouvoir mieux répondre aux attentes des utilisateurs, il est nécessaire de fournir des outils flexibles et fiables pour la gestion automatique des modèles ainsi que des langages dédiés pour leurs transformations **[9].** Dans ce chapitre, nous allons présenter les notions et les concepts relatifs à l'ingénierie dirigée par les modelés. Une attention particulière sera mise sur la transformation de modèles et son importance dans le cadre d'une ingénierie dirigée par les modèles. Enfin, nous décrirons le langage ATL (Atlas Transformation Langage), qui est un langage de transformation de modèle.

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 d'OMG[1] qui reprend et restreint les concepts de l'IDM **[10].**

[1] Object Management Group

3. Les principes généraux de l'IDM

3.1. La Modélisation

L'approche objet est fondée sur deux relations essentielles, « InstanceDe »et « HériteDe », 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. Nous considérerons dans la suite de cette thèse la définition suivante d'un modèle **[11].**

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**[11].**

Pour connaître la nature des différents modèles utilisés dans les systèmes informatiques, deux relations peuvent être identifiées. La première relation, appelée "représenté par", identifie un rôle de représentation d'un objet donné, modélisé à travers un modèle. Par exemple, un programme informatique peut être "représenté par" un ensemble de diagrammes de flux de données. Un modèle de données pourrait également représenter un autre modèle. Par exemple, une fonction mathématique peut être représentée par une approximation numérique. La seconde relation, appelée "conforme à", indique la dépendance d'un modèle par rapport à un langage de modélisation **[12]**.

3.2. La Méta-Modélisation

Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle **[13],** c.-à-d. le langage de modélisation **[11].**

Un méta-modèle est aussi un modèle, il est définit par **[14]:**

- Une syntaxe abstraite qui définit les concepts de base du langage.
- Une syntaxe concrète qui définit le type de notation qui sera utilisé pour chaque concept abstrait et qui peut être graphique, textuelle ou mixte.
- Une sémantique qui définit comment les concepts du langage doivent être interprétés par les concepteurs mais surtout par les machines.

Selon **[14],** un méta-modèle définit la structure que doit avoir tout modèle conforme à ce méta-modèle. Autrement dit, tout modèle doit respecter la structure définie par son méta-modèle.

La figure II.1 illustre les trois niveaux : modèle, méta-modèle et monde réel.



Figure II.1 Relations entre monde réel, modèle, méta-modèle.

Définition (Métamétamodèle) : Un métamétamodèle 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 II.2). Le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les méta-modèles permettant la définition de ces modèles (p. ex. UML) constituent le niveau M2. Enfin, le méta-métamodèle, unique et méta-circulaire, est représenté au sommet de la pyramide (niveau M3). 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 **[15][16][17]**.

La figure II.2 montre une représentation graphique de ses quatre niveaux d'abstraction.

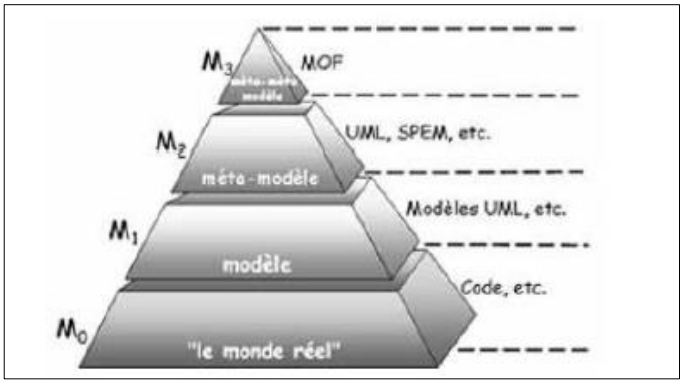


Figure II.2 Les niveaux d'abstraction.

4. L'approche MDA (Model Driven Architecture)

L'OMG (Object Management Group) a introduit l'approche nommée Architecture Dirigée par les Modèles MDA (Model Driven Architecture) afin de faciliter le développement et la maintenance des systèmes essentiellement logiciels. Cette approche qui reste imprécise consiste à manipuler différents modèles de l'application à produire, depuis une description très abstraite jusqu'à une représentation correspondant à l'implantation effective du système, la modélisation se base sur les standards de l'OMG et en particulier le langage de modélisation UML (Unified Modeling Language).

L'approche MDA vise à séparer la conception en deux niveaux : au niveau supérieur on décrit le modèle métier du système en termes d'un modèle indépendant de la plateforme PIM (Platform Independent Model), puis ce modèle est spécifié dans un modèle technologique de plus bas niveau dépendant de la plate-forme spécifique PSM (Platform Specific Model). Cette séparation permet de se concentrer d'abord sur la structure et la fonctionnalité du système, et plus tard sur l'implémentation, permettant au même modèle abstrait d'être implémenté sur différentes plateformes.

Cependant le terme plateforme reste ambigu et cela dû au manque de précision, car il est difficile de savoir s'il s'agit d'une plate-forme matérielle ou logicielle ou autres. MDA est une marque commerciale déposée par l'OMG. En conséquence, la communauté scientifique fait

souvent référence au MDE[2] pour exprimer des idées d'ingénierie des modèles sans se restreindre aux concepts de MDA détaillés dans les standards de l'OMG**[12].** Les quatre principaux types de modèles définis dans l'approche MDA sont les suivants **[18]** (figure II.3)

- CIM (Computation Independant Model) : Appelé aussi modèle de domaine ou modèle métier, le CIM capture les exigences en termes de besoins et d´écrit la situation dans laquelle le système sera utilise. 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 raffine 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.

[2] Model Driven Enineering

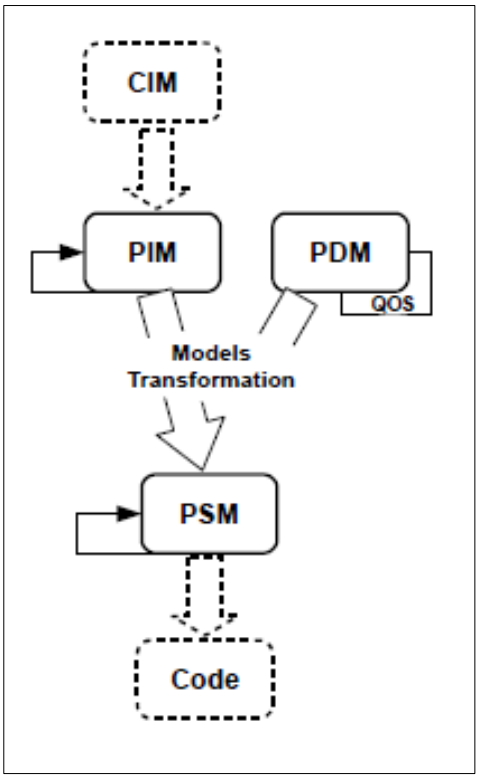


Figure II.3 MDA : un processus en Y dirigé par les modèles.

4.1. 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 **[14].**

La figure II.4 reprend les phases de MDA et illustre les principales transformations de modèles MDA **[14]:**

- **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 exprimes 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 exprime par le client.

- **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**.**

La Figure II.4 donne une vue générale de l'approche MDA. Nous constatons que la construction d'une nouvelle application commence par l'élaboration d'un ou de plusieurs modèles d'exigences (CIM). Elle se poursuit par l'élaboration des modèles d'analyse et de conception abstraite de l'application (PIM). Ceux-ci doivent en théorie être partiellement générés à partir des CIM afin que des liens de traçabilité soient établis. Les modèles PIM sont des modèles pérennes, qui ne contiennent aucune information sur les plates-formes d'exécution. Pour réaliser concrètement l'application, il faut ensuite construire des modèles spécifiques des plates-formes d'exécution. Ces modèles sont obtenus par une transformation des PIM en y ajoutant les informations techniques relatives aux plates-formes. Les PSM n'ont pas pour vocation d'être pérennes. Leur principale fonction est de faciliter la génération de code. La génération de code à partir des modèles PSM n'est d'ailleurs pas réellement considérée par MDA. Celle-ci s'apparente plutôt à une traduction des PSM dans un formalisme textuel **[14].**

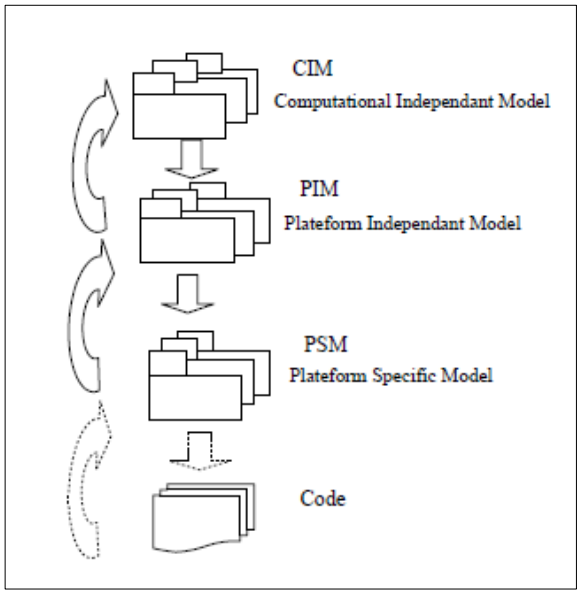


Figure II.4 Transformation principales de MDA.

5. Transformation de modèles

Transformations de modèles L'IDM considère les opérations de transformations de modèles comme le moteur de développement, que ce soit pour l'analyse, l'optimisation ou la génération de code. Cette procédure consiste à générer, à partir d'un ou de plusieurs modèles sources d'un système, un ou plusieurs modèles cibles du même système. Les modèles sont dans tous les cas conformes à leurs méta-modèles respectifs

Ces transformations sont gouvernées par un ensemble de règles utilisées par le moteur de transformation. Ce moteur prend en entrée le modèle source, exécute les règles de transformation et génère le modèle cible **[19].**

La figure II.5 présente un scénario de transformation de modèle avec ses principaux concepts. Un modèle source en entrée et un modèle cible en sortie. Les deux modèles sont conformes à leurs méta-modèles respectifs. Le méta-modèle exprime la syntaxe abstraite de la notation du modèle. Une transformation respecte les méta-modèles et est réalisée par l'intermédiaire d'un moteur de transformation.

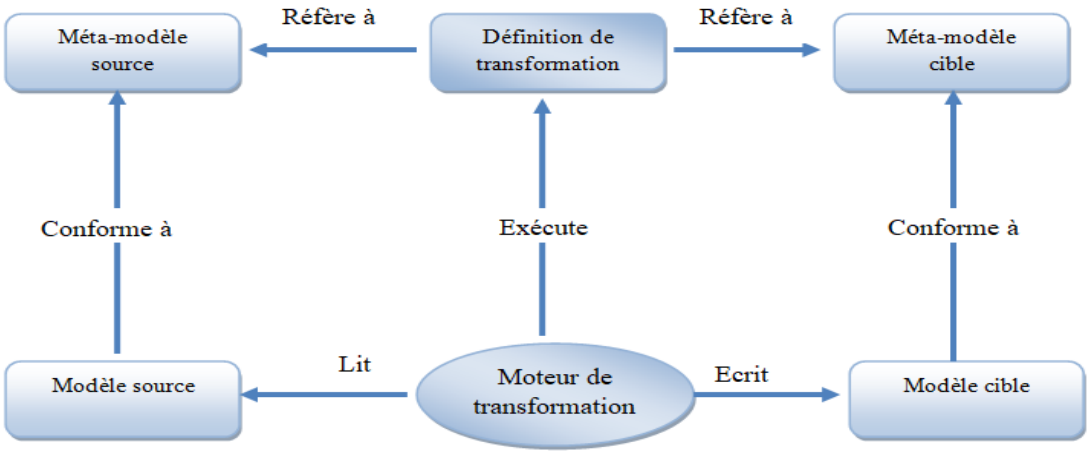


Figure II.5 Scénario d'une transformation de modèle.

Les transformations de modèles se partagent également en deux grandes classes **[20]** : les transformations « Modèle vers Code », et les transformations « Modèle vers Modèle »

- **Transformation Modèle vers Modèle**

Les transformations de type modèle vers modèle consistent à transformer un modèle source en un modèle cible, ces modèles peuvent être des instances de différents métamodèles. Elles offrent des transformations plus modulaires et faciles à maintenir. Dans les cas où on trouve un grand espace d'abstraction entre PIMs et PSMs, il est plus facile de générer des modèles intermédiaires qu'aller directement vers le PSM cible. Les modèles intermédiaires peuvent être utiles pour l'optimisation ou bien pour des fins de déboguage. De plus, les transformations de type modèle vers modèle sont utiles pour le calcul des différentes vues du système et leurs synchronisation**[21]**. Nous pouvons distinguer cinq techniques de transformation « Modèle vers Modèle » **[19]** :

- **Approche de manipulation directe :** Cette approche est basée sur une représentation interne des modèles source et cible, en plus des API (Application Programming Interface) pour les manipuler.

- **Approche relationnelle** : Cette approche utilise les contraintes pour spécifier les relations entre les éléments du modèle source et ceux du modèle cible en utilisant une logique déclarative basée sur des relations mathématiques.

- **Approche basée sur les transformations de graphes** : Cette approche convient lorsque les modèles sont représentés par des graphes. Elle exprime les transformations sous une forme déclarative. Les règles de transformation sont définies sur des parties de modèle et non pas sur des éléments basiques

- **Approche basée sur la structure** : Divisée en deux étapes, la première se charge de la création d'une structure hiérarchique du modèle cible, la seconde ajuste ses attributs et ses références.

- **Approche hybride:** Les approches hybrides représentent une combinaison des différentes techniques ou alors celle d'approches utilisant à la fois des règles déclaratives et impératives.

- **Transformation Modèle vers Code**

Il existe deux types de transformations« Modèle vers Code »: la première est basée sur le principe du visiteur (visitor-based) où l'on se sert du modèle et ses éléments qui rapprochent sa sémantique du langage de programmation pour obtenir le code ;la deuxième est basée sur le principe des patrons (Template-based) avec laquelle on accède aux informations du modèle source en utilisant les fragments de méta-code dans le code cible**[11].**

5.1. Type de Transformation

Un des concepts les plus importants de l'IDM est la transformation de modèle : partir d'un modèle "source" et aller vers un modèle "cible". L'IDM distingue deux types de transformation**[22]** :

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)**.**

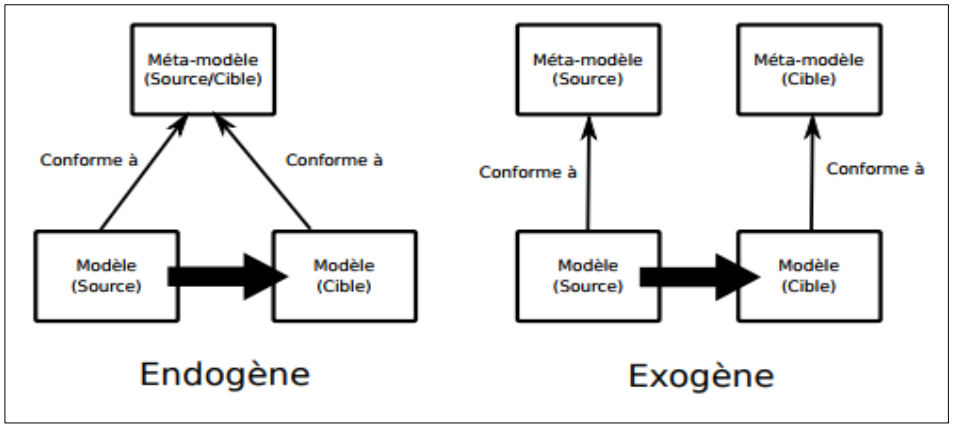


Figure II.6 Type de transformation.

L'IDM et la méta-modélisation permettent d'établir des règles à un niveau d'abstraction plus élevé (M2 du MOF[3]), d'établir les règles entre un méta-modèle source et un méta-modèle cible. Les règles sont donc définies pour toutes les instances du méta-modèle source (tous les modèles source). Il y a alors possibilité d'automatiser la transformation. Pour l'écriture des règles de transformation au niveau méta-modèle, l'OMG à défini le standard QVT **[23]** (Query/View/Transformation). A ce jour les outils qui supportent ce standard ne sont pas suffisamment matures et ne sont pas encore adaptés à des applications réelles. 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 boîte à outils TOPCASED d'Eclipse **[22].**

6. Langage de transformation ATL

Le langage ATL[4] a été intégré dans la plate-forme de modélisation EMF[5]. Dans cette section on va introduire la plate-forme de modélisation EMF, ensuite, on défini le langage de transformation ATL.

[3] Meta Object Facility
[4] Atlas Transformation Langage
[5] Eclipse Modeling Framework

6.1. La plate-forme de modélisation EMF

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[6] 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**[22].**

6.2. Atlas Transformation Langage

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) **[24]** 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[7]. Sachant qu'il existe des dialectes d'XMI, ADT est adapté pour interpréter des méta-modèles 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 Meta-Meta-Model) qui est une forme textuelle simplifiée d'EMOF[8] permettant de décrire des méta-modèles et des modèles. En effet, ADT supporte les modèles décrits dans différentes dialectes d'XMI, qui peuvent par exemple être décrits au format KM3[9] et transformés au format XMI voulu. ATL est défini par un modèle MOF pour sa syntaxe abstraite et possède une syntaxe

[6] Graphical Modeling Framework
[7] XML Metadata Interchange
[8] The Essential Meta Object Facility
[9] Kernel Meta Meta Model

concrète textuelle. Pour accéder aux éléments d'un modèle, ATL utilise des requêtes sous forme d'expressions OCL[10] **[24].**

6.3. Programmation d'un module ATL

Un module ATL correspond à la transformation d'un ensemble de modèles sources vers un ensemble de modèles cibles conformes à leurs méta-modèles. Sa structure est composée d'une section d'en-tête, d'une section d'importation facultative, d'un ensemble de helpers et d'un ensemble de règles.

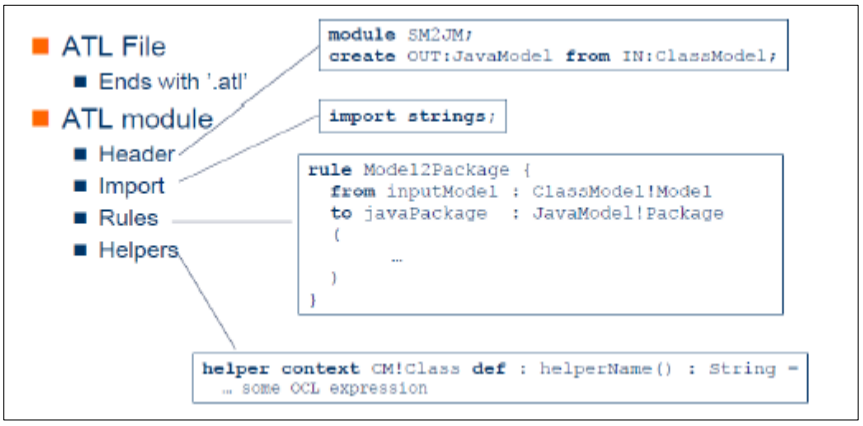


Figure II.6 La structure de langage ATL.

6.3.1. La section d'en-tête

La section d'en-tête (header en anglais) définit le nom du module de transformation ainsi que le nom des variables correspondantes aux modèles sources et cibles. Elle encode également le mode d'exécution du module qui peut être soit le mode normal (défini par le mot clé from) ou le mode de raffinage (défini par le mot clef refining).

6.3.2. La section d'importation

Cette section est optionnelle, elle permet de déclarer les bibliothèques ATL qui doivent être importées. Le code source ATL suivant représente l'en-tête d'importation de la bibliothèque Strings Par exemple:

.

[10] Object Constraint Language

6.3.3. Les helpers

Les fonctions ATL sont appelées des helpers d'après le standard OCL (Object Constraint Language) sur lequel ATL se base. Les fonctions helper commence par le mot-clé helper et se termine par un point-virgule, entre les deux est l'expression OCL. Ce dernier définit deux sortes de helpers : **opération** et **attribut**.

6.3.4. Les règles

Les règles (représenter dans la partie *rules*) décrivent la transformation d'un modèle source (la partie *from*) à un autre modèle cible (la partie *to*) en rapportant un méta-modèle.

6.3.5. Les étapes de la transformation

Les étapes à suivre pour réaliser une transformation de modèles avec ATL et EMF :

- Définir un méta-modèle (ecore) source
- Définir un méta-modèle (ecore) cible
- Ecrire les règles de transformation ATL.
- Exécuter la transformation en indiquant :
  - ✓ Le méta-modèle source
  - ✓ Le méta-modèle cible
  - ✓ Le modèle source
  - ✓ Le nom et le chemin du modèle cible.

7. Conclusion

Le développement logiciel dirigé par les modèles a pour principal objectif de concevoir des applications en séparant les préoccupations et en plaçant les notions de modèles, métamodèles et transformations de modèles au centre du processus de développement.

Dans ce chapitre Nous avons défini les principes de base de l'IDM et 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. En plus, des langages de transformation ont été développés et implémentés. Parmi ces langages on trouve ATL.

Chapitre III

Formalisme de RECATNets

1. Introduction

Un réseau de Petri est un modèle mathématique servant à représenter divers systèmes (informatiques, industriels…) travaillant sur des variables discrètes.

Le formalisme des réseaux de Petri (RdP) a été proposé par *Carl Adam Petri* en 1962 à l'université Darmstadt dans sa thèse intitulée : "Communication avec des automates" comme un outil mathématique permettant la modélisation des systèmes dynamiques à événements discrets**[25].**

Les réseaux de petri voient beaucoup d'extensions permis ces extension les réseaux de Petri récursif (abrégés RdPR) qu'est été introduit par El Fallah et Haddad en 1996. L'avantage des réseaux de Petri est qu'ils confectionné d'une représentation graphique facile à comprendre tout en alliant des fondations mathématiques solides ce qui les rend utilisables par les concepteurs des systèmes .ainsi que par les spécialistes du domaine de la vérification comportementale de ces systèmes.

L'objectif de ce chapitre est principalement de définir les concepts de base de Réseau de petri et leur l'extension RdP récursif. Ensuite nous définissons un modèle, à savoir les EcatNets récursifs basés sur une combinaison saine de réseaux algébriques à terme concomitants étendus et de réseaux de Petri récursifs.

La figure III.1 montre la méthode générale basée sur le formalisme des réseaux de Petri pour la modélisation et l'analyse des systèmes.

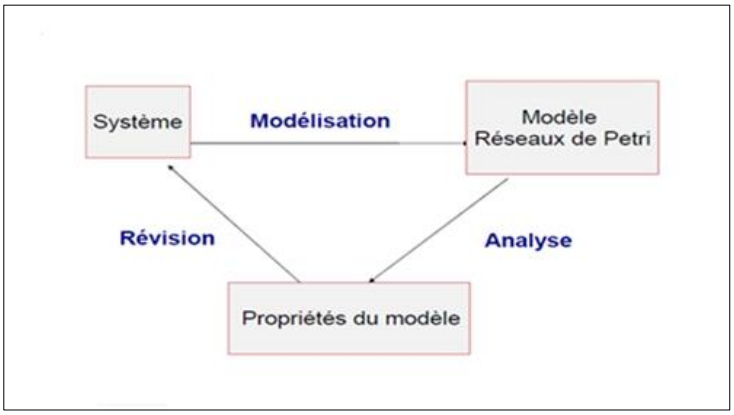


Figure III.1 Méthode générale de modélisation et d'analyse basée sur les réseaux de Petri **[26].**

1. Réseau de Petri

2.1. Concept et définitions

Un réseau de Petri est un graphe orienté, pondéré et biparti, c'est à dire comprenant deux types de nœuds : les places représentées par des cercles qui contiennent les données appelées jetons et des transitions par des rectangles qui indiquent les calculs à réaliser avec les jetons **[26].** Les arcs peuvent relier une place à une transition ou une transition à une place (pas d'arcs reliant les places ni d'arcs entre transitions)[**27].**

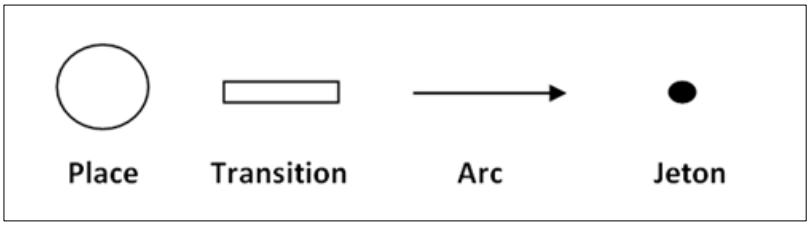


Figure III.2 Les composants graphiques d'un RDP

- **Condition :**

L'état du système peut être vu comme un ensemble de conditions qui sont de ce fait, des prédicats ou des descriptions logiques d'un état du système. Une condition est soit vraie soit fausse**[26].**

- **Evénement :**

Les actions se déroulant dans le système définissent les événements. L'état du système conditionne le déclenchement d'un événement**[26].**

- **Déclenchement, pré-condition, post-condition :**

Les pré-conditions de l'événement sont les conditions nécessaires au déclenchement de celui-ci. Quand un événement se produit, certaines de ses pré-conditions peuvent cesser d'être vraies alors que d'autres conditions (post-conditions de l'événement), deviennent vraies**[26].**

2.1.1. Définition formelle

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

- O`u P = (P1, P2,..., Pm) un ensemble fini de places.
- T = (T1, T2, ..., Tn) un ensemble fini de transitions, disjoint de P.
- Pre : P × T ! l'application incidence avant (ensemble des entiers naturels).
- Post : P × T ! l'application incidence.

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**[28].**

À propos des places :

- « p » est une place d'entrée de la transition « t » si Pré (p, t) > 0
- « p » est une place de sortie de la transition « t » si Post (p, t) > 0 **[29][30].**

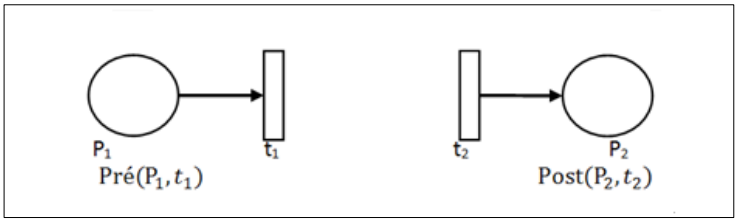


Figure III.3 Représentation des éléments du réseau de Petri

2.1.2. Marquage

Chaque place contient un nombre entier positif ou nul de marques ou jetons. Le marquage M définit l'état du système décrit par le réseau à un instant donné. C'est un vecteur colonne de dimension le nombre de places dans le réseau. Le i^{eme} élément du vecteur correspond au nombre de jetons contenus dans la place P_i **[31].**

Dans la figure III.4 représente réseau de petri :

- ✓ T = $\{T_1, T_2, T_3, T_4\}$
- ✓ P = $\{P_1, P_2, P_3, P_4, P_5\}$
- ✓ Arcs = $\{(P_1, T_1), (T_1, P_2), (T_1, P_4), (P_2, T_2), (P_4, T_3), (T_2, P_3), (T_3, P_5), (P_3, T_4), (P_5, T_4), (T_4, P_1)\}$
- ✓ Le marquage initial $\boldsymbol{M_0}[\mathbf{1\ 0\ 0\ 0}]^T$

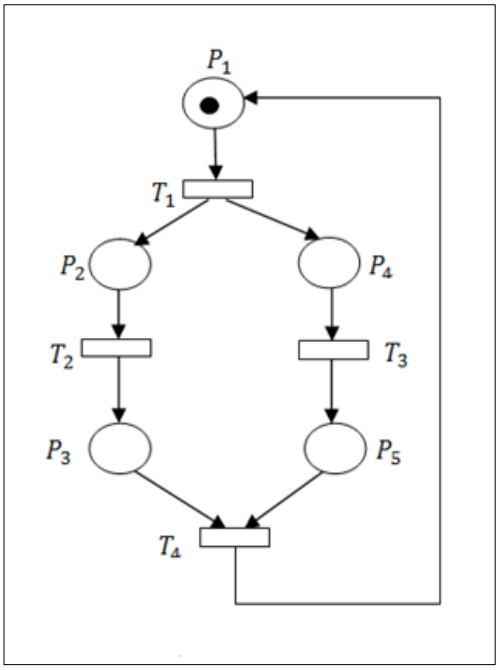


Figure III.4 Exemple de réseau de Petri

2.1.3. Représentation matricielle

Il est facile de construire une représentation matricielle d'un RdP PT. On construit la matrice correspondant à Pré et celle correspondant à Post. Les lignes identifient les places et les colonnes, les transitions. Une intersection (i,j), dans Pré, correspond au coût associé à l'arc menant de la place « i » à la transition « j », tandis qu'une intersection (i,j), dans Post, correspond au coût associé à l'arc menant de la transition « j » à la place « i »**[29][30].** Il est défini par la matrice d'incidence du RdP $\boldsymbol{C}$: $\boldsymbol{C = post - pré}$

La matrice incidence pour le réseau de Petri précédent (Figure III.4) est :

Pré =

	T_1	T_2	T_3	T_4
P_1	1	0	0	0
P_2	0	1	0	0
P_3	0	0	0	1
P_4	0	0	1	0
P_5	0	0	0	1

Post =

	T_1	T_2	T_3	T_4
P_1	0	0	0	1
P_2	1	0	0	0
P_3	0	1	0	0
P_4	1	0	0	0
P_5	0	0	1	0

C =

	T_1	T_2	T_3	T_4
P_1	-1	0	0	1
P_2	1	-1	0	0
P_3	0	1	0	-1
P_4	1	0	-1	0
P_5	0	0	1	-1

2.2. Dynamique des RdP

2.2.1. 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 (Figure III.5). Le franchissement des transitions et les modifications de marquage qu'il entraîne permettent d'analyser la dynamique du système modéliser **[28].** Une transition *t* est sensibilisée pour un marquage *M* si et seulement si ∀ *p* ∈ *P* : *Pi* le franchissement de *t* conduit au nouveau marquage M' tel que :

∀ *p* ∈ *P M*0(*p*)= *M*(*p*) - *Pre*(*p*, *t*) + *Post*(*p*, *t*) .

La différence des marquages *M* et *M*0 est égale à *Post*-*Pre*. On introduit la matrice W, dite matrice d'incidence telle que :

∀ *p* ∈*P*, ∀ *t* ∈ *T* : *W*(*p*, *t*)= *Post*(*p*, *t*) - *Pre*(*p*, *t*) .

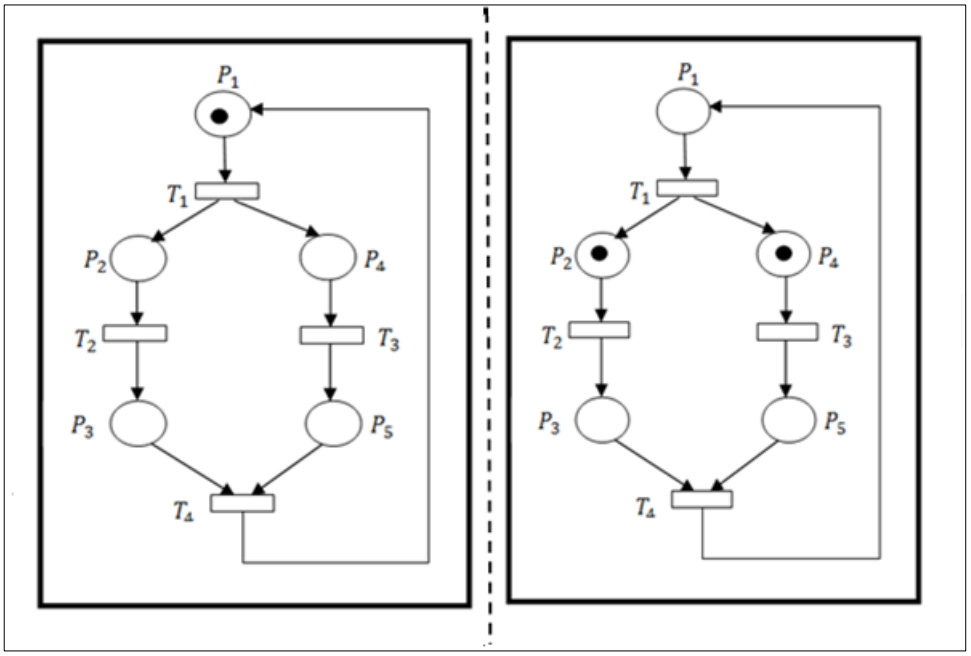


Figure III.5 Franchissement d'une transition

2.2.2. Séquence de franchissements

Une séquence de franchissement S est une suite de transitions T_i $T_j \ldots T_k$ qui peuvent être franchies successivement à partir d'un marquage donné. Une seule transition peut être franchie à la fois.

On note : $M_i[s \rightarrow M_j$ ou $M_i[s > M_j$: à partir du marquage M_i , le franchissement de la séquence S aboutit au marquage M_j**[31].**

2.2.3. Marquages accessibles

L'ensemble des marquages accessibles est l'ensemble des marquages Mi qui peuvent être atteint par le franchissement d'une séquence S à partir du marquage initial M_0.

On le note $*M_0$ **[31].**

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

Notation : On notera par *M l'ensemble des marquages accessibles à partir du marquage M. Par s la séquence de transitions $t_1, t_2, \ldots, t_n$ et le passage du marquage M au marquage M' par franchissement de la séquence de transitions s par $M \xrightarrow{s} M'$**[32].**

2.2.4. 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).un arc orienté relie deux sommet M_i et M_j s'il existe une transition t franchissable permettant de passer d'un marquage à un autre M_i [$t > M_j$]. Les arcs du graphe sont étiqueté par les transitions correspondantes**[32].** La figure III.6 représente le graphe de marquages accessibles Pour le réseau de Petri de la figure III.4

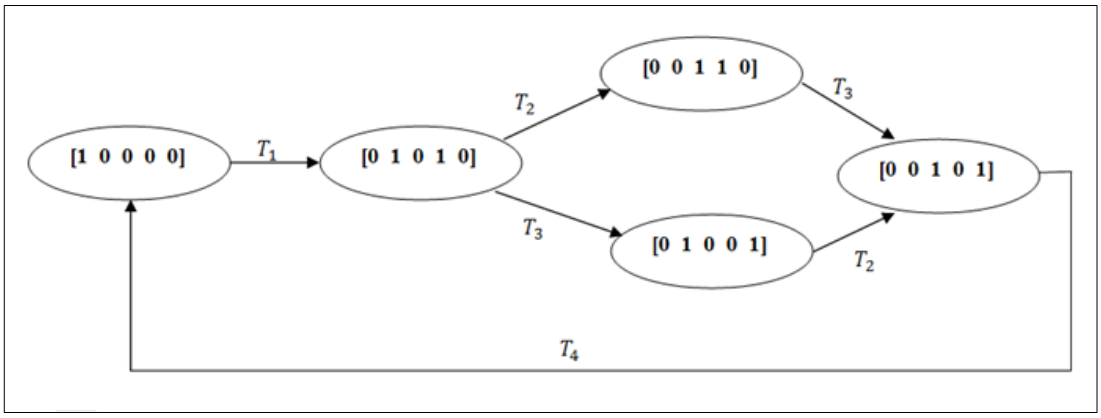


Figure III.6 Graphe de marquages accessibles

Remarque : Le graphe peut être fini ou infini. Dans le cas ou il est fini, il sera complet lorsque toutes les possibilités sont représentées.

2.3. Propriétés des réseaux de petri

2.3.1. Caractère borné

Cette propriété définit et caractérise la possibilité pour une place d'accumuler une quantité bornée ou pas de jetons au cours de l'évolution d'un réseau**[33].**

2.3.1.1. Place k-bornée

Pour un réseau R et un marquage M_0, une place « p » du réseau marqué (R, M_0) est k-bornée si pour tout marquage M accessible depuis M_0, M(p) ≤ k. Dans le cas contraire la place « p » est dite non-bornée. Autrement dit **[29][30][34]:**

p est k-bornée ↔ ∀ M ∈ A(R, M_0), M(p) ≤ k

2.3.1.2. Réseau borné

Un réseau marqué est borné si toutes ses places sont bornées**[29][30][34].**

Dans la figure(III.7) le réseau non borné car croissance infinie des jetons dans la place P2.

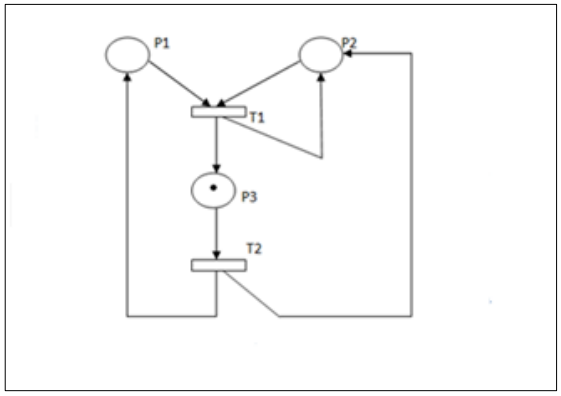


Figure III.7 Exemple d'un RdP non-bornée

Dans la figure III.8 le réseau est 3-borné puisque toutes ses places sont 3-bornées

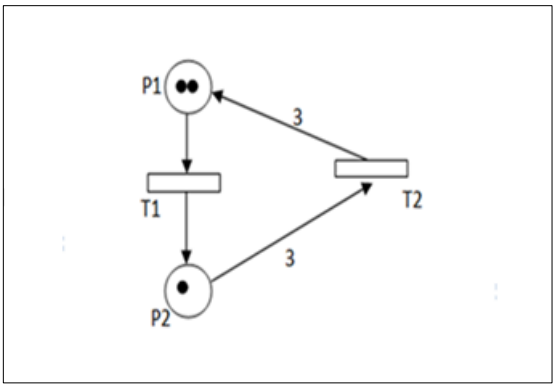


Figure III.8 Exemple d'un RdP 3-bornée

Remarque : Les réseaux **1-bornés** sont appelés des réseaux **saufs**.

2.3.2. Vivacité

La vérification de la vivacité correspond à vérifier qu'à tout instant le système conserve la possibilité de reproduire toutes les actions**[35].**

2.3.2.1. Transition quasi-vivante

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 transitions sont quasi vivantes**[36].**

Dans la figure III.9 La transition t1 est quasi vivante pour le marquage initial M_0. t1 est tirable au moins une fois

2.3.2.2. 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**[36].**

Dans la figure III.9 t2 et $t3$ sont des transitions vivantes

2.3.2.3. 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 **[28].**

Donc de la figure III.9 RdP n'est pas vivant (à un certain moment, il n'est plus possible de tirer t_1)

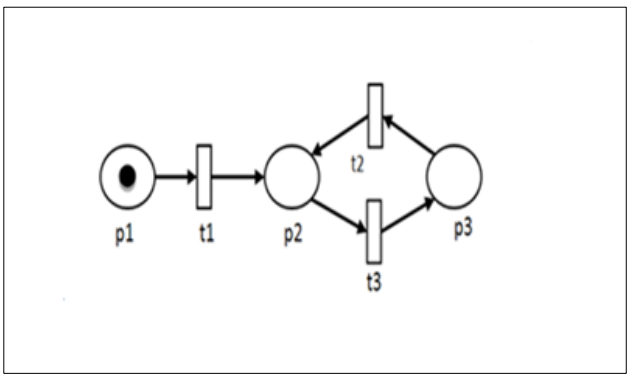


Figure III.9 Représentation des transitions vivantes et quasi-vivantes

2.3.3. Blocage

Un blocage est un marquage tel qu'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**[36]**.

Dans la figure III.10 il y a un RdP avec blocage dans M_1

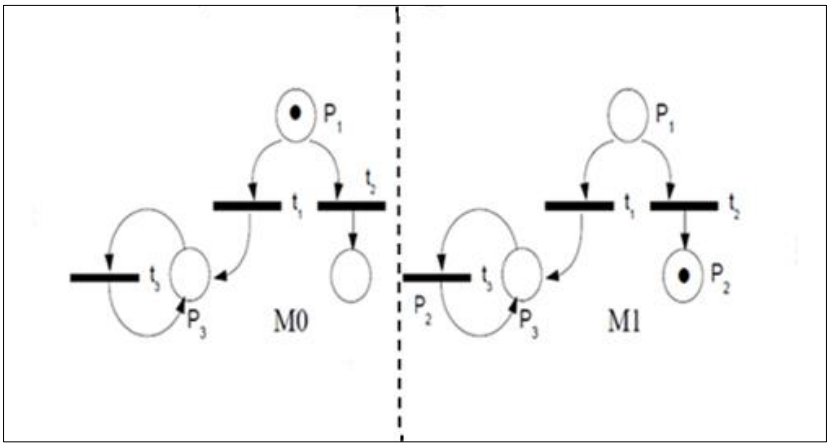


Figure III.10 exemple d'un RdP avec blocage

3. Réseaux de Petri Récursifs

Le réseau de Petri Récursif (RDPR) a la même structure que Réseau de Petri sauf que les transitions sont divisées en deux catégories **[37]** élémentaire et abstraite.

- **Transition élémentaire** (T_{el}) : représentée par un rectangle simple comme cela des transitions de réseaux de Petri qu'on a vu. Son franchissement entraine l'exécution de l'action élémentaire qui lui est associée.
- **Transition abstraire** (T_{abs}): représentée par un rectangle à double bord. Son franchissement donne lieu à l'activation d'un RdPR.

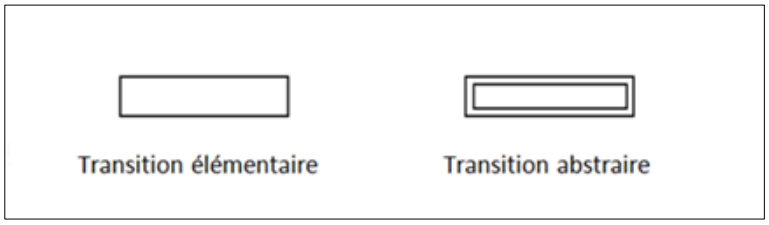


Figure III.11 Représentation des Transitions de RdPR

3.1. Définition

Un RdPR est un septuple R = $< P , T , I , W^{-} , W^{+} , \omega , \gamma >$ où

- P : est un ensemble fini de place.
- T : est un ensemble fini de transition où $P \cap T = \emptyset$, Une transition de T peut être élémentaire (T_{el}) ou abstraite(T_{abs}).
- I : est un ensemble fini d'indices, généralement appelé les indices de terminaison.
- W^{-} : est la fonction de Pré-conditions.
- W^{+} : est la fonction de Post-conditions.
- ω : est la fonction du marquage initial définie T_{abs} à $\square^{p}$, elle associe à chaque transition abstraite T_{abs} un marquage ordinaire.
- γ : est une famille indexée de représentation effective d'ensembles semi-linéaires de marquages finaux.

La figure III.13 montre les composantes graphiques d'un RdPR qui sont :

- ✓ $\{P_0, P_1, P_2, P_3, P'_0, P'_1, P'_2, P'_3\}$: Ensembles des places.
- ✓ t_1, t'_1, t'_2 : Transition élémentaire.
- ✓ t_2 : Transition abstraite.
- ✓ <0> : L'indice de terminaison.
- ✓ $\boxed{P'_0 + P'_2}$: Marquage initiale.
- ✓ γ_0 : une famille indexée de représentation effective d'ensemble semi-linéaire de marquages finaux.

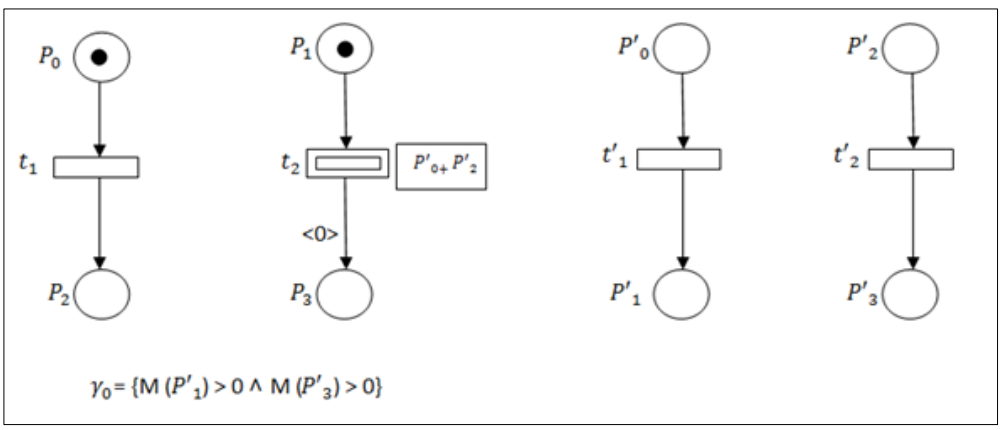


Figure III.12 Exemple d'un RdPR

3.2. Marqua

Le marquage étendu d'un RdPR est un arbre marqué $T_r = < V, M, E, A >$ où :

- V est l'ensemble fini des nœuds de l'arbre. Lorsque l'arbre T_r n'est pas vide, $V_0 \in$ V dénote la racine de l'arbre.
- M : V $\rightarrow \mathbb{N}^p$ est une application qui associe à chaque nœud un marquage ordinaire.
- $E \subseteq$ V x V est l'ensemble des arcs.
- A : E $\rightarrow T_{ab}$ est une application associant une transition abstraire à chaque arc de l'arbre.

La Figure III.13 représente graphe d'accessibilité du RDPR de la figure III.12

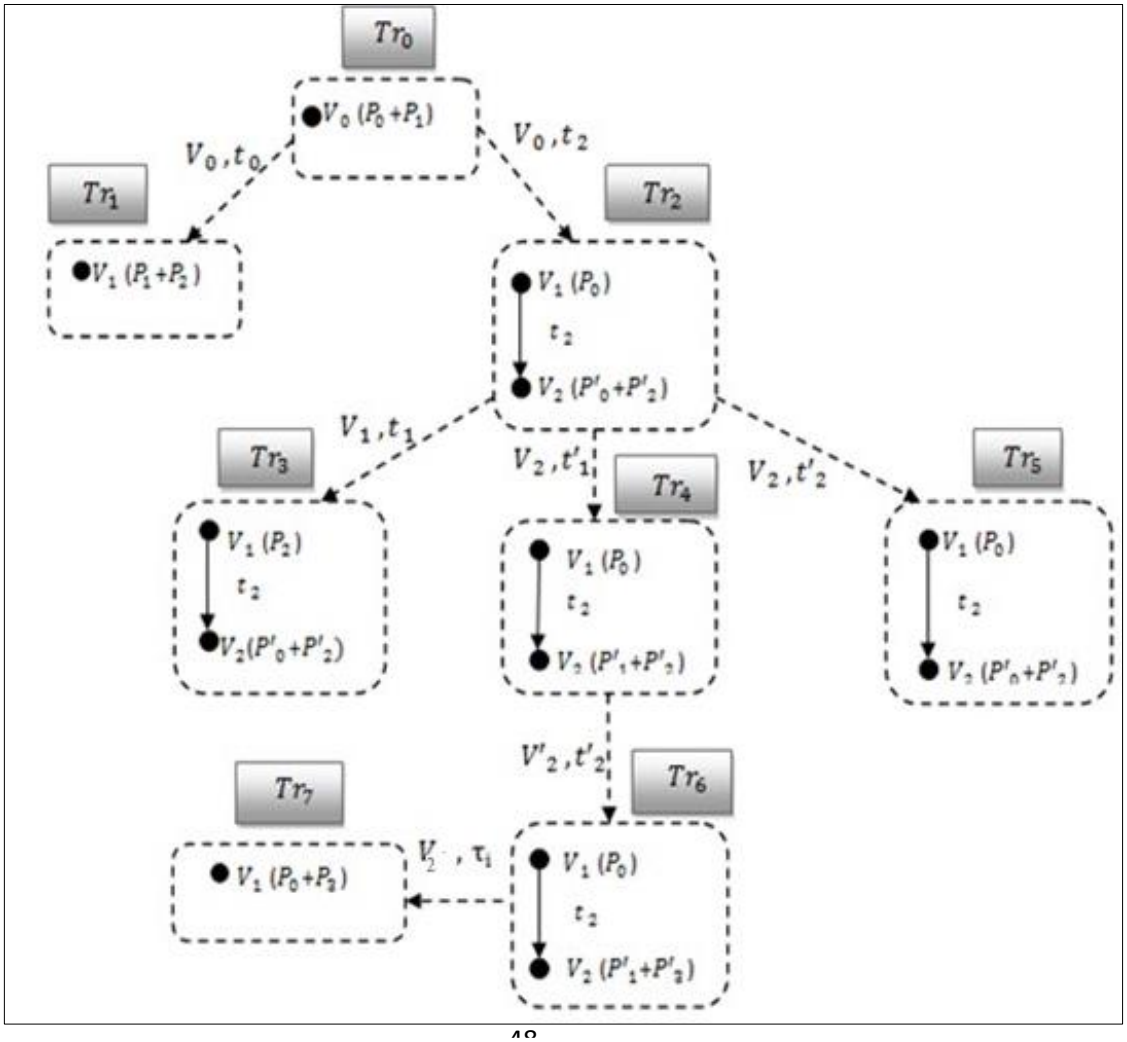


Figure III.13 Graphe d'accessibilité du RdPR

3.3. Franchissabilité et franchissement

3.3.1. Définition

Le franchissement dans un RdPR marqué peut être soit un franchissement d'une transition élémentaire ou abstraite, soit l'exécution d'une étape de coupure (noté τ_i avec $i \in I$).Une transition T est franchissable à partir du nœud v ∈V ssi : M(v) ≥ W^- (.,T)

3.3.2. Franchissement d'une Transition élémentaire

Le franchissement d'une transition élémentaire T_1 dans un nœud V_1 d'un marquage étendu $T_r =< V, M, E, A >$ conduit à un marquage étendu $T_{r'} =< V', M', E', A' >$ (noté $T_r \xrightarrow{t,v} T_{r'}$) tel que :

Pour $E'' = \{(v, v') \in E\}$ et $V'' = \{v' \in V \,|(v, v') \in E''\}$

- $\forall\, p \in P, M(v)(p) \geq W^-(p,t)$
- $V' = V|\,(U_{v' \in v''} Succ(v')), E' = E \cap (v' \times v')$
- $\forall e \in E', A'(e) = A(e)$
- $\forall v' \in V'|\{v\}, M'(v') = M(v')$
- $\forall p \in P|M'(v)(p) = M(v)(p) - W^-(p,t) + W^+(p,t)$

Dans la figure III.14 représente le franchissement d'une transition élémentaire de la figure III.12

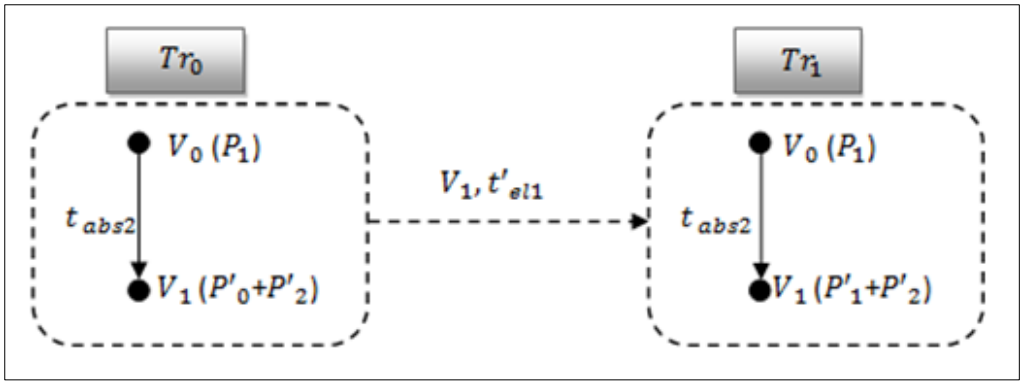


Figure III.14 Franchissement d'une transition élémentaire

3.3.3. Franchissement d'une Transition abstraite

Le franchissement d'une transition abstraite T_1 dans un nœud V_1 d'un marquage étendu $T_r = < V, M, E, A >$ conduit au marquage étendu $T_{r'} = < V', M', E', A' >$ (noté $T_r \xrightarrow{t,v} T_{r'}$) si et seulement si :

- Soit v′ un nouvel identificateur dans l'arbre $T_{r'}$.
- $\forall p \in P, M(v)(p) \geq W^-(p,t)$.
- $V' = V \cup \{v'\}, E' = E \cup \{(v,v')\}, \forall e \in E, A'(e) = A(e), A'((v,v')) = t$.
- $\forall v'' \in V\{v\}, M'(v'') = M(v'')$.
- $\forall p \in P, M'(v)(p) = M(v)(p) - W^-(p,t)$.
- $M'(v') = \Omega(t)$ //le nouveau nœud est associé par le marquage initial de t.

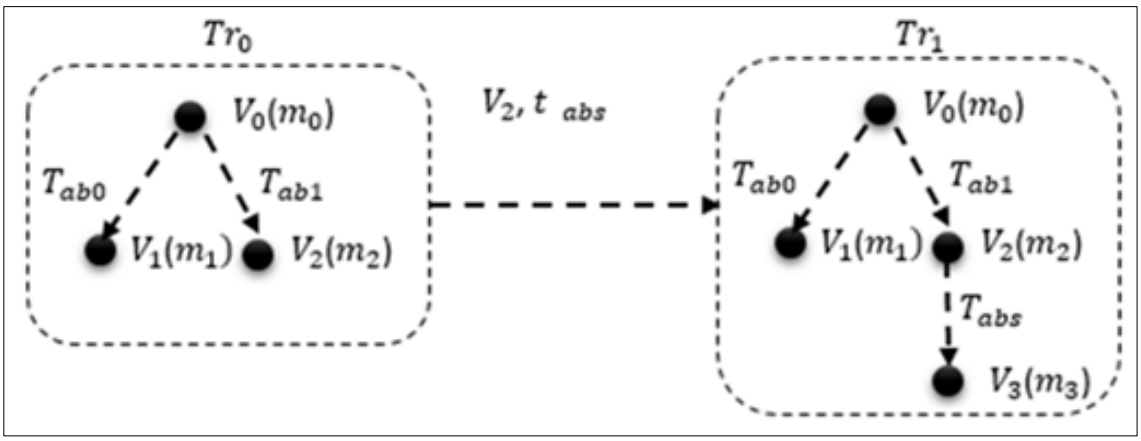


Figure III.15 Franchissement d'une transition abstraite

3.3.4. Franchissement d'une étape de coupure

Un franchissement d'une étape de coupure d'un nœud de l'arbre élimine le sous arbre issu de ce nœud, et produit les jetons en sortie de la transition abstraite qui a généré ce nœud.

Formellement, le franchissement d'une étape de coupure τ_i par un nœud v d'un marquage étendu $T_r = < V, M, E, A >$survient quand $M(v) \in \Upsilon_i$ mène au marquage étendu $T_{r'} = < V', M', E', A' >$(dénoté par$T_r \xrightarrow{\tau_i, v} T_{r'}$) ssi :

- $(v = v_0)$ alors $T_{r'} = \perp$
- $(v \neq v_0)$ alors :
  - $V' = V \setminus Succ(v), E' = E \cap (V' \times V'), \forall e \in E', A'(e) = A(e)$
  - $\forall v' \in V \setminus \{Préd(v)\}, M'(v') = M(v')$
  - $\forall p \in P, M'(Préd(v))(p)\ M(Préd(v))(p) + W^+(p, A(Préd(v), v, i).//$

    i représente l'indice de terminaison associé à Yi.

❖ Succ(v) : l'ensemble de successeurs directs et indirects incluant pour chaque $v_i \in V$.

❖ Préd(v) : l'ensemble de prédécesseurs dans l'arbre quand v_i n'est pas la racine de l'arbre.

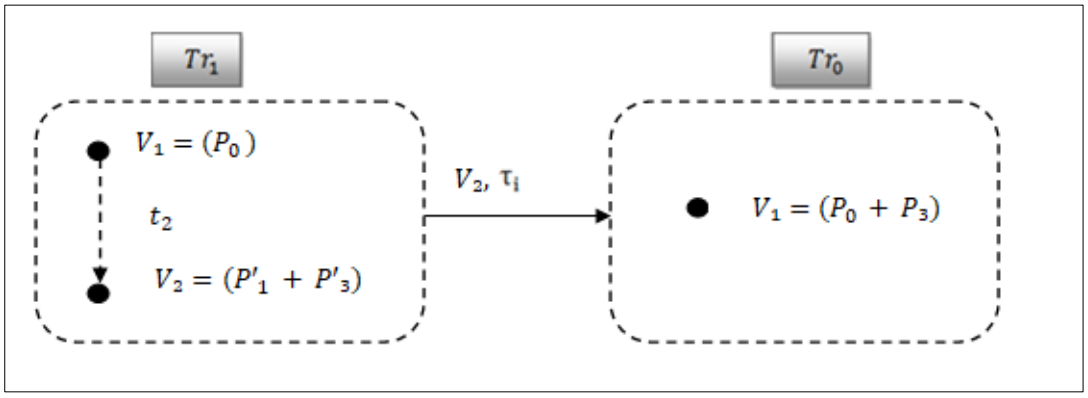


Figure III.16 Franchissement d'une étape de coupure

4. Les RECATNets

Les ECATNets sont des réseaux algébriques qui permettent de décrire de façon compacte des systèmes complexes caractérisés par des structures. La sémantique concurrence est exprimée en termes de logique de réécriture, leur donnant une base mathématique solide**[38].** Les ECATNets récursifs (RECATNets) **[39]** sont une sorte de réseaux de Petri algébriques de haut niveau combinant la puissance expressive des types de données abstraites et des réseaux

récurrents de Petri**[40].** L'utilité du formalisme de RECATNet est: (1) la modélisation et l'analyse d'interactions dans des systèmes distribués ayant une structure dynamique. (2) Permettre de spécifier et d'analyser formellement des systèmes complexes comprenant de nombreux composants avec des types de données complexes, comme c'est le cas pour les diagrammes d'activités complexes comportant des activités composites. (3) Leur sémantique peut être définie en termes de logique de réécriture conditionnelle.

4.1. Syntaxe des RECATNets

Comme dans les autres types de réseaux de Petri, Le réseau d'un ECATNet est représenté par un graphe orienté biparti dont l'ensemble des sommets est $P \cup T$ (Les places étant représentées par des cercles ou des éllipses; et les transitions par des rectangles ou des barres) **[41].**

Chaque place dans un RECATNet est associée à une sorte et les jetons qui y résident sont des multi-ensembles de termes algébriques **[40].**

Les arcs reliant les places et les transitions d'un RECATNet sont étiquetés par des multi-ensembles de termes algébriques construits à partir de la signature multi-sorte et de l'ensemble des variables associées à ce réseau. Comme pour les RECATNets ordinaires, dans un RECATNet un arc entrant (P, T) reliant une place P à une transition T (élémentaire ou abstraite) est étiqueté par les deux inscriptions IC (P, T) (Input Conditions) et DT (P, T) (Destroyed Tokens). L'expression IC (P, T) précise les conditions sur le marquage de la place d'entrée P pour le franchissement de la transition T. L'expression DT (P, T) précise les termes détruits de la place P lorsque la transition T est franchie. Une transition T peut-être aussi étiquetée par une inscription de garde TC(T) (Transition Condition). Cependant à la différence des ECATNets ordinaires, les arcs sortant d'une transition dans un RECATNet sont étiquetés par des expressions différentes selon si cette transition est élémentaire ou abstraite.

Pour une transition élémentaire T, un arc sortant (T,P') la reliant à une place P' est étiquetée par l'inscription CT(P',T) (Created Tokens). Par contre, pour une transition abstraite T un arc sortant (T, P') la reliant à une place P' est étiqueté par une expression $< i >$ ICT (P', T, i) (Indexed Created Tokens). Ces deux expressions précisent les jetons à créer dans la place de sortie P' de la transition t **[40].**

La Figure III.17 montre une représentation graphique d'une transition élémentaire d'un RECATNets.

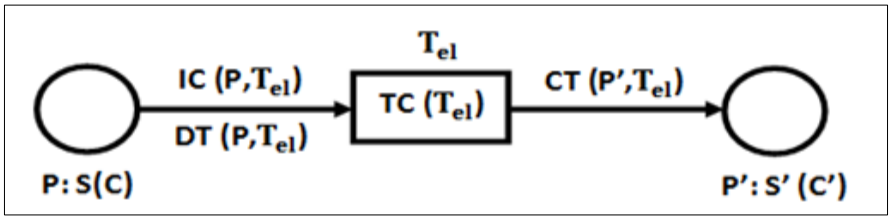


Figure III.17 Représentation graphique d'une transition élémentaire de RECATNets

La Figure III.18 montre une représentation graphique d'une transition abstraite d'un RECATNets.

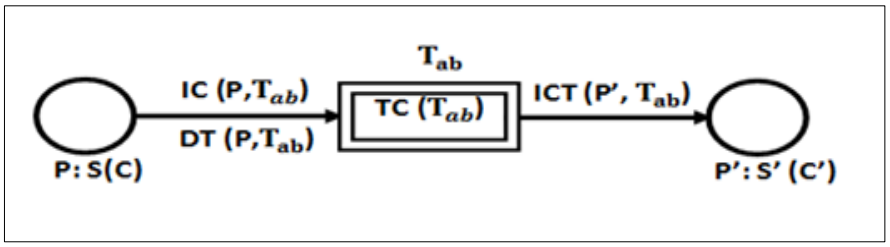


Figure III.18 Représentation graphique d'une transition abstraite de RECATNets

4.2. Définition formelle

Un ECATNet est une paire $E = (Spec, R)$ où :

- $Spec = (\sum, E)$ est une spécification algébrique telle que :
  - ✓ $\sum = (S, Op)$ est une signature
    - S: est l'ensemble de sortes
    - Op: est l'ensemble d'opérations sur S
  - ✓ E: est l'ensemble de S-équations
- R : est un uplet tel que $R = (P, T, S, Cap, IC, DT, CT, TC)$
  - ✓ P: ensemble fini de places
  - ✓ T: ensemble fini de transitions ($P \cap T = \emptyset$)
  - ✓ S : application typage des places
  - ✓ Cap: fonction capacité des places
  - ✓ *DT : fonction jetons détruits (Destroyed Tokens)*

- ✓ *CT : fonction jetons créés (Created Tokens)*
- ✓ *TC : fonction condition de transition (Transition Condition)*
- ✓ *IC* : fonction condition d'entrée (*Input Condition*)

Tableau 4.1 Les différentes formes de l'expression $IC(p,t)$

$IC\ (p,t)$	**Condition de franchissement**
α^0	Le marquage de p doit être égal à α. ($IC(p,t) = \emptyset^0$ signifie que le marquage de p doit être vide)
α^+	Le marquage de la place p doit inclure α. ($IC(p,t) = \emptyset^+$ signifie que la condition est toujours satisfaite).
α^-	Le marquage de la place p ne doit pas inclure α (avec $\alpha \neq \emptyset$)
$\alpha_1 \wedge \alpha_2$	Les conditions α_1 et α_2 sont toutes les deux à vraie
$\alpha_1 \vee \alpha_2$	α_1 ou $\alpha 2_2$ sont à vraie

4.3. Marquage étendu

Le marquage étendu d'un RECATNet est un arbre étiqueté $Tr = < V, M, E, A >$, tel que:

- V est l'ensemble fini des nœuds (processus) de l'arbre. Cette ensemble est vide si Tr = ⊥ (arbre vide).
- M : est l'application qui associe un marquage ordinaire à chaque processus de l'arbre.
- E ⊆ V × V est l'ensemble des arcs de cet arbre.
- A : E → T_{ab} est une application associant une transition abstraite à chaque arc de l'arbre.

Un RECATNet marqué (RN, Tr0) est un RECATNet RN associé à un marquage étendu Tr0 décrivant son état en cours. Pour un marquage étendu Tr, V(Tr) désigne l'ensemble de ses noeuds.

Particulièrement, lorsque l'arbre Tr n'est pas vide, v0(Tr) dénote la racine de l'arbre. Pour chaque noeud $v \in V$, Succ(v) représente l'ensemble de ses nœuds successeurs (directs et indirects) incluant v. De plus, lorsque v n'est pas la racine de l'arbre Tr, Pred(v) désigne son unique nœud prédécesseur (c'est-à-dire, son processus père) **[40].**

4.5. Exemple illustratif

Considérons le RECATNet de la figure III.19 où le processus envoi des requêtes, la requête peut être accepté ou refusé si la requête a été bien envoyé nous obtenons un message (Rq, Accepté) sinon nous obtenons une réponse (Rq, Rejeté).les termes Rq_1et Rq_2 sont des constantes de sorte Request et les termes $Ok, No\ Accepté\ et\ Rejeté$ sont des constantes de sorte Data.

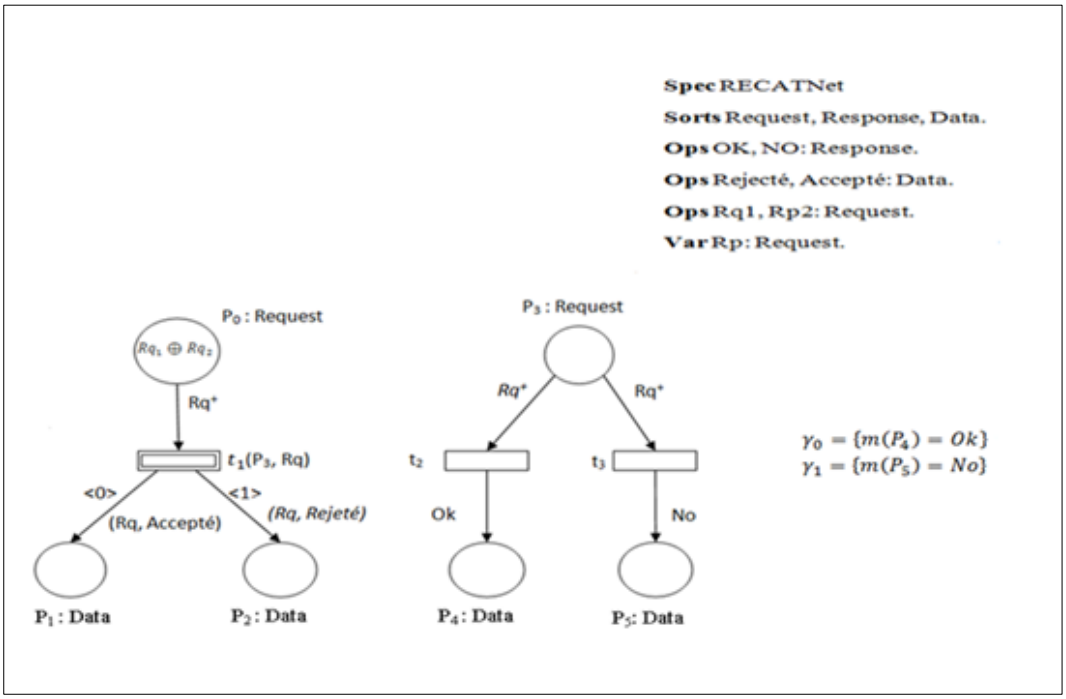


Figure III.19 Exemple de RECATNets

La figure III.20 représente la séquence de franchissement de l'exemple précédent

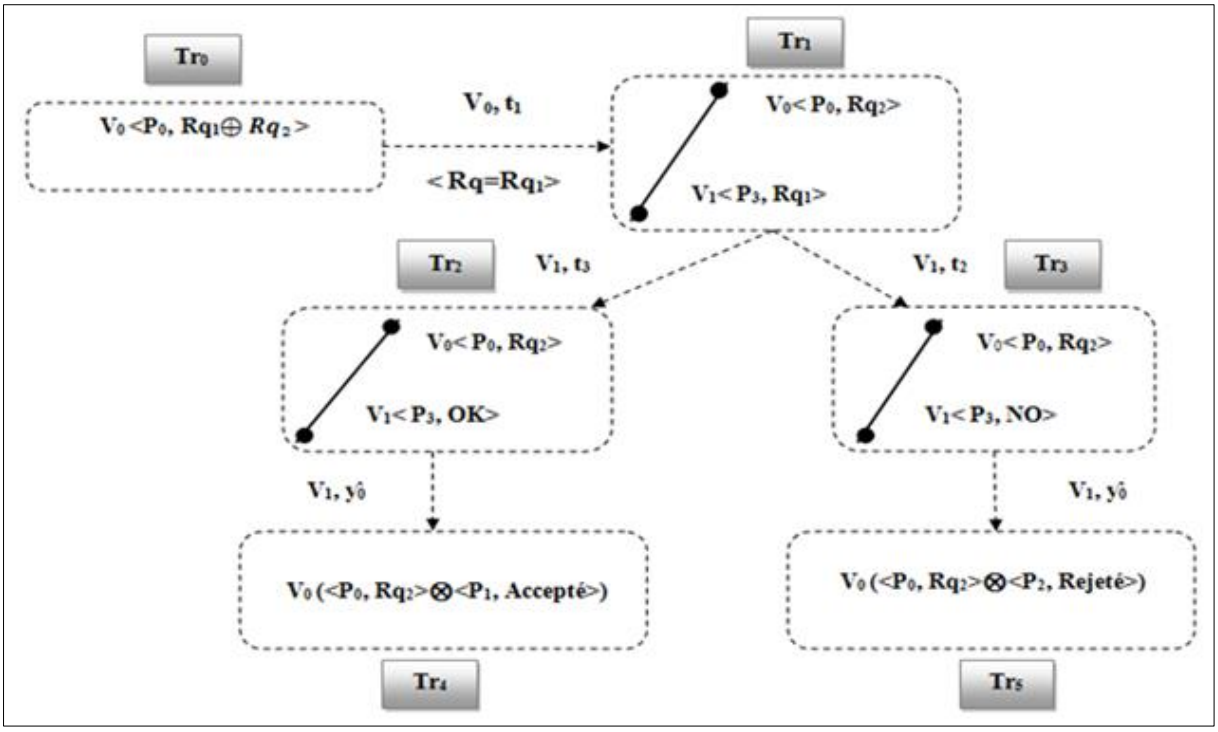


Figure III.20 Séquence de franchissement de l'exemple précédent

5. Méta-modèle RECATNet

Dans l'ingénierie dirigée par les modèles on pourrait dire : il faut modéliser le formalisme de RECATNet. Nous avons défini tous les éléments nécessaires pour construire un modèle RECATNet. La figure représente le méta-modèle de RECATNet.

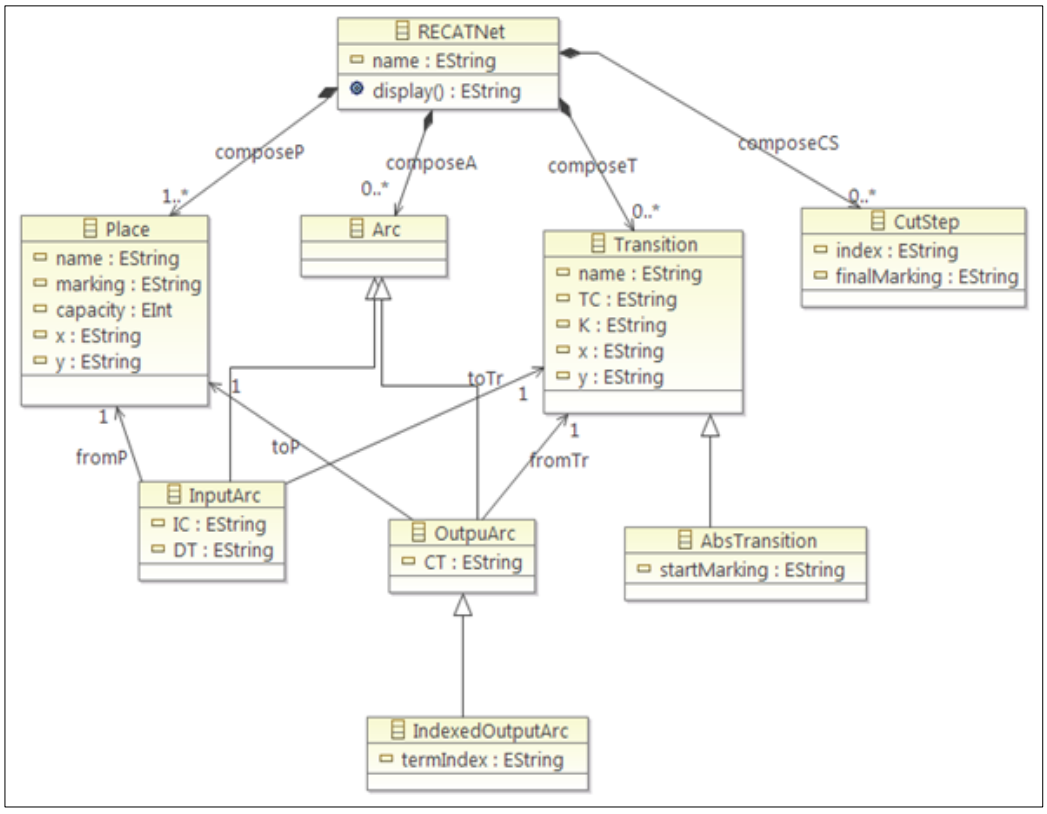


Figure III.21 Le méta-modèle de RECATNet

5.1. Composition de Méta-Modèle

Le tableau 5.1 résume les principales entités de méta-modèle RECATNet et la relation entre ces entités.

Tableau 5.1 Concepts organisationnels du méta-modèle pour le RECATNet

Les entités	Description
RECATNet	C'est une entité racine, elle a un attribut 'name' de type 'string', elle peut posséder une ou plusieurs place, zéro ou plusieurs transitions, zéro ou plusieurs arcs et elle peut comporter À suivre dans la page suivante coupure
Place	Cette entité représente les nœuds places d'un RECATNet, chaque place contient un attribut 'name' et un marquage 'marking' de type 'string' et un attribut 'capacity' de type 'int' qui représente la capacité de la place, et les coordonné de la place dans le graphe résultat de ce projet x et y.
Arc	Cette entité représente la relation entre les places et les transitions, cet arc possède deux sous classe inputArc et outputArc.
InputArc	Cette entité hérite de la classe mère Arc, elle représente les arcs entrants dans les différentes transitions, chaque inputArc sort d'une seule place et entre dans une s eule transition. InputArc contenir deux attributs l'un représente les jetons détruit 'DT' de la place source et deuxième attribut représente

	la condition d'entrée 'IC' de type 'string'. Il est deux relation 'fromP' avec la classe Place et 'toTr' avec la classe Transition.
OutpuArc	Cette entité représente un arc sortant, elle hérite de la classe mère Arc. OutputArc possède un attribut 'CT' de type 'string' qui insérer un jeton dans la place destinataire, chaque OutputArc possède une relation 'fromTr' avec la classe Transition qui indique une seule transition source et une relation 'toP' avec la classe Place qui indique une seule place cible.
IndexedOutputArc	Cette entité hérite de la superclasse OutputArc qui est un sous type de l'arc sortant, elle a un attribut 'termIndex' de type 'string' qui représente indices de terminaison.
Transition	Représente les transitions élémentaires d'un modèle RECATNet, chaque transition contient des attributs 'name' et 'TC' de type 'string' qui représente la condition de la transition, et les coordonné de la place dans le graphe résultat de ce projet x et y, elle est héritée par une autre classe AbsTransition .
AbsTransition	Cette entité représente les transitions abstraites, elle hérite de la classe mère Transition, elle est l'attribut 'startMarking' qui représente le marquage initial des nouveaux processus créé.

CutStep	C'est une coupure qui se fait à la fin d'un processus appelé par une transition abstraite, elle contient l'attribut 'index' de type 'string' qui est de la forme γ_n et qui représente la famille indexé, et l'attribut 'finalMarking' de type 'string' représente le marquage final qui est nécessaire pour réaliser la coupure.

Dans notre travail nous avons ajouté les contraintes OCL sur le méta-modèle cible (RECATNet), on ne peut plus trouver

- ✓ un arc relie une place et une autre place
- ✓ un arc relie une transition et une autre transition

L'implémentation des méta-modèles sous POLARSYS doit être conforme au méta-m éta-modèle ECORE La figure suivante III.22 représente le Méta-Modèle cible sous son extension Ecore.

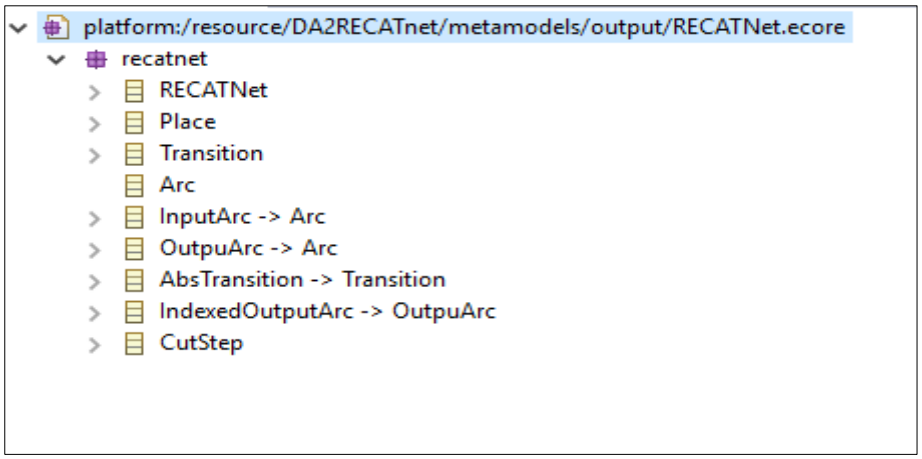


Figure III.22 RECATNet sous extension (ecore)

6. Sémantique de RECATNet en Maude

L'exécution d'un RECATNet génère un arbre dynamique de fils marqués appelé marquage étendu, qui reflète l'état global du RECATNet. Cette dernière dénote la relation de paternité

entre les fils générés (décrivant les appels entre fils). Chacun de ces threads a son propre contexte d'exécution. On donne plus de détails sur les RECATNets concernant le déclenchement des transitions et la construction du graphe d'accessibilité étendue. Par conséquent, l'analyse d'un RECATNet est basée sur la construction de son graphe d'accessibilité étendue, qui est utilisé pour vérifier les propriétés telles que l'accessibilité, l'impasse et la vivacité. De plus, la sémantique de RECATNets peut être définie comme des règles logiques de réécriture exprimées dans le langage Maude**[42].**

6.1 Maude et logique de réécriture

Maude est un langage de haut niveau et un système performant supportant les spécifications exécutables et la programmation déclarative dans la logique de réécriture pour un large éventail d'applications. La logique de réécriture est une logique de changement simultané qui peut naturellement traiter des états et avec des calculs concurrents fortement non déterministes. Un système, sous Maude, est représenté en utilisant la logique équationnelle d'appartenance décrivant son ensemble d'états et un ensemble de règles de réécriture représentant ses transitions d'état. Maude est strictement typé, où les types sont appelés sortes et peuvent être construits hiérarchiquement en utilisant des sous-ensembles. Les énoncés de programmation de base de Maude sont des équations et des règles et ont dans les deux cas une sémantique de réécriture simple dans laquelle les instances du modèle de gauche sont remplacées par des occurrences correspondantes du côté droit**[42].**

7. Conclusion

Dans ce chapitre nous avons commencés par définir le modèle formel « Réseaux de petri », et nous avons vu les concepts de base sur les réseaux de Petri notamment les réseaux de Petri récursifs qui servent à cacher la limitation des réseaux de Pétri.

Ensuit nous avons examiné les notions de base des ECATNets récursive qui sont une sorte de réseaux de Petri algébriques de haut niveau combinant la puissance expressive des types de données abstraites et des réseaux récurrents de Petri.

En fin on a parlé sur la logique de réécriture de mise en œuvre du langage « Maude » pour générer le code de ce dernier à partir du modèle formel RECATNet.

Chapitre IV

Translation des diagrammes d'activité SysML en RECATNet

1. Introduction

Dans ce chapitre est réalisé une transformation des diagrammes d'Activité (DA) vers de RECATNet en respectant les concepts MDA et donc les spécifications définies par l'OMG.la transformation est une opération qui prend en entrée des modèles (input) qui s'appelle source et fournit en sortie des modèles (output) qui s'appelle cible , en général un seul modèle source et un seul modèle cible, et en détailler les règles de transformation

2. Processus de transformation

Notre objectif est d'automatiser la transformation des Diagrammes d'activités (modèle source) vers le RECATNet (modèle cible) et comme nous avons deux formalismes différents, nous somme dans le cas d'une transformation exogène (Voir chapitre II). Donc Il faut établir des règles de transformation pour passer du modèle source au modèle cible. Notre choix s'est porté sur le langage ATL (ATLAS Transformation Language) qui est intégré à la boite d'outils PolarSys d'Eclipse. La démarche générale de transformation est représenté par la figure IV.1 qui montre plus en détail le fonctionnement de la transformation dans l'environnement de travail choisi en vue d'une transformation Diagramme d'activité vers RECATNet. 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 modélisation source : Les Diagrammes d'activité SysML

– Le méta-modèle cible conforme au méta-méta-modèle Ecore qui décrit le langage de modélisation cible : Le RECATNet

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

– Les règles de transformation en ATL.

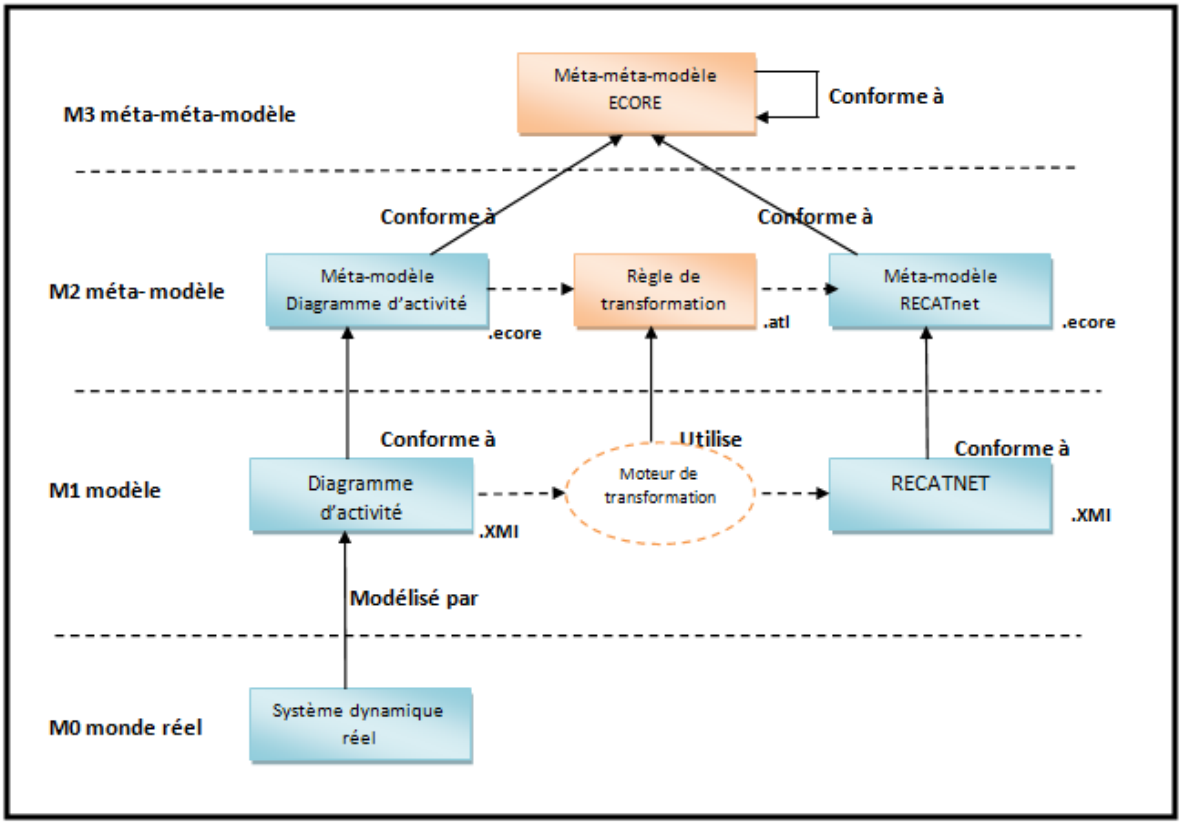


Figure IV.1 Démarche générale de transformation de DA vers RECATNet

2.1. Syntaxe abstraite du diagramme d'activité SysML

Rappelons que le diagramme d'activité SysML est formé par un ensemble d'activités afin de modéliser des comportements complexes des systèmes. En effet, un diagramme d'activité DA est défini par DA=Σ ACT, où ACT représente une activité qui peut être simple (ex : action) ou composée (ex : call behavior).

Une activité ACT est défini par ACT =<N, F, CACT, CALL> où :

- $N = A \cup N^i \cup N^e \cup N^j \cup N^f \cup N^D \cup N^M$ où $A = A^s \cup A^c$ l'ensemble des action (simple A^s et composé A^c) , N^i l'ensemble des nœuds initial , N^e l'ensemble des nœuds final , N^j l'ensemble des nœuds d'union (join node), N^f l'ensemble des nœuds bifurcation (fork node) , N^D l'ensemble des nœuds décision l, N^M l'ensemble des nœuds de fusion (merge node)
- $F \subseteq N \times N$ représente la séquence de flux ,

- CACT est l'ensemble des activités invoquées par l'activité call behavior ,
- CALL : A → CACT est une fonction associée une activité invoquée pour chaque call behavior.

2.2. Les règles de transformation

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 RECATNet.

Afin de simplifier la formalisation des règles de transformation, nous définissons les fonctions suivantes :

- ✓ in(x) : Cette fonction « in(x) = {y ∈ O | (y, x) ∈ F} » retourne les entrées des objets diagramme d'activité d'un objet x de diagramme d'activité.
- ✓ out(x) : Cette fonction « out(x) = {y ∈ O | (x, y) ∈ F} » retourne les soties des objets diagramme d'activité d'un objet x de diagramme d'activité.

Règle N°1 : Transformation de flux de séquence

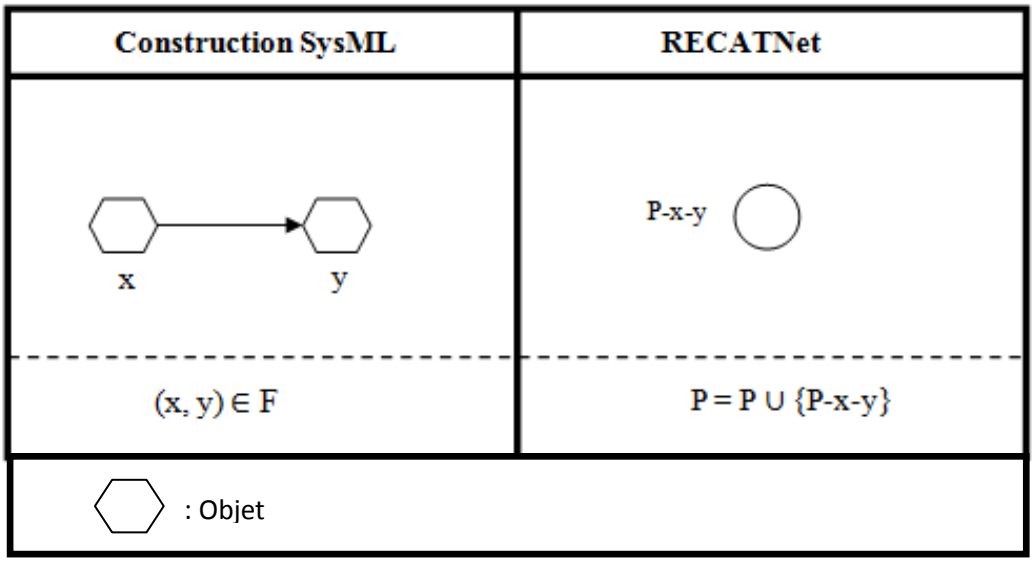


Figure IV.2 Transformation de flux de séquence

La figure IV.2 montre la transformation d'un flux de séquence de SysML, qui est représenté par une flèche entre deux objets *x* et *y*. Alors, la transformation d'un flux de séquence en

RECATNet est une place P-x-y ayant un nom, qui est la concaténation des noms des objets *x* et *y*.

Règle N°2 : Transformation le nœud initial

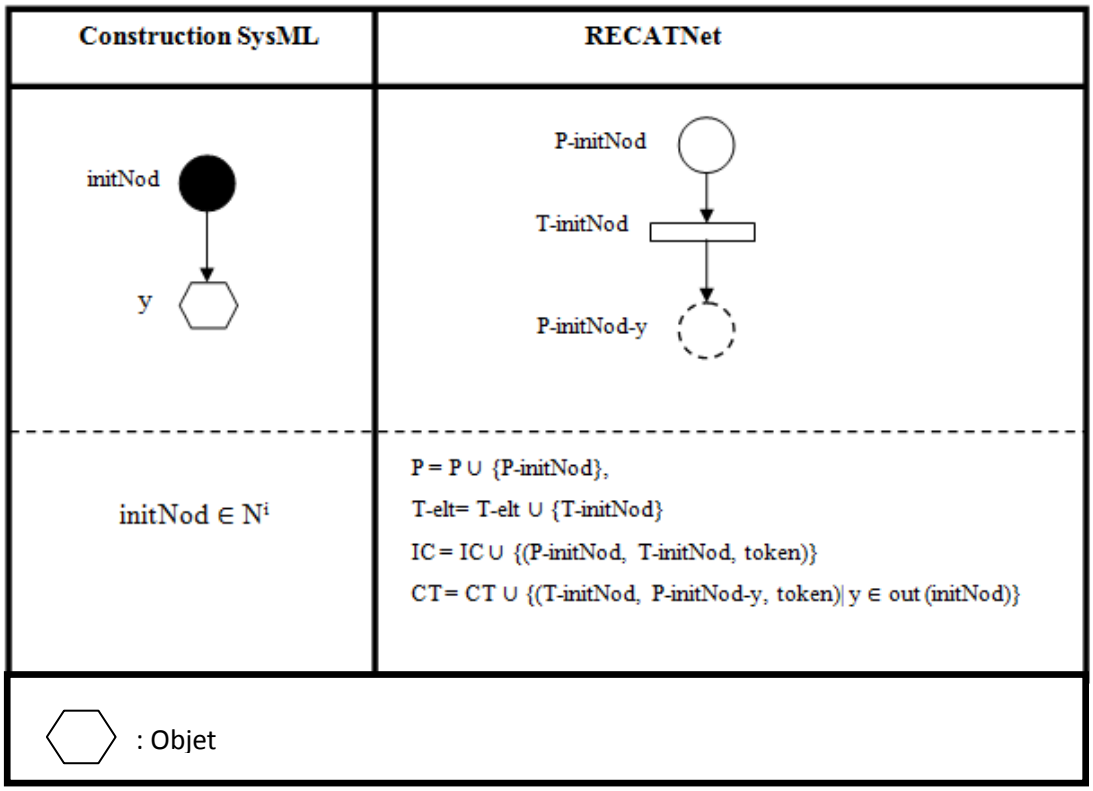


Figure IV.3 Transformation le nœud initial

La figure IV.3 montre le nœud initial représente le point de départ d'une activité, l'objet 'y' est pour définir un objet quelconque pour présenter l'objet suivant de nœud initial. Alors on transforme cet objet vers une transition élémentaire (T-initNod) avec une place d'entrée (P-initNod) et une place de sortie (P-initNod-y) est pour confirmer qu'il existe obligatoirement une place suivante qui présente le 'y' de diagramme d'activité.

Règle N°3 : Transformation le nœud de fin d'activité

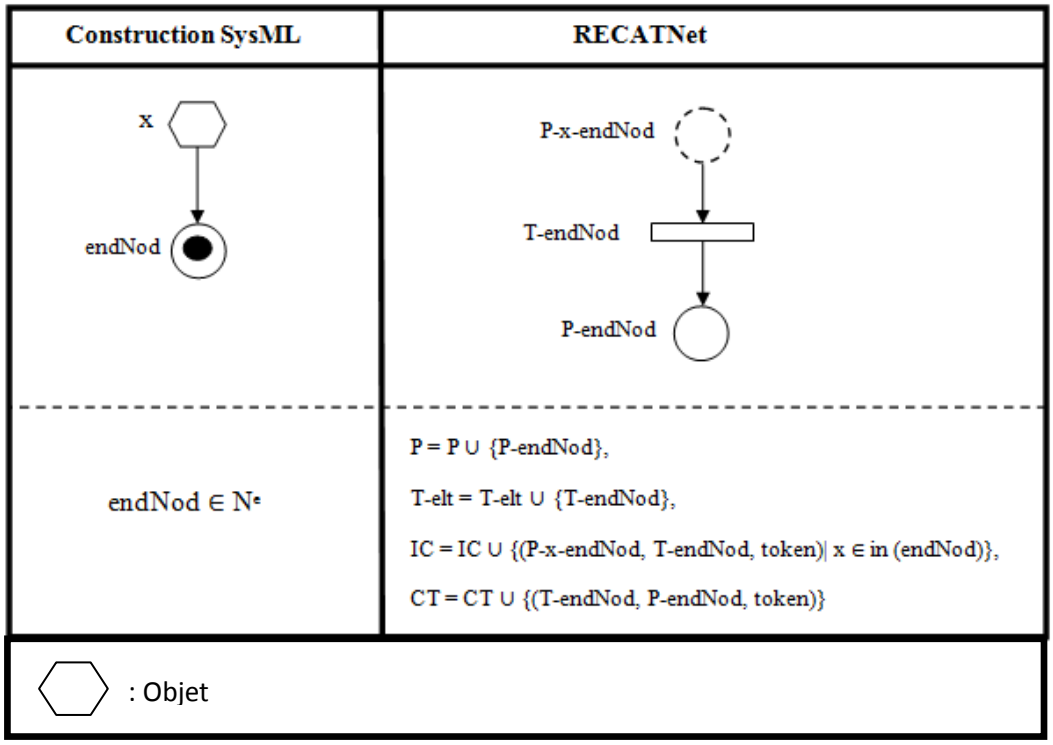


Figure IV.4 Transformation le nœud de fin d'activité

La figure IV.4 montre le nœud de fin d'activité représente le point final d'une activité, et le '*x*' indique qu'il existe un objet précédent. Alors elle est transformée vers une transition élémentaire (T-endNod) avec une place d'entrée (P-x-endNod) et une place de sortie (P-endNod). Le but de la place d'entrée (P-x-endNod) est de confirmer qu'il existe obligatoirement une place entrante.

Règle N°4 : Transformation des actions simples

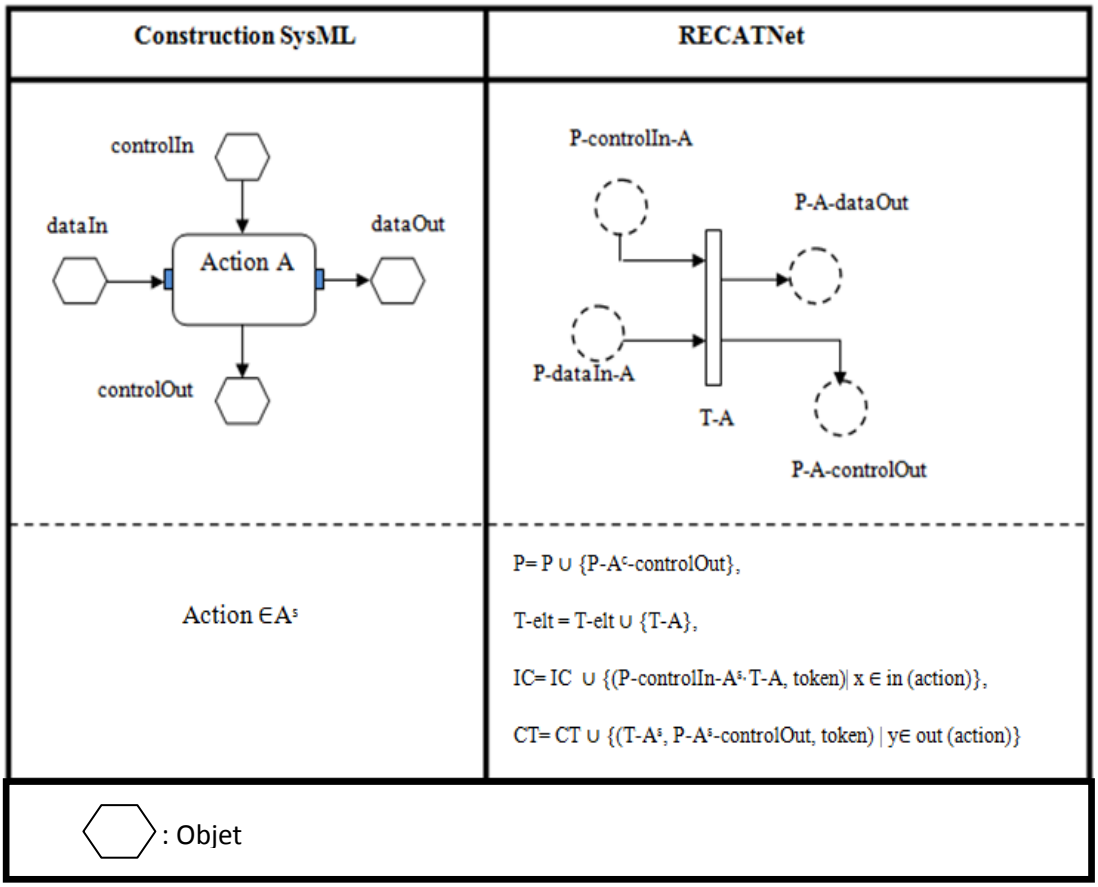


Figure IV.5 Transformation une action simple

La Figure IV.5 montre une action qui liée avec une broche (pin) connecté en entrée ou en sortie d'une activité et une control, ils sont transformer vers une transition élémentaire (T-A) est utilisée pour représenter l'action, et deux places d'entrées (P-dataIn-A), (P-controlIn-A) et deux places de sorties (P-A-dataOut) et (P-A-controlOut) qui peuvent être sous la forme d'une ligne continue ou discontinue.

Règle N°5 : Transformation des flux d'objets

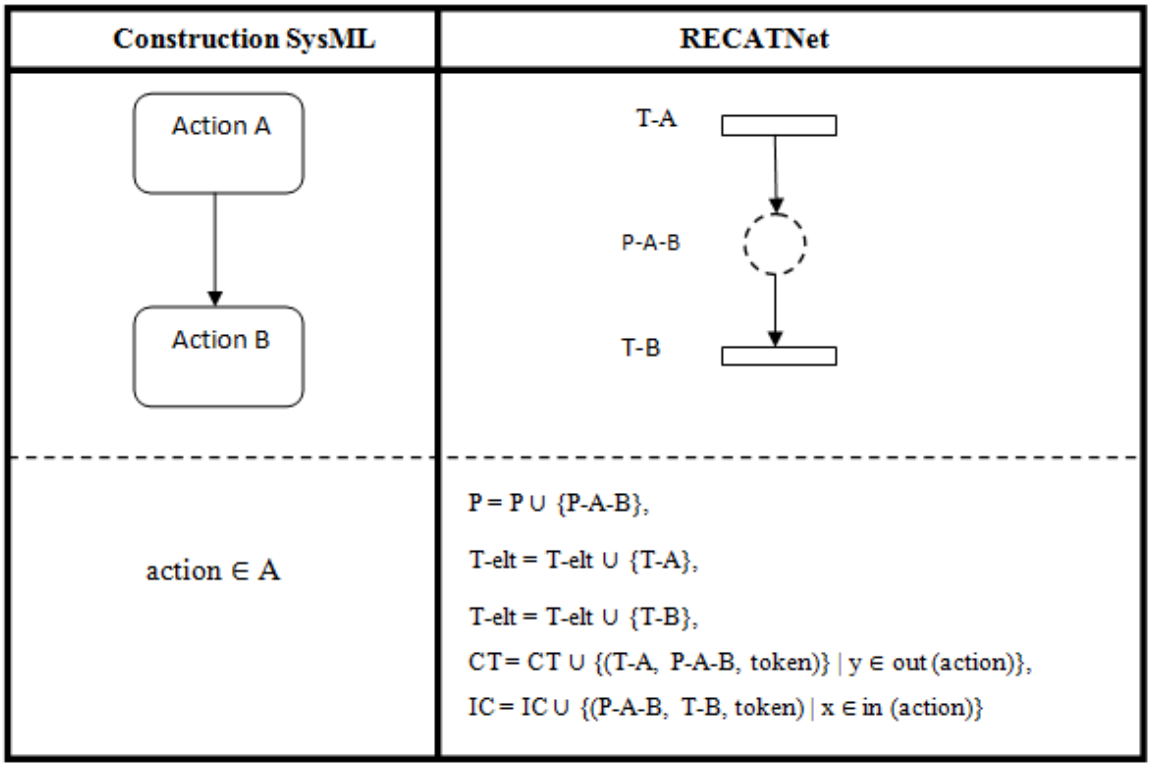


Figure IV.6 Transformation un flux d'objet

La figure IV.6 montre un flux d'objet qui relie la broche de sortie d'une action A à la broche d'entrée d'une autre action B, qui transformer vers une transition élémentaire (T-A) est utilisée pour représenter l'action A et qui transformer vers une transition élémentaire (T-B) est utilisée pour représenter l'action B, on fusionne par une place (P-A-B)

Règle N°6 : Transformation le noeud d'union (join node)

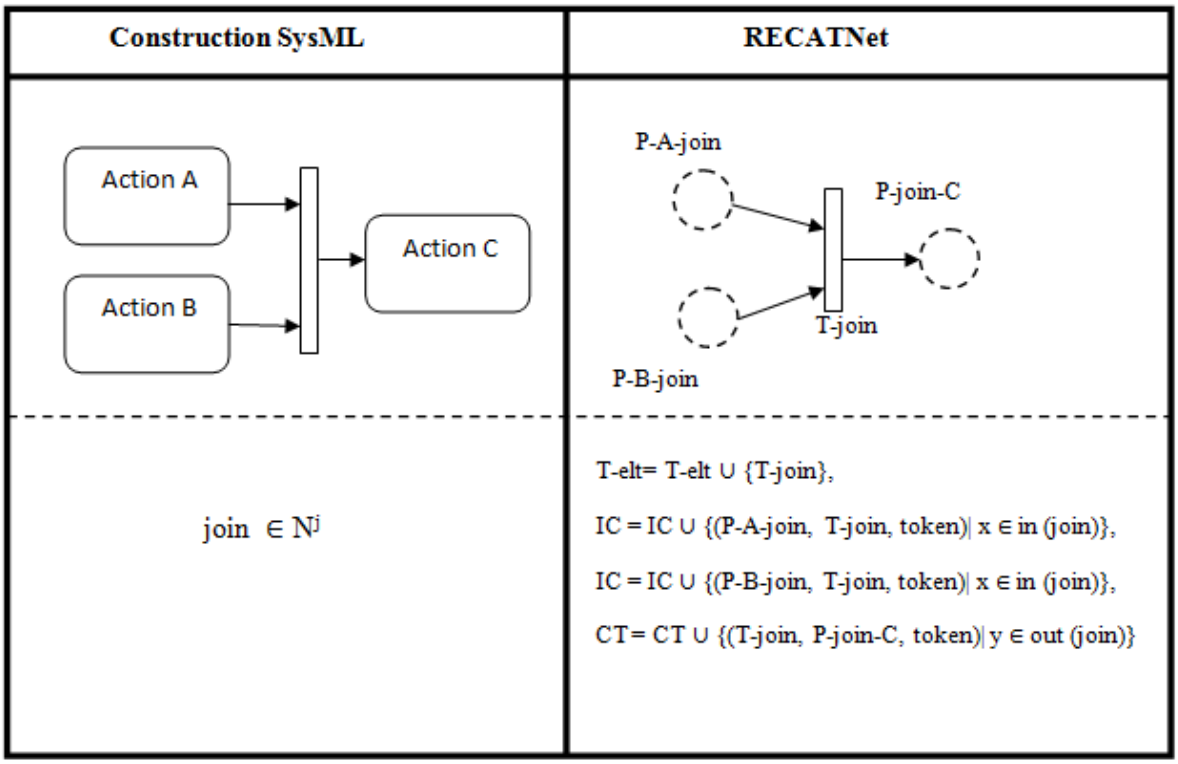


Figure IV.7 Transformation le **noeud d'union (join node)**

La figure IV.7 montre d'un nœud d'union (join node) permettant de combiner deux ou plusieurs d'actions (A, B….) vers une seul action (C), Alors il est transformer vers une transition élémentaire (T-join), l'ensemble des places d'entrée (P-A-join) et (P-join-B) sont pour présenter les actions entrants, et la place de sortie (P-join-C) est pour présenter l'action sortant.

Règle N°7 : Transformation le noeud de bifurcation (fork node)

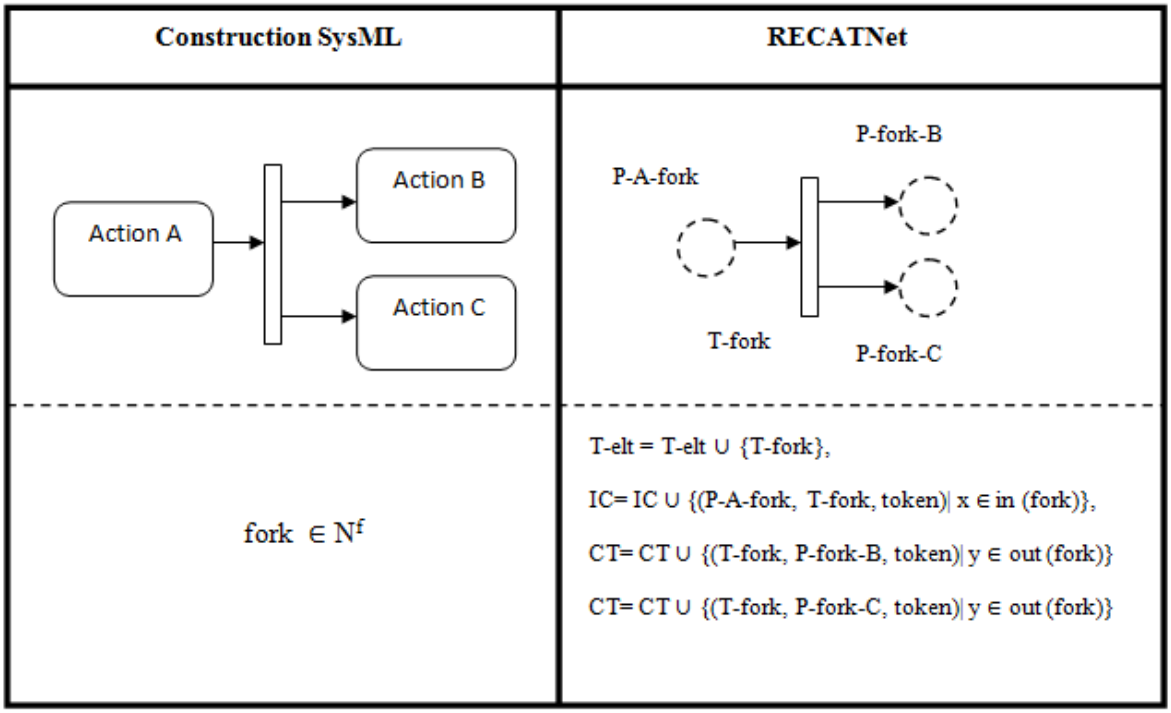


***Figure IV.8 Transformation* le noeud de bifurcation (fork node)**

La figure IV.8 montre d'un nœud de bifurcation (fork node), permettant de diviser une action 'A' en deux ou plusieurs d'actions 'B, C….' qui peuvent être exécutées en parallèle dans le processus. Alors Il est transformé vers une transition élémentaire (T-fork) la place d'entrée (P-A-fork) est pour présenter l'action entrant et l'ensemble des places de sortie (P-fork-B) et (P-fork-C) qui présentent l'ensemble des actions sortants.

Règle N°8 : Transformation des nœuds de décision

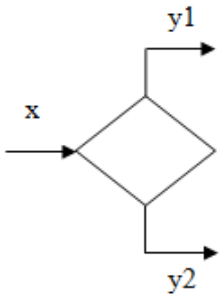
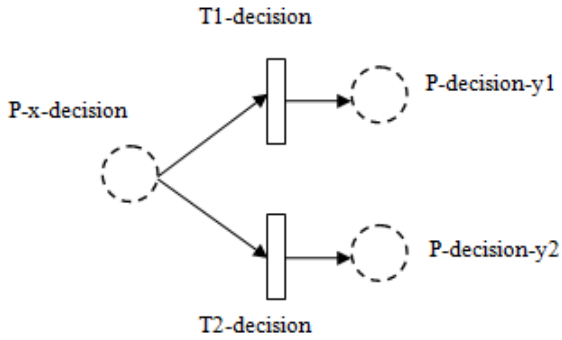
Construction SysML	RECATNet
	
decision ∈ N^D	T-elt = T-elt ∪ {T1-decision, T2-decision | y ∈ out (decision)}, TC = TC ∪ {(T1-decision, Cond (decision, y)) | y ∈ out (decision)}, IC = IC ∪ (P-x-decision, T-decision, token) | y ∈ out (decision) ∧ x ∈ in (decision)}, CT = CT ∪ { T-decision, P-decision-y, token) | y ∈ out (decision)}

Figure IV.9 Transformation le nœud de décision

La figure IV.9 montre le nœud de décision il est utilisé pour créer des chemins alternatifs au sein d'un flux de processus. Le '*x*' présente le chemin entrant et les y (y1, y2..) présentent les chemins de sortie qui ont des conditions à vérifier pour utiliser le chemin, Alors Il est transformé vers un ensemble des transitions élémentaires (T1-decision, T2-decision) et chaque transition a une place sortante (P-decision-y) est pour présenter les flux conditionné, et une place d'entrée (P-x-decision) qui présente le flux entrant '*x*'. Chaque transition (T1-decision, T2-decision) on s'unifié une condition de franchissement (TC).

Règle N°9 : Transformation des nœuds de fusion (merge node)

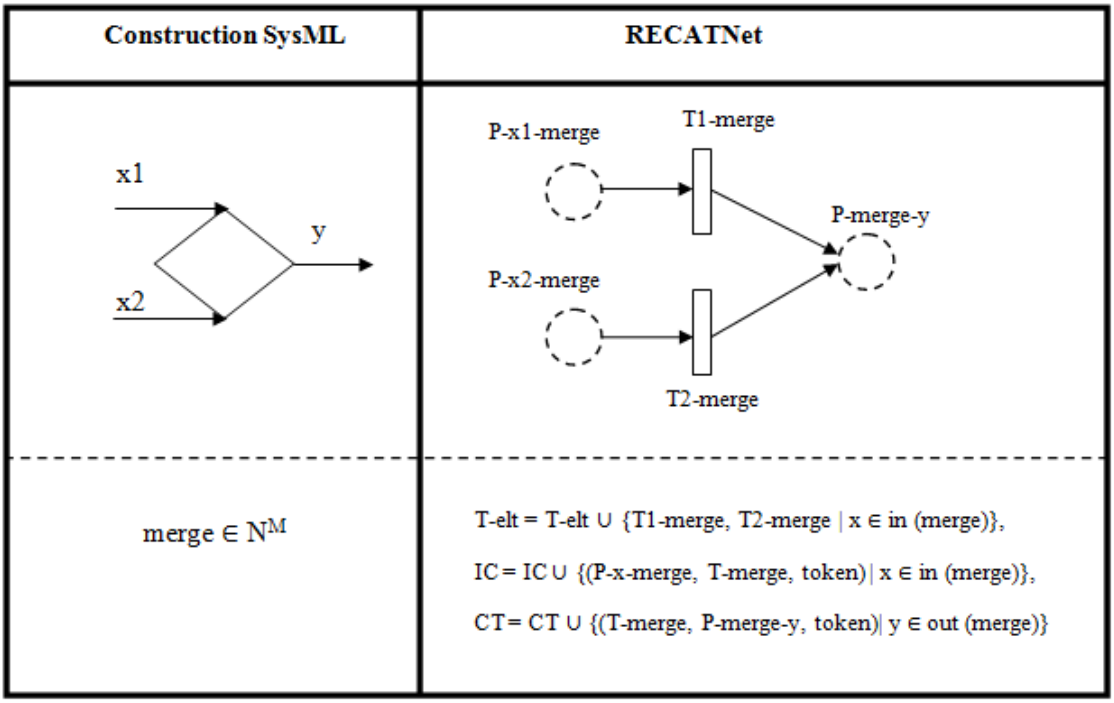


***Figure IV.10 Transformation le* noeud de fusion (merge node)**

La figure IV.10 montre le noeud de fusion (merge node), permettant de combiner deux ou plusieurs flux de séquence parallèles (x1, x2..) en un seul flux de séquence *y*, Alors Il est transformé vers un ensemble des transitions élémentaires (T1-merge, T2-merge) avec une place de sortie (P-merge-y), chaque transition a une place entrante pour présenter les flux entrantes de diagramme d'activité.

Règle N°10 : Transformation des actions de comportement des appels (call behavior actions)

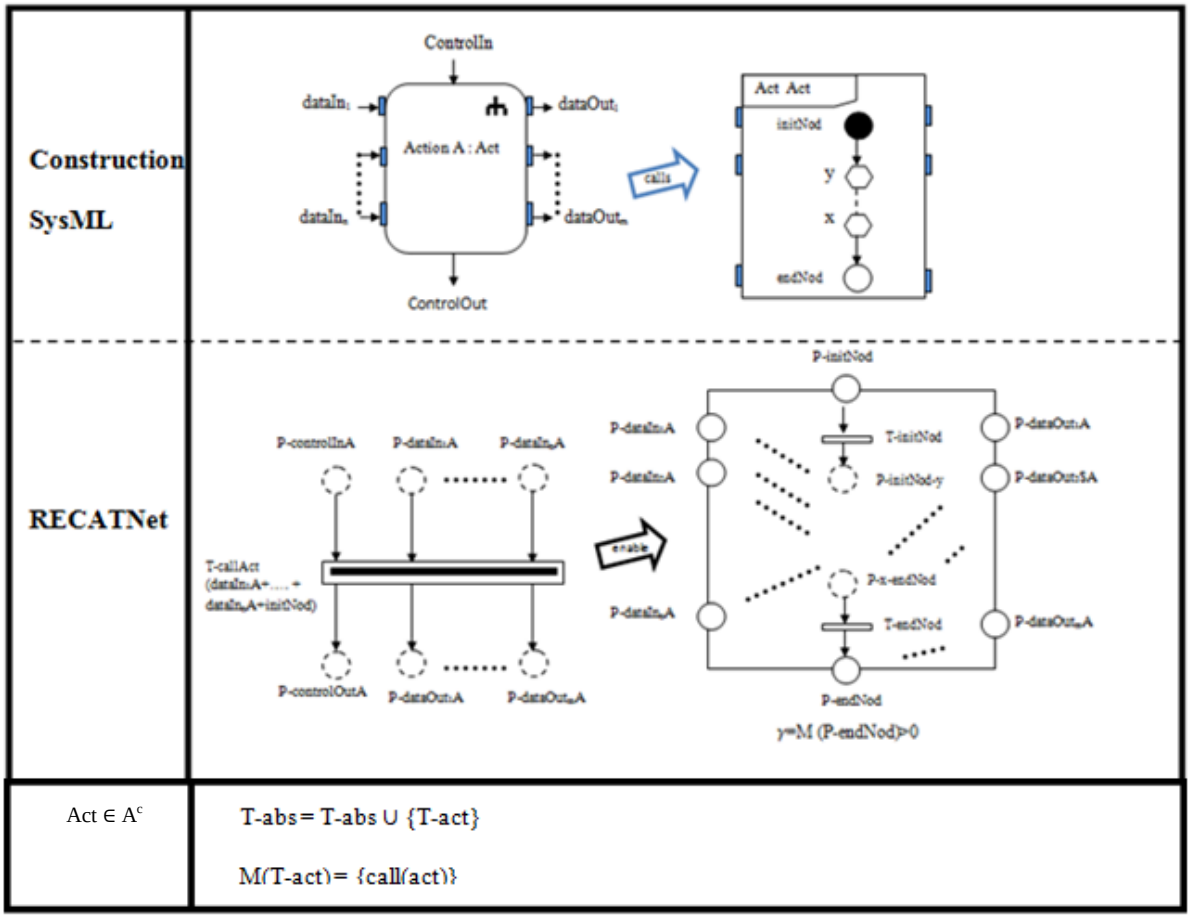


Figure IV.11 Transformation le comportement d'appel (call behavior)

Pour mapper une action de comportement d'appel (call behavior), nous devons considérer trois étapes principales:

- ✓ Passer les flux d'entrées de l'action de comportement d'appel à l'activité appeler.
- ✓ Exécute l'activité appelée.
- ✓ Passer les flux de sortie de l'activité appelée à l'action de comportement d'appel.

La figure IV.11 montre pour mappage d'une action A de comportement d'appel (call behavior) qui appelle une activité Act avec un flux de control d'entrée et de sortie, n broches d'entrée ($dataIn_1$, $dataIn_2$, ... $dataIn_n$) et m broches de sortie ($dataOut_1$, $dataOut_2$, ... $dataOut_m$).
Tous les flux d'entrée et de sortie sont représentés par des places RECATNet. L'action A est mappée dans RECATNet en transition abstraite (T-callAct) qui représente l'exécution de l'action A de comportement d'appel (call behavior).

La transition (T-callAct) possède (P-$dataIn_1$ A, P-$dataIn_2$ A…… P-dataIn□A) sont des places qui représentent les n broches d'entrée de l'action A et (P-controlInA) est une place qui représente le flux de control d'entrée comme place d'entrée. Le déclenchement de cette transition abstraite permet l'exécution du module RECATNet qui représente l'activité invoquée. Les paramètres sont transmis à l'activité invoquée en fonction de l'inscription en la transition (T-callAct). Lorsque l'activité appelée se termine, tous les flux de sortie de l'activité invoquée sont transmis à l'action de comportement d'appel (call behavior). Ceci est présenté en utilisant les places de sortie qui représente les m broches de sortie de l'action A (P-$dataOut_1$ A, P-$dataOut_2$ A …… P-dataOut□A) et une place de flux control (P-controlOutA).

3. Exemple illustratif

Le processus de financement et de lancement d'un projet se fait par : 1er mot le distributeur doit établir le dossier de projet puis soumis simultanément à la banque et au siège, qui répond très rapidement. Si le siège est défavorable, le projet est abandonné et la banque est prévenue. Si le siège accepte de co-financer le projet, on attend la réponse de la banque pour décider de réétudier ou de poursuivre (call act) le dossier. Quand les deux réponses sont positives, un dossier de financement définitif est établi puis le projet est lancé. (Figure IV.12)

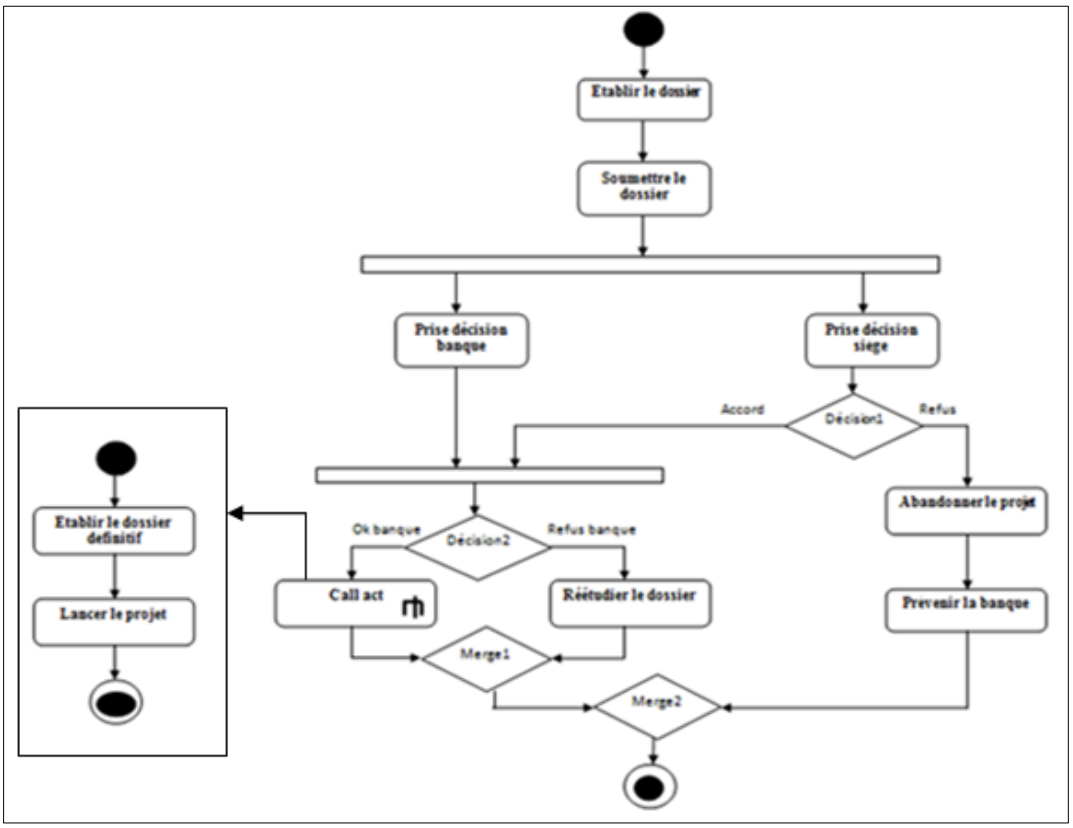


Figure IV.12 Processus de financement et de lancement d'un projet.

Quand on applique les règles de transformation présentées précédemment, le diagramme d'activité se transforme en RECATNet. (Figure IV.13)

D'abord doit établir et soumettre le dossier puis en utilise fork pour simultanément à la banque et au siège. Ensuit en utilise décision, si le siège est défavorable, le projet est abandonné et la banque est prévenue, si non on attend la réponse de la banque et en utiliser join et décision pour poursuivre ou réétudier le dossier. La transition abstraite « T-calAct » va appeler le sous-processus pour un dossier de financement définitif est établi puis le projet est lancé.

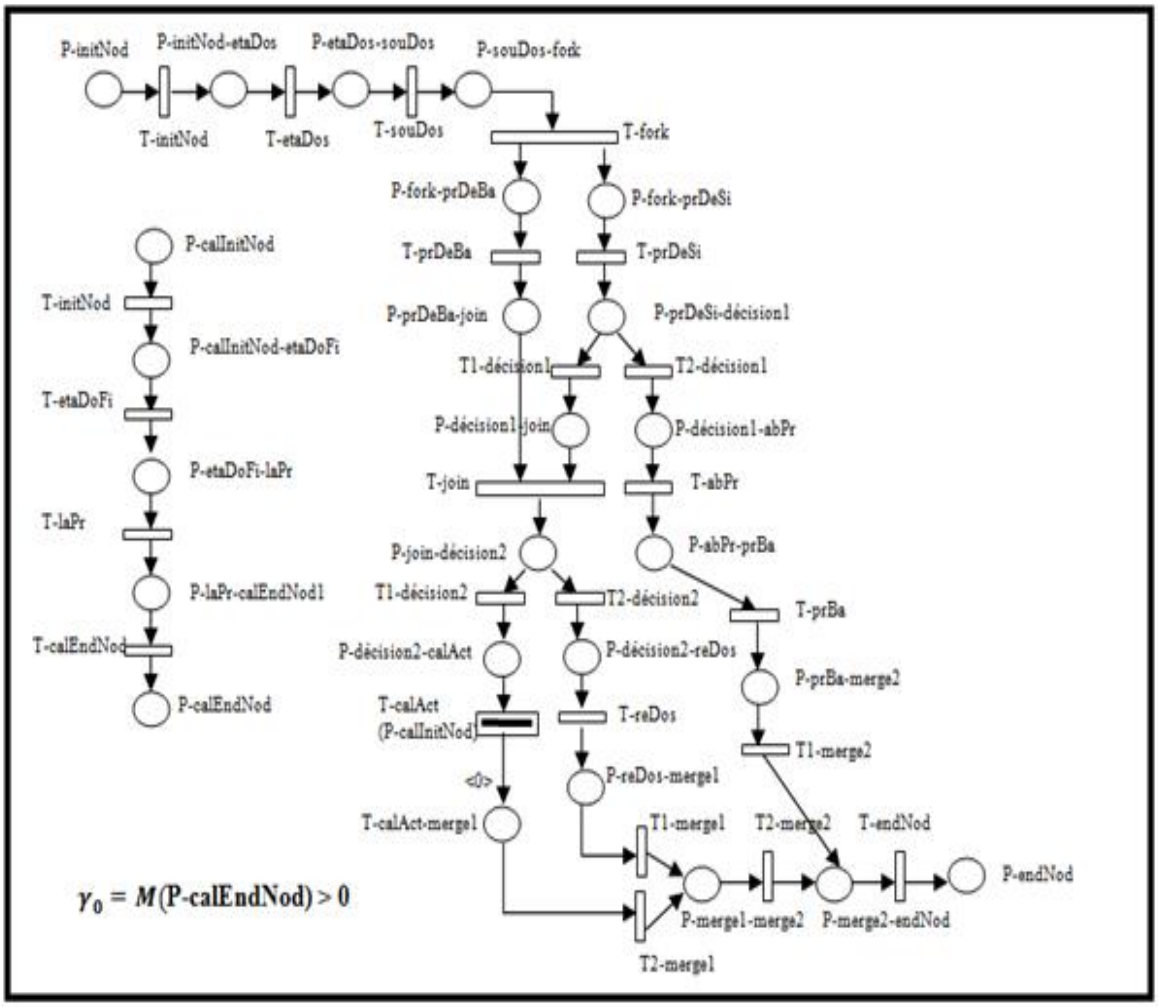


Figure IV.13 RECATNet pour le processus de financement et de lancement d'un projet.

4. Conclusion

Dans ce chapitre nous avons définis les différant règle de transformation en basant sur les spécifications de l'OMG, pour passer du modèle source : des spécifications SysML d'un diagramme d'activité au modèle cible : REACATNet, en utilisant le langage ATL pour obtenir un modèle conforme à notre méta-modèle RECATNet.

Chapitre V

Implémentation et Réalisation

1. Introduction

Après avoir présenté en détail dans les chapitres précédents les différents concepts essentiels à l'accomplissent de notre travail, alors on passe à la partie implémentation, réalisation et vérification.

Afin d'outiller la méthodologie proposée, nous avons implémenté un outil permettant de guider l'utilisateur à exécuter ses étapes. Le présent chapitre est dédié à la description de l'outil implémenté pour illustrer la méthodologie de vérification et montrer l'utilisation de l'outil développé.

2. Processus de développement

La démarche que nous avons proposé est représentée par la figure V.1. Elle la démarche admet plusieurs entrées. La première étant le diagramme d'activité (DA) qui doit être validé syntaxiquement. La seconde entrée est le digramme d'exigence (DE) qui comporte une ou plusieurs exigences liées aux diagrammes d'activité. La première étape consiste à modéliser le système par le langage de spécification SysML. Nous proposons d'utiliser les diagrammes d'activité pour représenter le comportement du système à développer. L'étape suivante consiste à passer d'une spécification SysML (DA) vers une spécification formelle, ceci permet d'obtenir un langage formel à partir d'un langage semi-formel. L'idée étant de réaliser une transformation d'un DA à un RECATNet en se basant sur le langage de transformation ATL (Atlas Transformation Language). Le RECATNet dérivé sera en format XMI. Le format résultant « XMI » sera transformé vers l'approche Maude. Cette transformation est basée sur des règles implémentées dans le générateur de code 'Acceleo'.

En parallèle des étapes réalisées, nous procédons à une étape de transformation des exigences SysML en formules de logique temporelle linéaire (LTL).

Une fois la transformation est accomplie, nous pouvons, finalement, commencer la vérification des propriétés LTL à l'aide de l'outil model-checker Maude.

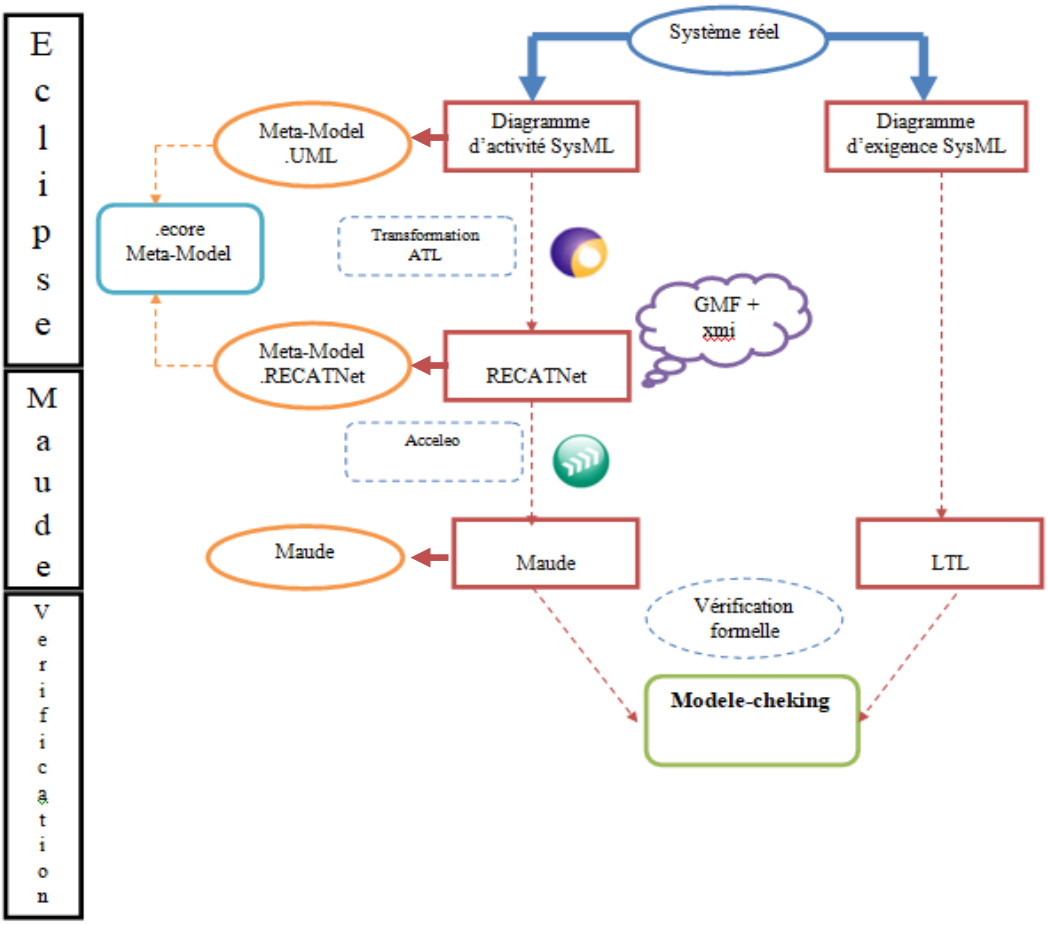


Figure V.1 Démarche générale

3. Etude de cas

Selon la méthodologie proposée au dessus, le but de cette étape est de produire un diagramme d'activité et un diagramme d'exigence *SysML* pour le système de la machine qui vend les tickets de métro.

3.1. Description de cas

Nous spécifions le comportement du système de la machine qui vend les tickets de métro (MVT) par le diagramme d'activité présenté dans la figure V.2. Ce diagramme a été édité en utilisant l'éditeur des diagrammes *papyrus* intégré dans PolarSys.

Lorsqu'un passager arrive pour acheter le ticket, il sélectionne le choix (acheter un ticket) après il choisi l'option qu'il veut, soit d'acheter des tickets normale ou d'acheter le paquet de 10 tickets car la déférence est dans le prix. Après cette étape sa vient l'étape du paiement. Le diagramme contient le call behaviour action (Procédure de paiement) qui fait appel à une autre activité représenté par un sous diagramme d'activité nommé Procédure de paiement, il est donnée par la figure V.3 L'activité Procédure de paiement se fait soit en cash, en rentre les billets d'argent ou les pièces et on valide, soit avec carte en rentre le code et on valide.

quand la fin de l'activité appelés par le call behaviour action Procédure de paiement, dans l'activité principale MVT, deux cas sont modélisés, les tickets sortent et le passager les récupèrent, la machine affiche un message de remerciement c'est le cas d'un paiement correct. Ou dans le cas échoué où la machine ne reçoit ni argent ni carte ou la carte ne marche pas, le paiement sera annulé ainsi que tous le procédure et la demande des tickets sera annulée, donc la machine affiche le message d'annulation.

3.1.1. Diagramme d'activité

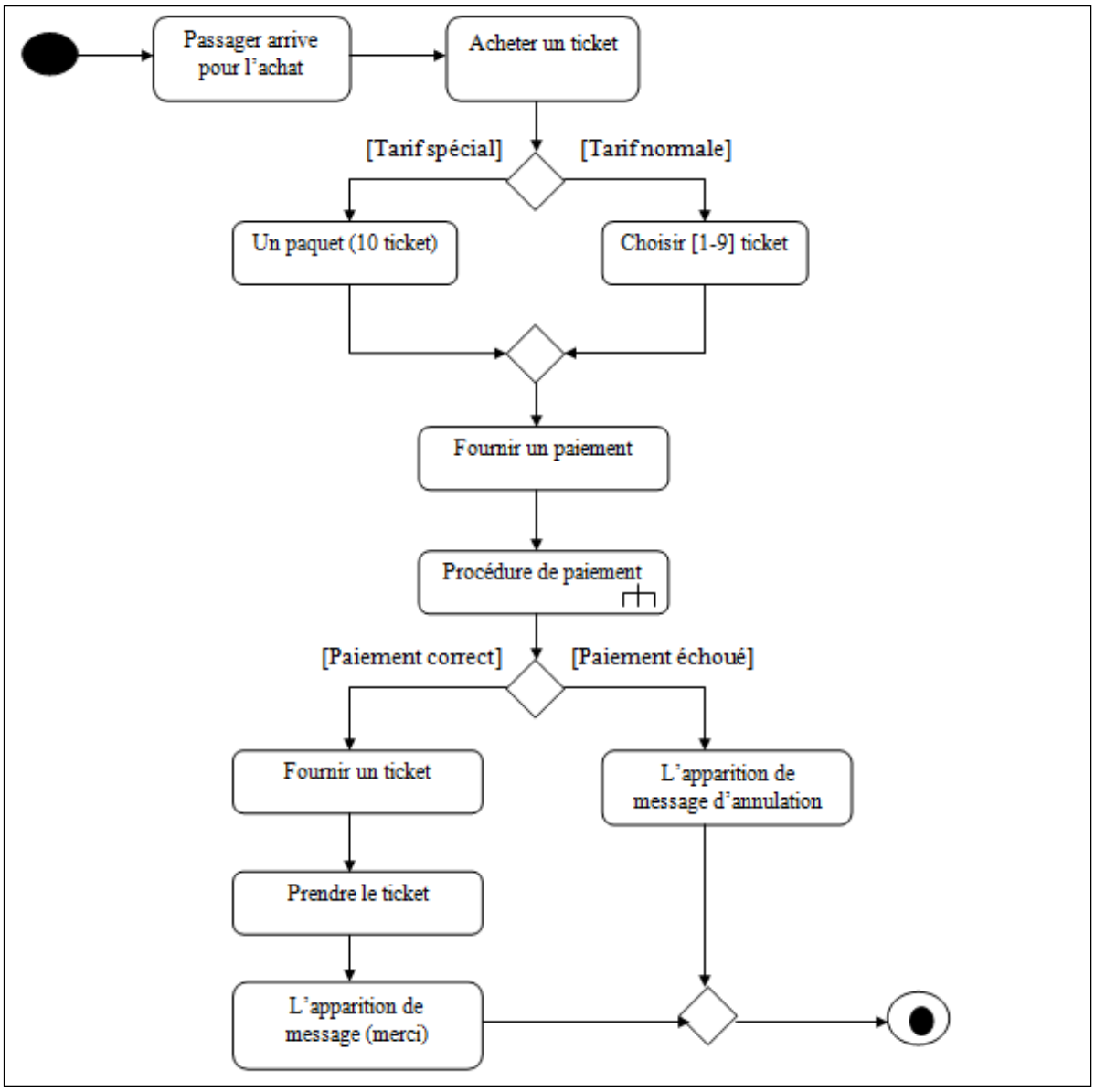


Figure V.2 L'activité principale de la machine qui vend les tickets

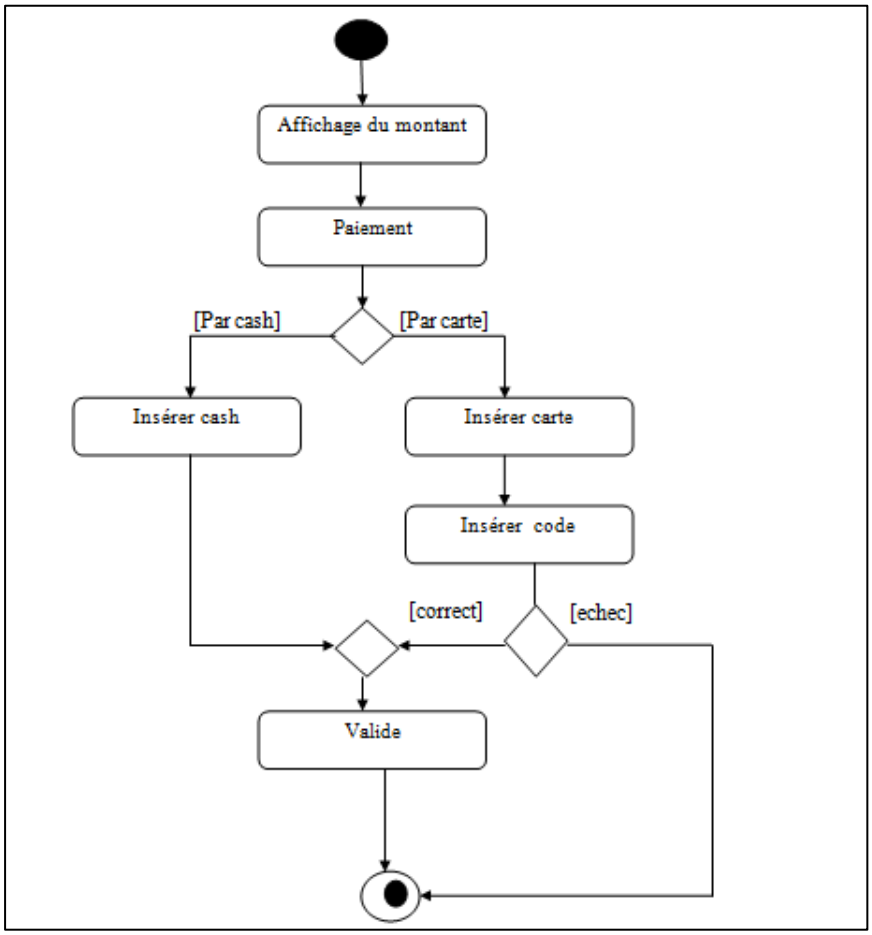


Figure V.3 L'activité appelée procédure de paiement

3.1.2. Diagramme d'exigence

Les exigences de système de la machine qui vend les tickets de métro sont modélisés dans un diagramme d'exigence *SysML*. Dans ce contexte, nous considérons seulement un ensemble d'exigences. Comme montre dans la figure V.4.

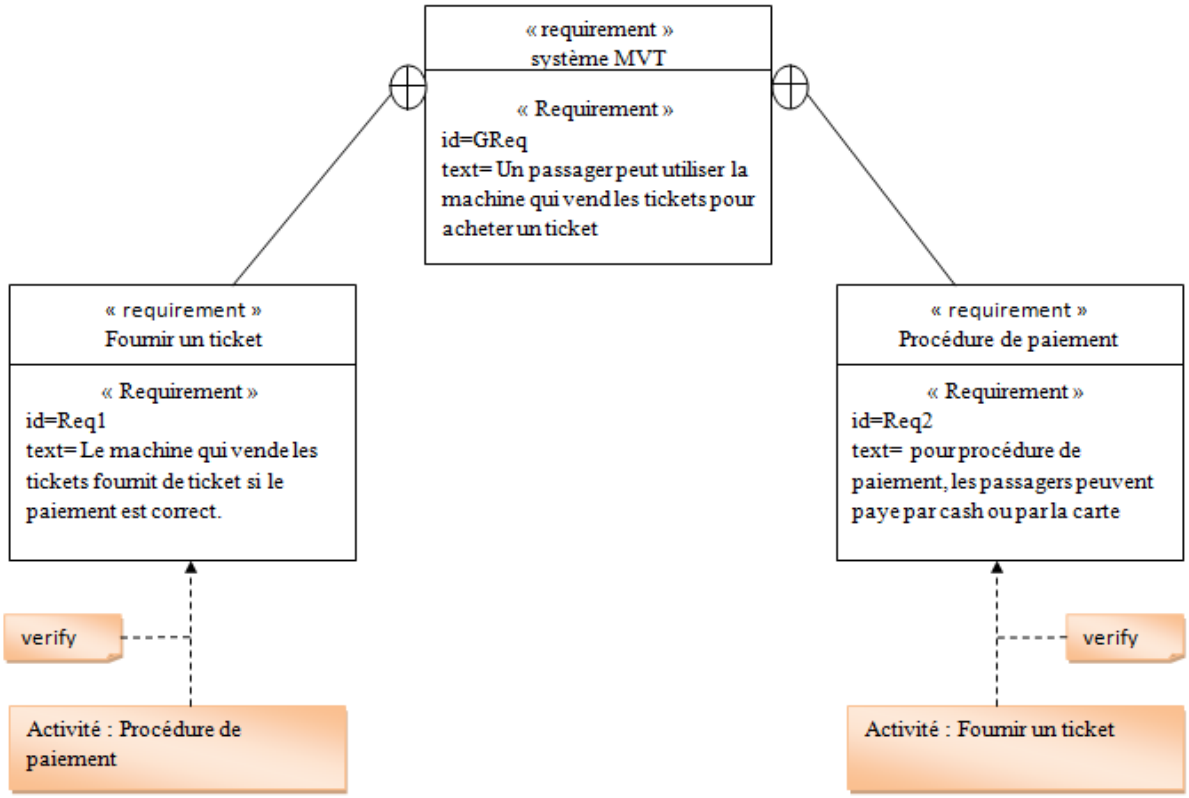


Figure V.4 Le diagramme d'exigence pour le système de la machine qui vend les tickets

L'exigence est composée de deux exigences (*Req*1, *Req*2). L'exigence *Req*1 est le cas où la machine fournit un ticket. Cette exigence est vérifiée par l'activité *Procédure de paiement* L'exigence *Req*2 concerne le moyen de paiement. L'exigence *Req*2 est vérifiée par l'action *Fournir un ticket* (voir Figure V.4).

Les exigences qui ont la relation << *verify* >> sont exprimées des exigences *SysML* en langage naturel.

4. Transformation des spécifications SysML en langages Formelles

Cette étape consiste a transformé le modèle précédents en des langages formelles. Premièrement, on va translater le diagramme d'activité en RECATNet. Ensuit le diagramme d'exigence en LTL manuellement.

4.1. Transformation de diagramme d'activité en RECATNet

Dans cette section on va présenter le résultat de transformation du diagramme d'activité (voir la figure V.2) en RECATNet. Pour cela, nous utilisons l'interface graphique que nous avons criée pour faciliter les procédures de transformation. L'interface est présentée dans la figure V.5.

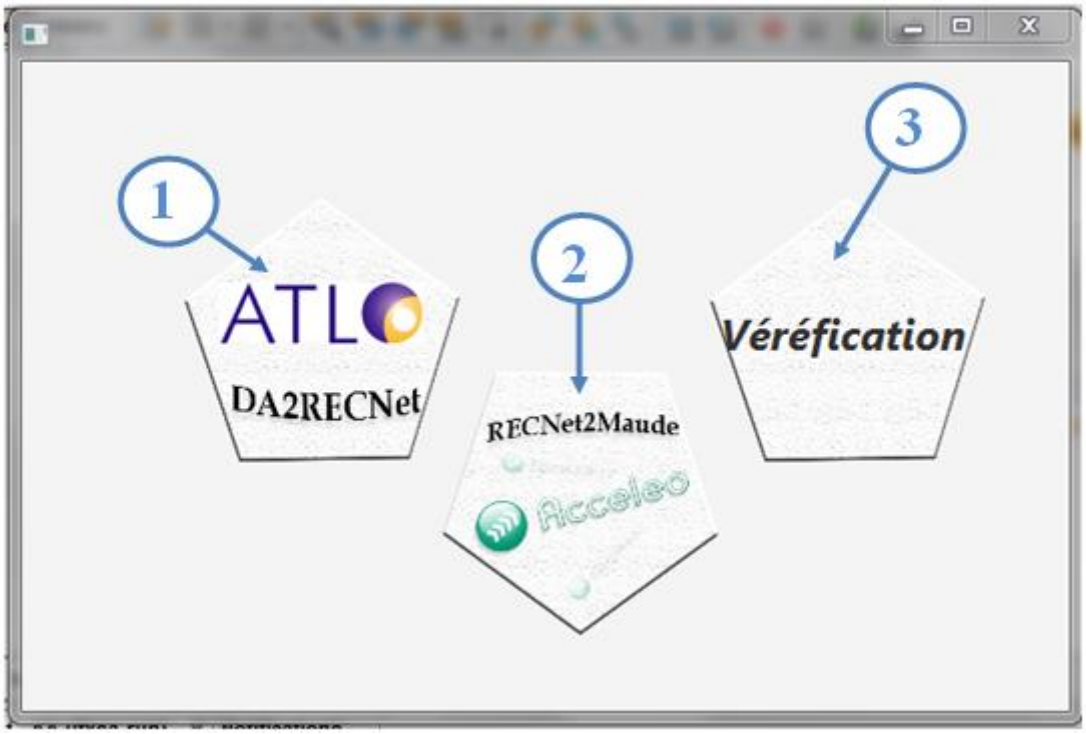


Figure V.5 L'interface principale

Premièrement, on clic dans l'interface de la figure V.5 le bouton numéroté par 1 qui concerne la transformation de diagramme d'activité en RECATNet. Ça donne interface dans la figure V.6. Ensuit, on choisit le fichier du diagramme d'activité par bouton numéro 4, son chemin est apparu dans le champ numéroté par 5 lorsque le fichier est ouvert le résultat s'apparait dans la figure V.7.

Puis on lance la transformation par le bouton numéro 6 qui nous donne l'espace de l'enregistrement puis l'apparition de résultat (figure V.8) sous format .XMI, pour un autre modèle on clic dans le bouton 7 (clean) et on choisit le nouveau modèle.

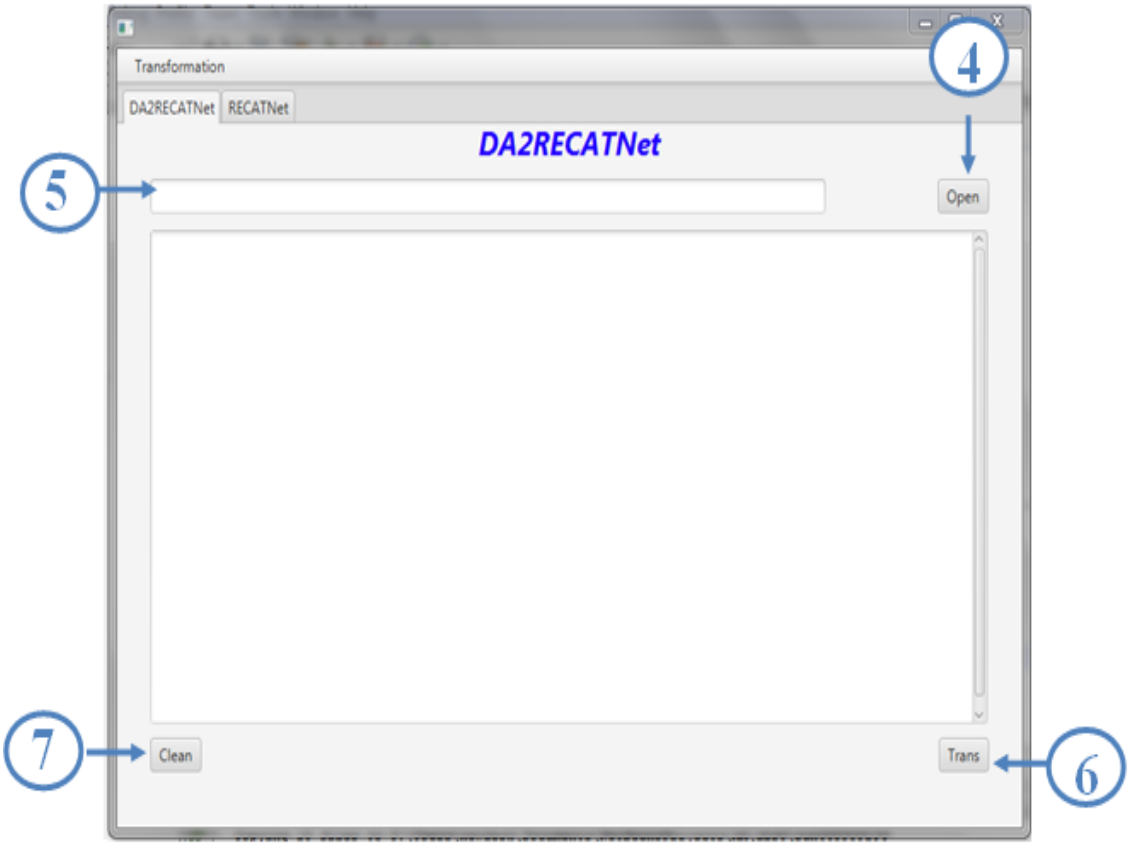


Figure V.6 Interface graphique de transformation DA2RECATNet

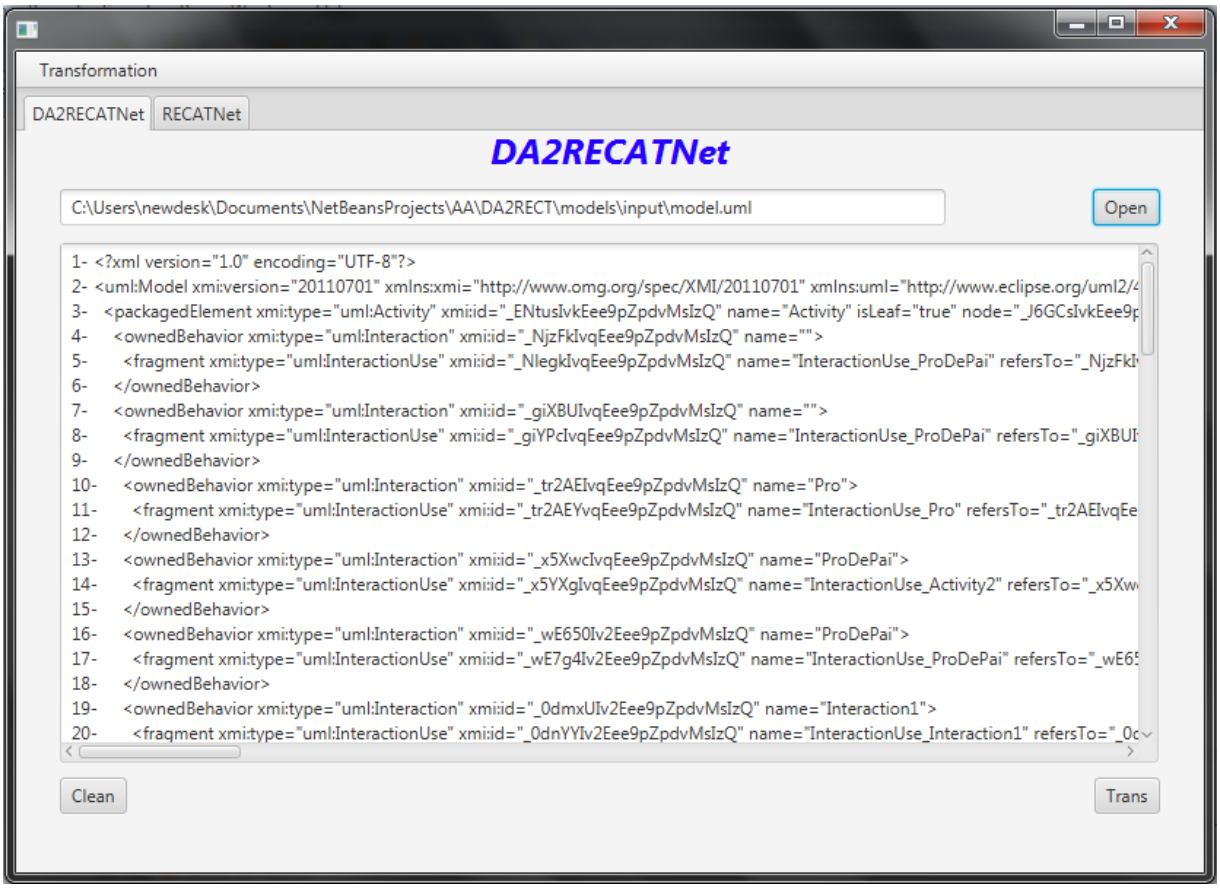


Figure V.7 Interface graphique de model.uml

Après la transformation dans la figure V.8 il y'a deux bouton numéroté par 8 qui affiché le graphe de RECATNet résultat de la transformation. Si on clique sur ce bouton « Graphe » une autre interface va s'afficher pour le graphe dessiné (figure V9, V10). Et l'autre de graphe avec GMF (figure V.11)

Et pour imprimé le fichier de .XMI on clic sur le bouton numéro 9.

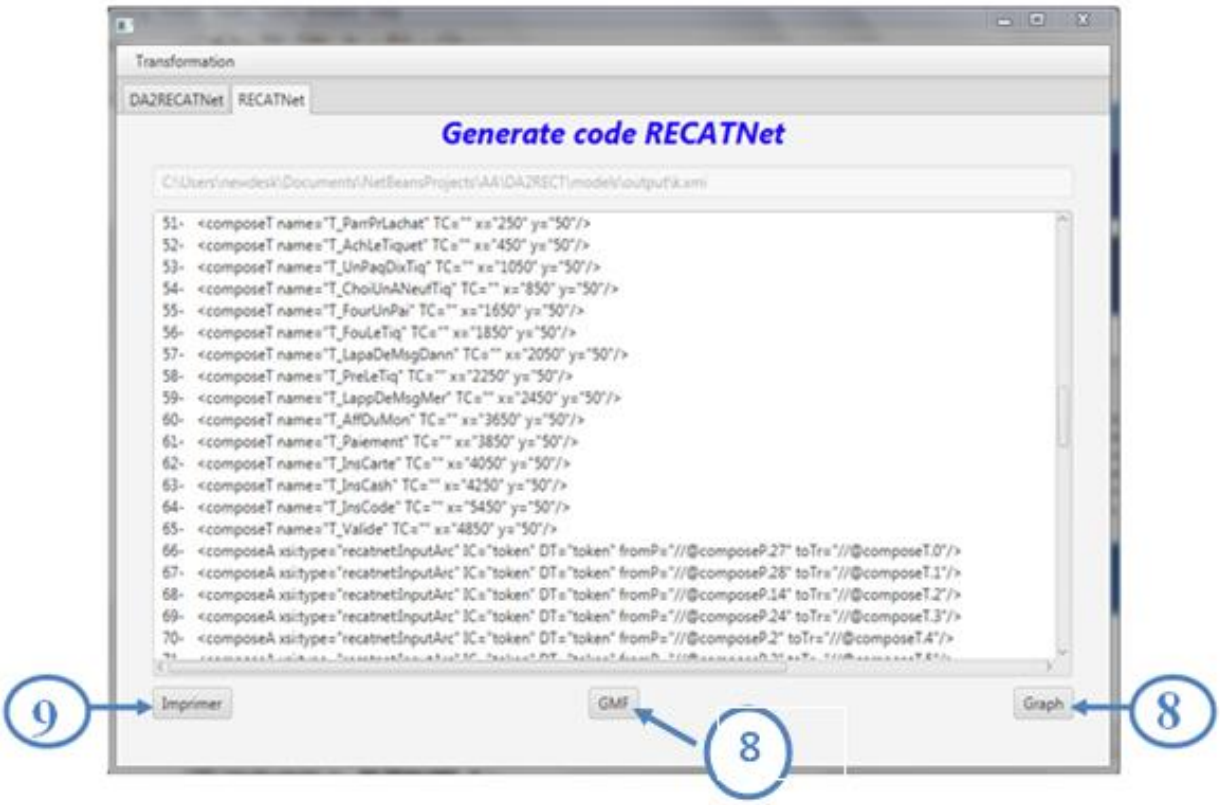


Figure V.8 Interface graphique résultat de transformation DA2RECATNet

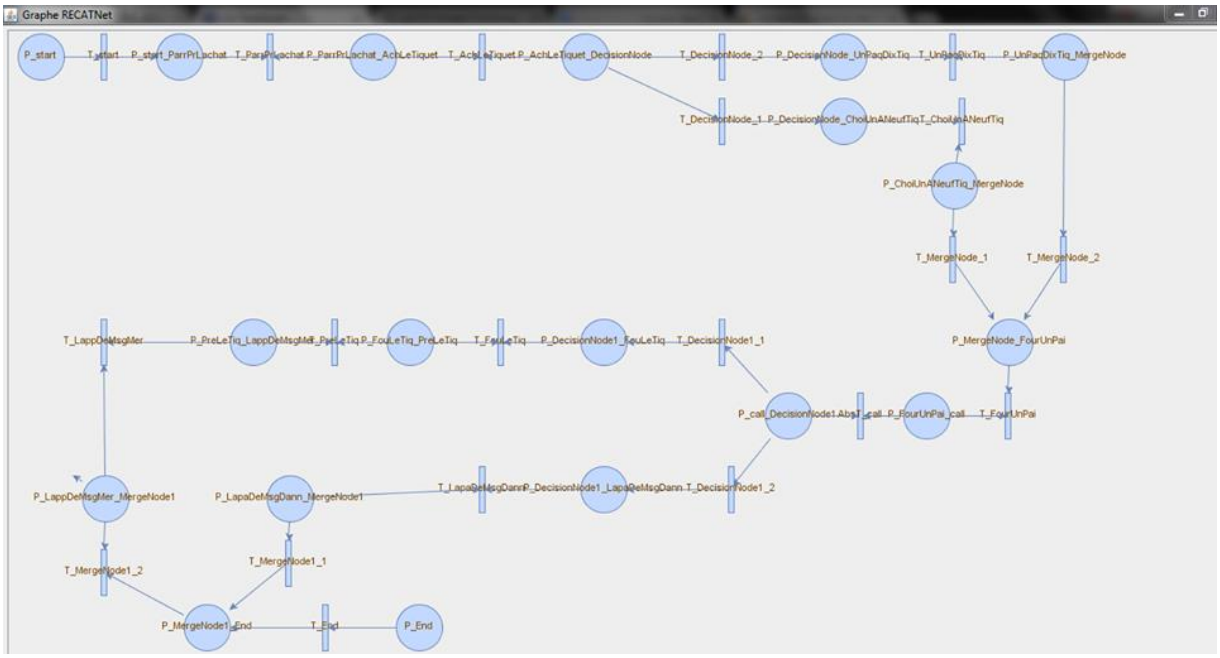


Figure V.9 Résultat graphique de transformation

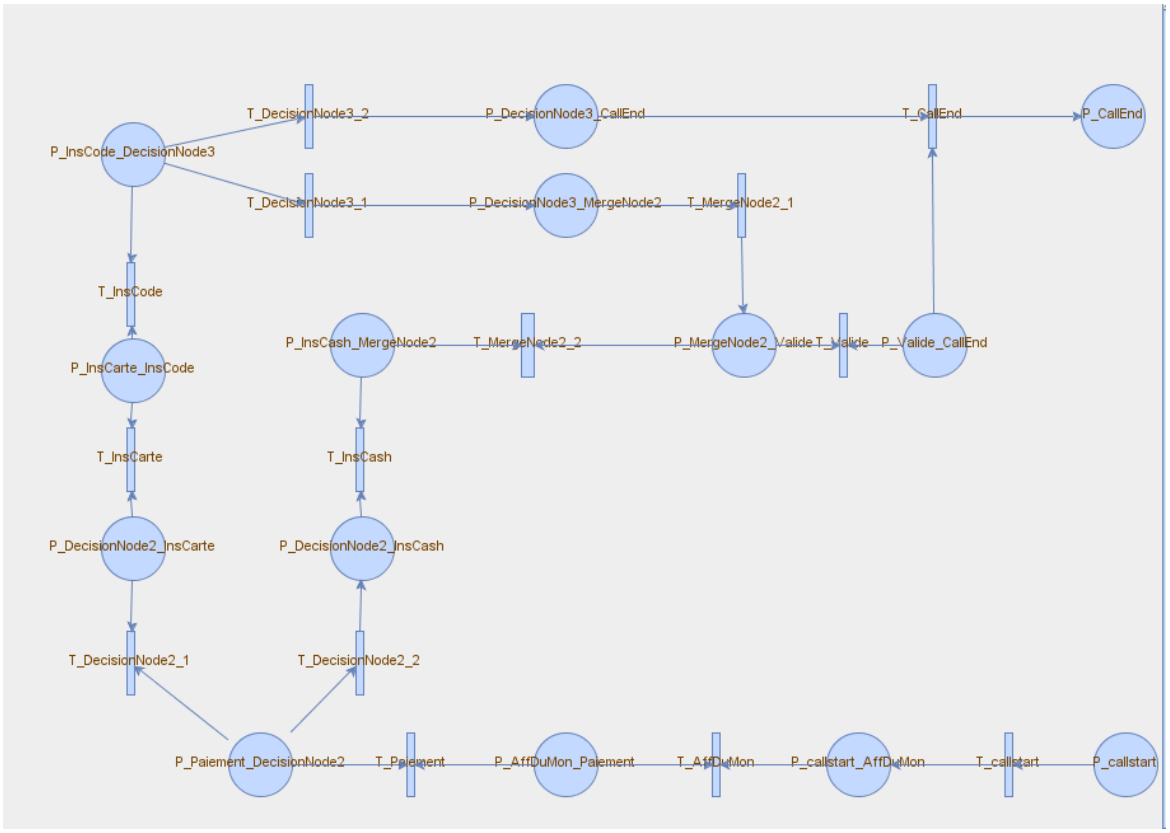


Figure V.10 Résultat graphique de transformation (call behavior)



Figure V.11 Résultat graphique de transformation sous format GMF

4.2. Transformation diagramme d'exigence en LTL

Dans cette étape consiste a transformé diagramme d'exigence en LTL a l'aide les exigences *AcTRL* manuellement**[43]**

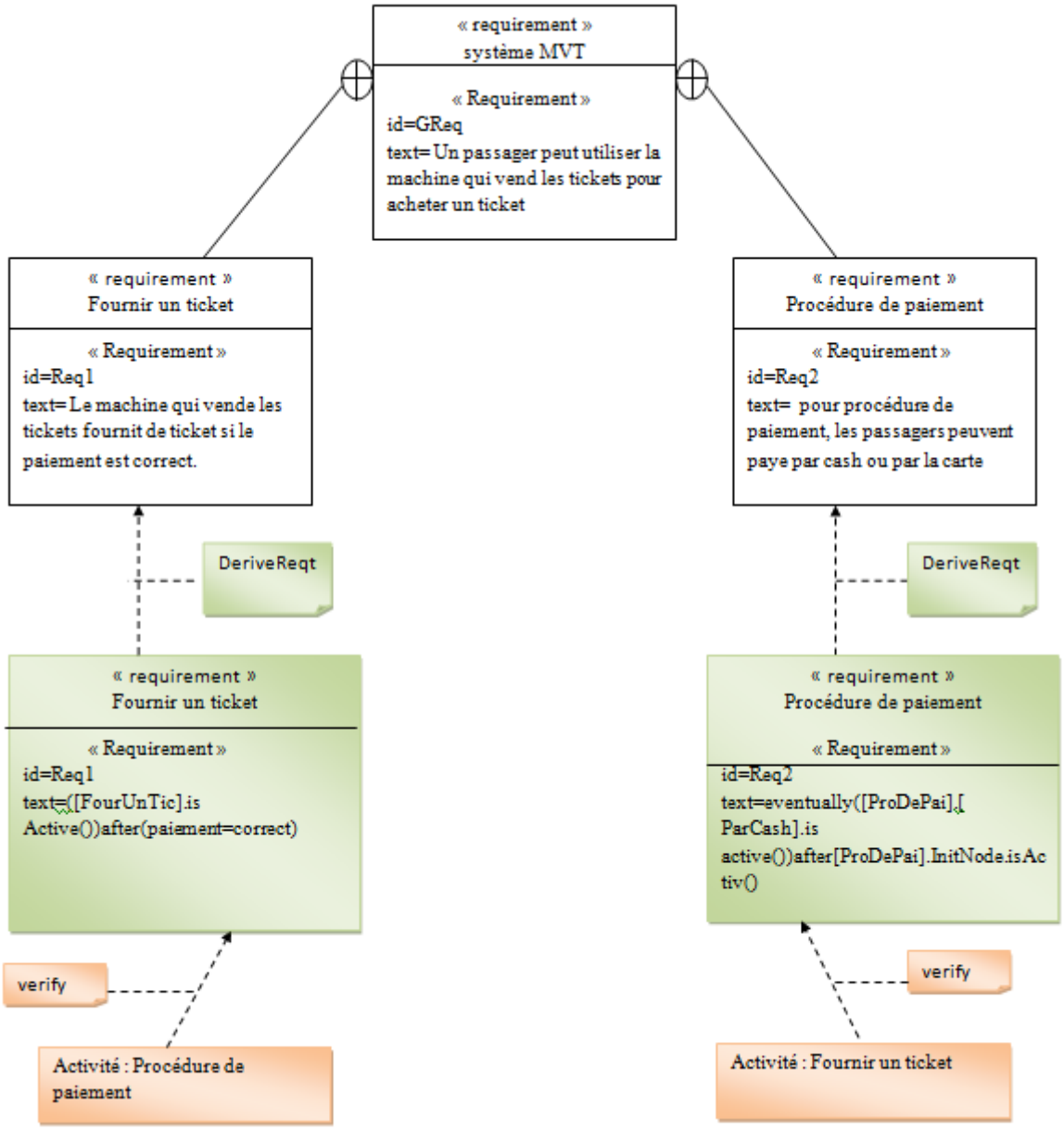


Figure V.12 Le DE après l'intégration des expressions extrait de l'éditeur ActRL

Req1 :

[](paiement=Correct->[](([FourUnTic].is active())))

Req2:

[](![ProDePai].InitNode.isActiv()<>([ProDePai].InitNode.isActiv() and ([ProDePai].[ParCash].is active()))

5. Génération de code Maude à partir un modèle formelle RECATNet

Dans cette partie de la transformation en Maude à partir d'un modèle RECATNet, Pour cela nous utilisons l'interface de la figure V.5, et on choisi le bouton numéroté par 2 qui concerne la transformation de modèle RECATNet en Maude, ça donne interface dans la figure V.13. Puis, on choisit le fichier .XMI par bouton numéro 10, son chemin est apparu dans le champ numéroté par 11 lorsque le fichier est ouvert le résultat s'apparait dans la figure V.14. Ensuit on lance la transformation par le bouton numéro 12 qui nous donne l'espace de l'enregistrement puis l'apparition de résultat dans la figure V.15, pour un autre modèle on clic dans le bouton 13 (clean) et on choisit un nouveau fichier .XMI



Figure V.13 Interface graphique de transformation RECATNet2Maude

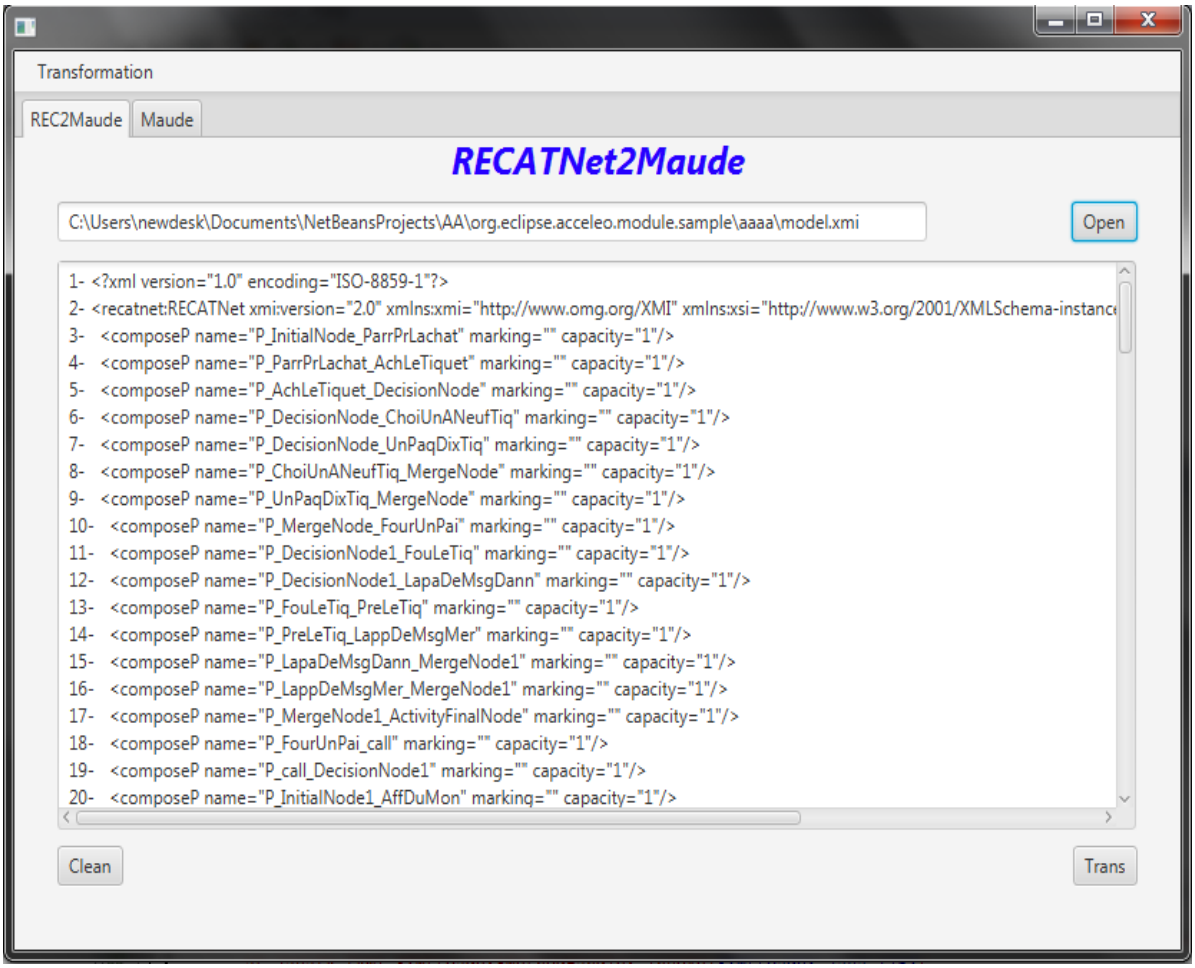


Figure V.14 Interface graphique de modèle .XMI

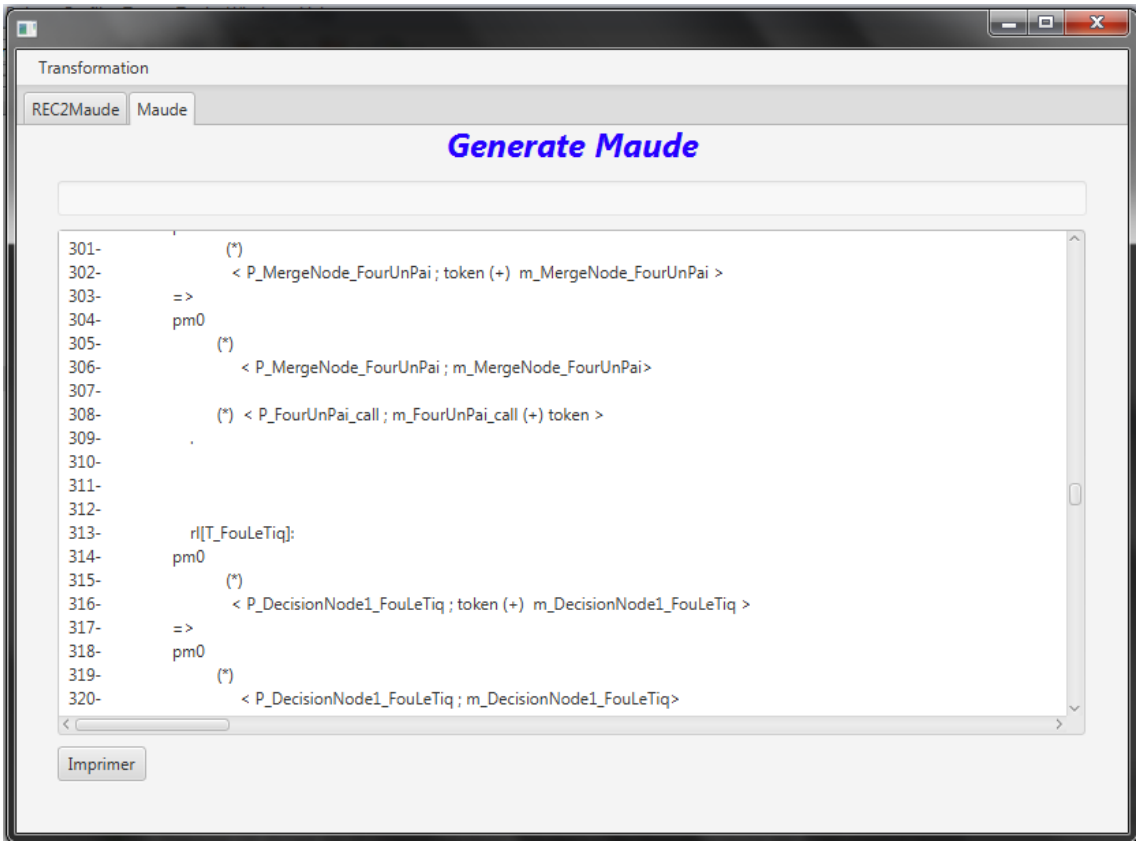


Figure V.15 Interface graphique résultat de transformation RECATNet2Maude

6. Vérification avec Maude

Ce qui concerne la vérification, nous utilisons dans un premier temps la fonctionnalité de la génération des spécifications Maude offerte par notre outil. Dans un deuxième temps, nous présentons un exemple d'analyse formelle en utilisant le model checker « Maude LTL Le model checker ».

6.1. Vérification des propriétés

Pour bien illustrer l'utilisation du Maude LTL Model Checker, nous reprenons le module 'RECATNet' de machine MVT (*Figure V.2 et Figure V.3)*. Une propriété temporelle que nous pouvons vérifier si quand le paiement est correct un ticket sera fourni.

Pour vérifier si le module 'RECATNet' satisfait cette propriété, nous avons défini d'abord dans deux nouveaux module qui contiennent un prédicat initial et prédicat finale, 'RECATNET-PRED' et 'RECATNet-CHEK' représenté ci-dessous.

```
mod RECATNET-CHECK is
protecting RECATNET-PREDS .
including MODEL-CHECKER .
including LTL-SIMPLIFIER . *** optional


var M : Marking .
op initialState : -> Extended-marking .


eq initialState = [ < P-start ; token > (*) < P-start-ParrPrLachat ; ems > , nullTrans , nullFils ].

endm


rew modelCheck ( initialState  , [] (PaiementCorrect -> [] (FourninTiquet)) ) .
```

Figure V.16 Module RECATNET-CHECK

```
mod RECATNET-PREDS is
protecting RECATNet .
including SATISFACTION .

subsort Extended-marking < State .
vars T T1 : TransAbs .
vars Th Th1 : Extended-marking .
vars M Ma : Marking .

------------------------------
ops PaiementCorrect  : -> Prop .
ops FourninTiquet : -> Prop .
-------------------------------------

eq  [ < P-LapaDeMsgDann ; ems >   , T , Th ]  |= PaiementCorrect = true .


ep [ <  P-FouLeTiq ; token >  , T , Th ] |= FourninTiquet = true .


endm
```

Figure V.17 Module RECATNET-PRED

Pour vérifier la propriété LTL : [] (paiement=Correct-> [] ([FourUnTic].is active()))

nous devons appeler la fonction ‘modelCheck’. Cette fonction est définie dans le module « MODEL-CHECKER ». Elle est invoquée sur l’interpréteur Maude Comme suit :

$rew\ modelCheck\ (initialState,$ [] (PaiementCorrect) -> (FourninTiquet)).

Enfin, le résultat de la vérification est représenté dans la *Figure V.17* En utilisant le module ‘RECATNet’ dans le niveau de spécification système.

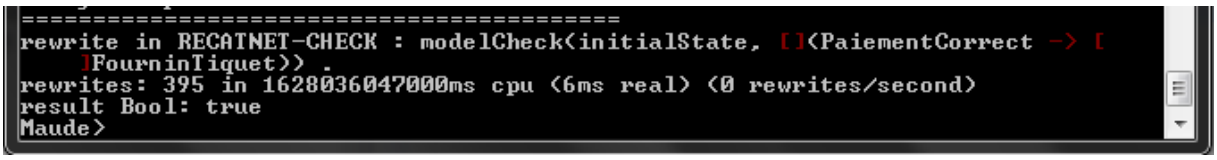


Figure V.18 Résultats de la vérification

7. Conclusion

Dans ce chapitre, nous avons présenté une étude de cas où nous avons utilisé les spécifications SysML, un diagramme d’activité et un diagramme d’exigence. Ensuite, nous avons translaté le diagramme d’activité en RECATNet, et translaté les exigences manuellement en LTL. Puis, nous avons générer un code Maude à partir d’un modèle RECATNet. Enfin, nous pouvons commencer la vérification des propriétés LTL à l’aide de l’outil model-checker Maude. Pour faciliter ces procédures, nous avons développé une application qui automatise les étapes de l’approche proposée.

Conclusion Générale

Le SysML est un langage dédié à l'ingénierie des systèmes. Malgré les avantages qu'il présente, il reste un langage semi-formel, c.à.d. nécessitant une formalisation pour pouvoir effectuer des vérifications formelles. Une des approches proposées pour résoudre ce problème consiste à co-utiliser le langage SysML avec des méthodes formelles.

Dans ce travail de fin d'étude, nous avons proposé une démarche qui permet la vérification automatique des exigences SysML sur les diagrammes d'activité. Notre démarche est basée sur la transformation des spécifications SysML en spécifications formelles. Comme Dans la première étape, nous avons transformé les diagrammes d'activité en RECATNet, en utilisant le l'approche IDM, et en respectant les spécifications définies par l'OMG. Pour implémenter les règles de transformation, nous avons fait appel au langage de transformation ATL.

Dans la deuxième étape, nous avons exprimé les RECATNets obtenus lors de la première étape en terme de la logique de réécriture. Ceci a été réalisé en utilisant le langage de transformation Acceleo.

En parallèle, nous abordons l'étape de transformation des diagrammes d'exigences en formules de logique temporelle linéaire (LTL).

Lorsque toutes les transformations sont achevées, nous pouvons entamer la vérification des propriétés LTL à l'aide de l'outil model-checker Maude.

Enfin, nous avons implémenté l'approche proposée dans un outil JAVA permettant de vérifier les propriétés fonctionnelles, décrites par les diagrammes d'exigences, sur les diagrammes d'activités SysML.

Perspectives

Nous proposons avons comme perspective :

- ✓ Faire une vérification formelle d'autres diagrammes des SysML.
- ✓ Appliquer notre approche sur d'autres cas d'études inspirés du monde réel.

Bibliographie

[1] *Object Management Group, Systems modeling language (sysml) specification*. SysML Partners (www.sysml.org). 2012

[2] Guillaume Finance, Djug : *Modélisation SysML*. Publié le 27 décembre 2010 http://perso.numericable.fr/starnaud/Analyse/Cours%20sysML.pdf

[3] *Analyse des systèmes : Langage SysML*. http://perso.numericable.fr/starnaud/Analyse/Cours%20sysML.pdf

[4] Guillaume FINANCE *: Modélisation SysML*. 30 septembre 2010, http://www.umlchannel.com/docs/Article%20Mod%C3%A9lisation%20SysML.pdf

[5] Didier Fagnon, Stéphane Gaston : *SysML : les diagrammes*. avril 2012. http://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/techniques/979/979-179-p100.pdf

[6] Pascal Roques : *SysML par l'exemple*, Code éditeur : G85006 ,ISBN : 978-2-212-85006-2 , Eyrolles . 2009

[7] *Livementor*. 2016. https://www.livementor.com/ressources/fiche/641

[8] Agusti CANALS : *sysml : une notation pour spécifier et concevoir des systèmes*.

[9] Samba Diaw, Redouane Lbath, Bernard Coulette : *etat de l'art sur le développement logiciel basé sur les transformations de modèles.* Article. 2010.

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

[11] Benoit Combemale : *Ingénierie Dirigée par les Modèles (IDM)*. 2008.

[12] Yassine Ouhammou, Mickael Baron alias keulkeul : *Initiation à l'ingénierie dirigée par les modèles*. 22 mai 2012 http://yassineouhammou.developpez.com/tutoriels/conception/initiation-ingenierie-dirigee-modeles/#LII

[13] *Object Management Group, Inc. Meta Object Facility (MOF) 2.0 Core Specification*. January 2006.

[14] Xavier blanc : *MDA en action ingénierie guidée par les modèles*.Groupe Eyrolles, 2005.

[15] Ivan Kurtev, Jean Bézivin, and Mehmet Aksit. Technological Spaces : An Initial Appraisal. In CoopIS, DOA'2002 Federated Conferences, Industrial track, Irvine, 2002.

[16] Jean Bézivin, Frédéric Jouault, Peter Rosenthal, and Patrick Valduriez. Modeling 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.

[17] Jean Bezivin and Ivan Kurtev. Model-based Technology Integration with the Technical Space Concept. In Metainformatics Symposium, Esbjerg, Denmark, 2005. Springer-Verlag.

[18] Jean Bezivin. Xavier Blanc : *Promesses et interrogations de l'approche mda. Journal développeur référence*. septembre 2002.

[19] Mouna Bouarioua : *Une approche basée transformation de graphes pour la génération de modèles de réseaux de Petri analysables à partir de diagrammes UML*. Thèse de Doctorat en Informatique. 24/ 04 /2013.
http://www.univ-constantine2.dz/files/Theses/Informatique/Doctorat/Bouarioua-Mouna.pdf .

[20] Krzysztof Czarnecki and Simon Helsen : *Classification of Model Transformation Approaches*, OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture. 2003.

[21] Houda Hamrouche : Une Approche de transformation des diagrammes D'activité d'UML vers CSP basée sur la transformation de graphes. Thèse de MAGISTER en INFORMATIQUE.
http://vrpg.univskikda.dz/recherchePG/theses_memoires/fac_sciences/infor/memoire_houda_Hamouche.pdf

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

[23] *Meta object facility (mof) 2.0 query/view/ transformation specification*. Version 1.1. 2011.

[24] Samba Diaw, Rédouane Lbath, Bernard Coulette : *etat de l'art sur le développement logiciel dirigé par les modèles* .Article . 12. janvier 2009.

[25] Raida ElMansouri : *Modélisation et Vérification des processus métiers dans les entreprises virtuelles : Une approche basée sur la transformation de graphes*. Thèse de Doctorat en Sciences en Informatique,http://bu.umc.edu.dz/theses/informatique/ELM5432.pdf

[26] Dehimi Nardjess Tissilia : *Un Cadre Formel pour La Modélisation et L'analyse Des Agents Mobiles*. Thèse de Doctorat en Sciences en Informatique, http://www.univ-constantine2.dz/files/Theses/Informatique/Doctorat/DEHIMI-NARDJESS-TISSILIA.pdf

[27] François Defossez : *Modélisation discrète et formelle des exigences temporelles pour la validation et l'évaluation de la sécurité ferroviaire*. Submitted on 7 Apr 2011, https://tel.archives ouvertes.fr/tel-00584005

[28] Pierre LADET : *r7252 réseaux de petri*. Cet article est issu de : Archives.

[29] Diaz, Michel : *Les Réseaux de Petri – Modèles fondamentaux*. Hermes Science Publications : Paris. 2001, http://www.metz.supelec.fr/metz/personnel/vialle/course/CNAM-ACCOV-NFP103/extern-doc/RdP/RdP-modelisation.pdf

[30] Peterson, J. L : *Petri Nets.* 1977. http://www.daimi.au.dk/PetriNets/, http://www.metz.supelec.fr/metz/personnel/vialle/course/CNAM-ACCOV-NFP103/extern-doc/RdP/RdP-modelisation.pdf

[31] *Les Réseaux de Petri – Supelec***.** 20/04/2007. http://www.metz.supelec.fr/metz/personnel/vialle/course/CNAM-ACCOV-NFP103/extern-doc/RdP/Introduction-RdP.pdf

[32] *Le formalisme de réseau pétri*. Pdf. http://tadic.eb2a.com/chapitre%202.pdf

[33] Benmoussa Fatima Zohra, Houidi Farida : *Une approche basé sur la transformation du graphe sous ATom3 pour l'intégration des deux outils de réseau de petri* .2014 – 2015

[34] Razclos : *Réseaux de Petri*. Cours de programmation des systèmes II. 1998, http://référenceweb,1998

[35] *Vérification Formelle*. Support de Cours, Master : RSD. 2009 – 2010

[36] Ben Saud : *Les réseaux de petri*. avril. 2015

[37] Yann Morère : *Cours de réseau de Petri* . Avril 2002

[38] Arthur ter Hofstede, Boualem Benatallah: *Business Process Management Workshops*. Livre.2007

[39] K.Barkaoui , A. Hicheur: *Towards Analysis of Flexible and Collaborative Workflow Using Recursive ECATNets," in Business Process Management Workshops*. 2008

[40] S. Haddad , D. Poitrenaud: *Theoretical Aspects of Recursive Petri Nets*, in Application and Theory of Petri Nets. 1999.

[41] F.Belala ,M. Bettaz : *A Natural Semantic Framework for ECATNets*. Journal of Applied Categorical Structures. 2002.

[42] Messaoud Rahim , Ahmed Kheldoun Malika Boukala-Ioualalen , Ahmed Hammad: *Recursive ECATNets-based approach for formally verifying system Modelling Language activity diagrams*.IET Software 9(5): 119-128 .2015.

[43] Benrekia Mohamed Mamoune, Menacer Abderaouf : *Formalisation d'exigences SysML relatives aux diagrammes d'activités*. Un PFE master encadre par de mr Rahim.2017

Méta-modèle source (diagramme d'activité)

Le diagramme d'activité est un des quatre diagrammes comportementaux de SysML. Il est composé de 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 spécialise et peut grâce aux arguments associés avoir un comportement sensiblement différent. La figure A.1 présente une traduction en diagramme de classe UML du méta-modèle de diagramme d'activité (simplifié) présent dans la spécification OMG.

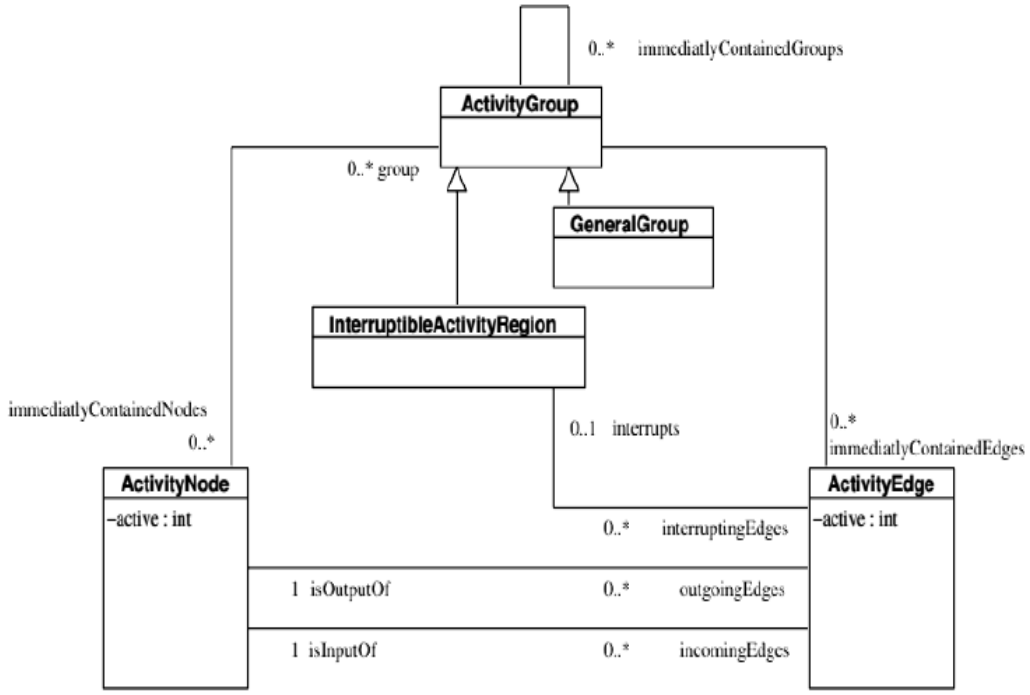


Figure A.1 La partie supérieure de méta-modèle de diagrammes d'activité

Description de méta-modèle

Une action peut avoir des flux d'activité entrants et sortants qui spécifient les flux de contrôle et les flux de données vers et depuis d'autres noeuds. La figure A.2 présente la méta-classe (classe du niveau m´eta-mod`ele) "ActivityEdge" : une méta-classe possédant deux liens d'héritage avec les méta-classes "ObjectFlow" et "ControlFlow".

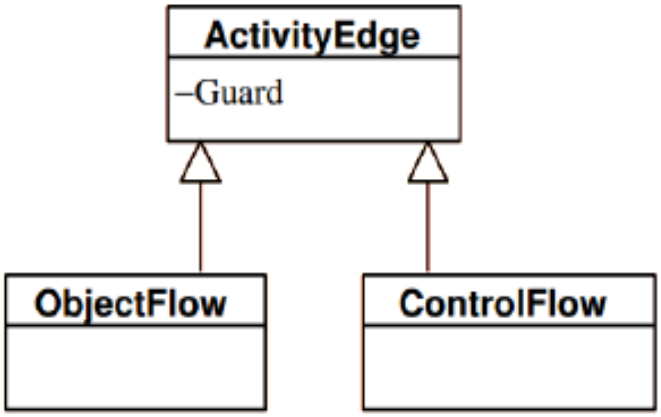


Figure A.2 Les flux d'activité (Activity Edge)

Comme il est dit précédemment, les *DA*s sont composés de noeuds. Ces noeuds sont de contrôle, d'objet ou d'action. Cette description a été appliquée pour l'ensemble du méta-modèle présenté en figure A.3

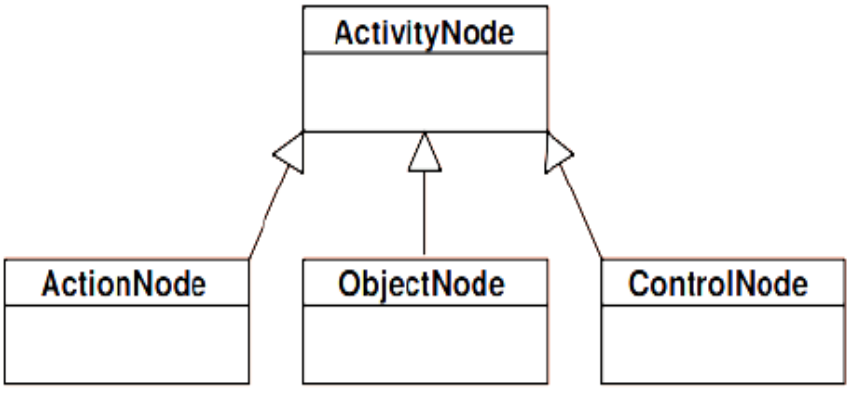


Figure A.3 Les nœuds d'activité (ActivityNode)

Il existe plusieurs types de noeuds de contrôle : noeud initial (initial node), noeud de fin d'activité (final node), noeud de fin de flot (flow final), noeud de décision (decision node), noeud de fusion (merge node), noeud de bifurcation (fork node), noeud d'union ”synchronisation” (join node). La partie de méta-modèle qui décrit ces noeuds est représentés par le diagramme donné par la figure A.4

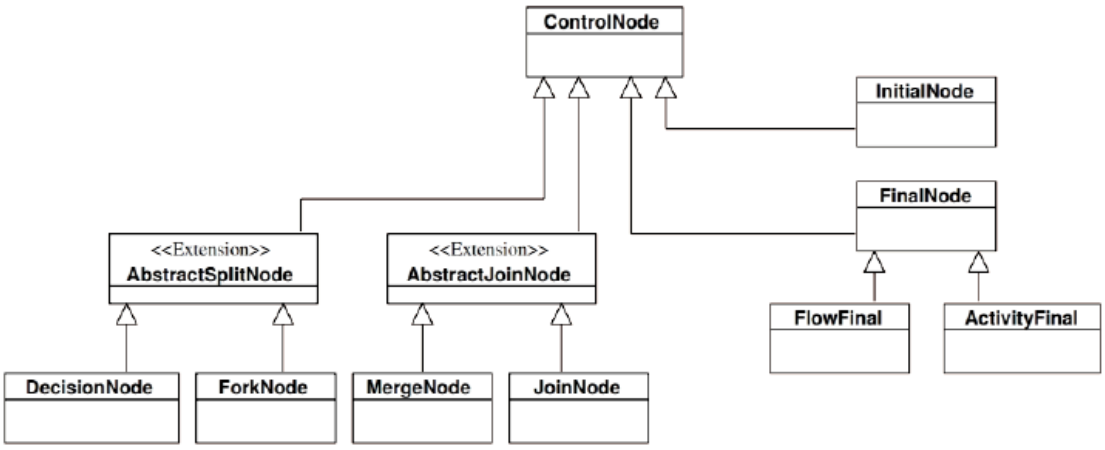


Figure A.4 Les nœuds de contrôle (controlNode)

La figure A.5 présente les noeuds d'action et d'objet, une action peut avoir des ensembles d'arcs d'activité entrants et sortants qui spécifient le flux de contrôle et les flux de données depuis et vers d'autres noeuds. Et Un noeud d'objet est une méta-classe abstraite permettant de d´efinir les flux d'objets dans les diagrammes d'activité. Il existe plusieurs types de noeuds d'objet : Broche (pin), Noeud paramètre d'activité (activity parameter node), Nœud central de mémoire tampon (central buffer node) et Noeud d'expansion (expansion node).

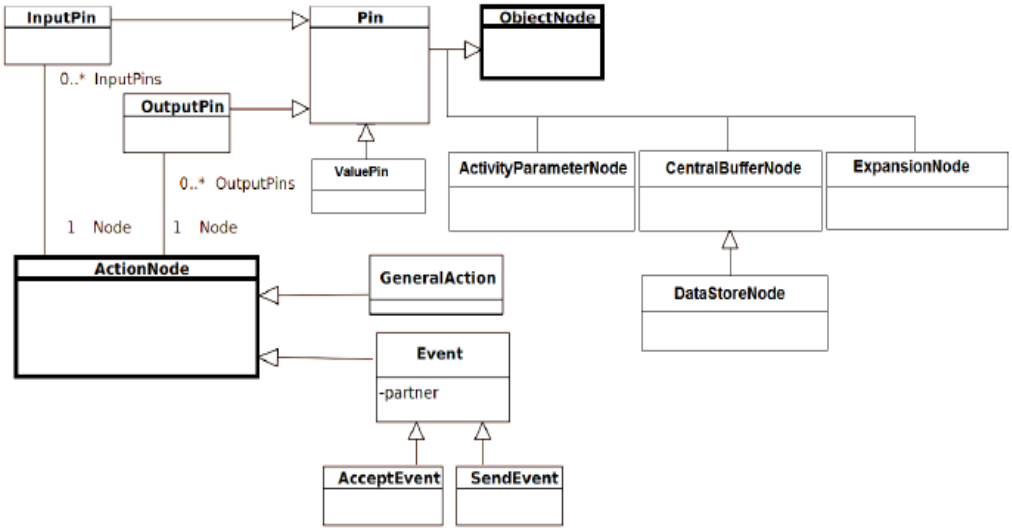


Figure A.5 Les nœuds d'action et d'objet (ActionNode, ObjectNode)

Dans la figure A.6 représente l'action d'appelé (Call Action) qui contient les types de comportement d'appel (Call behavior) et d'opération d'appel (Call opération).

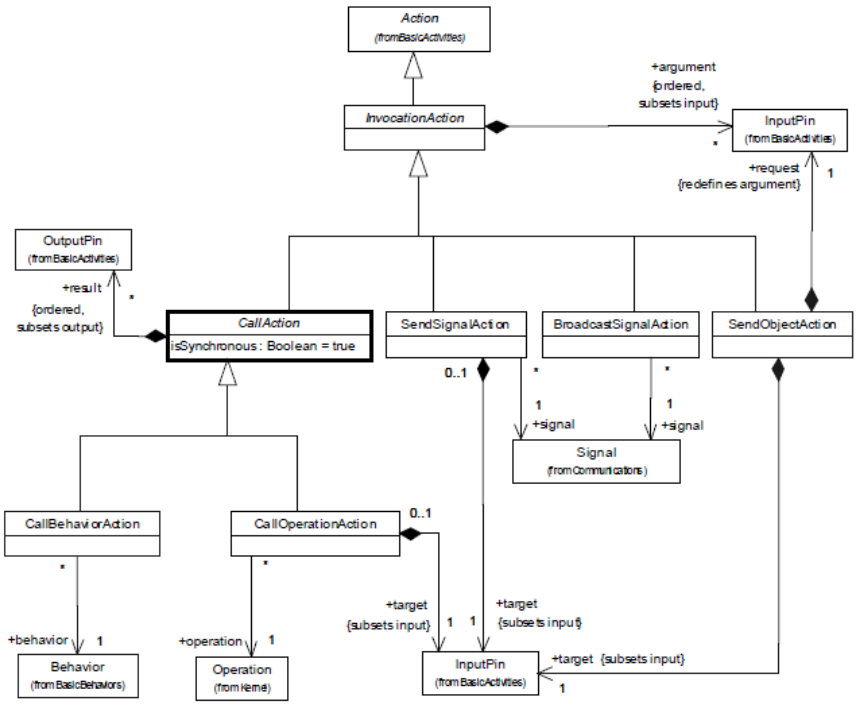
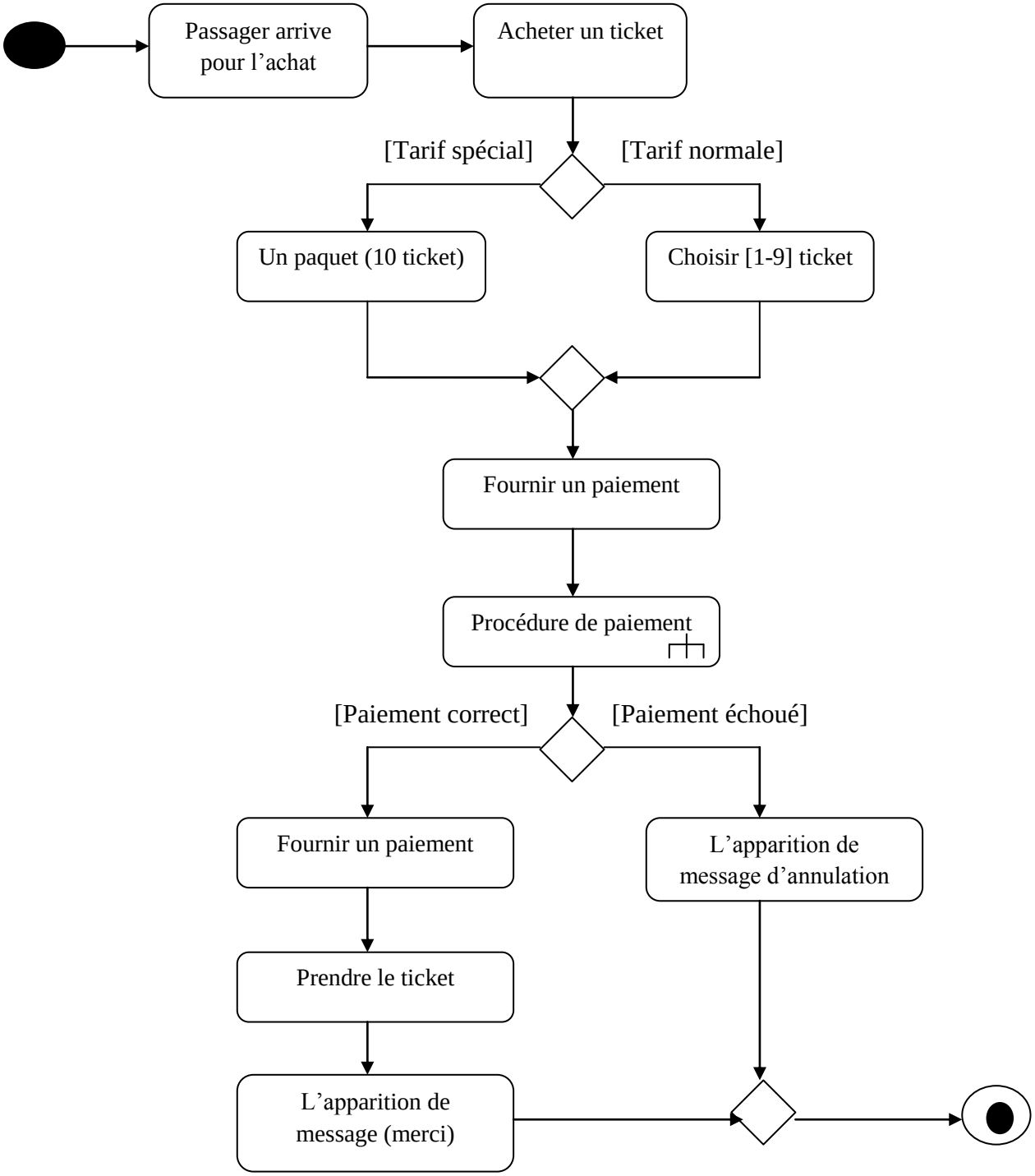


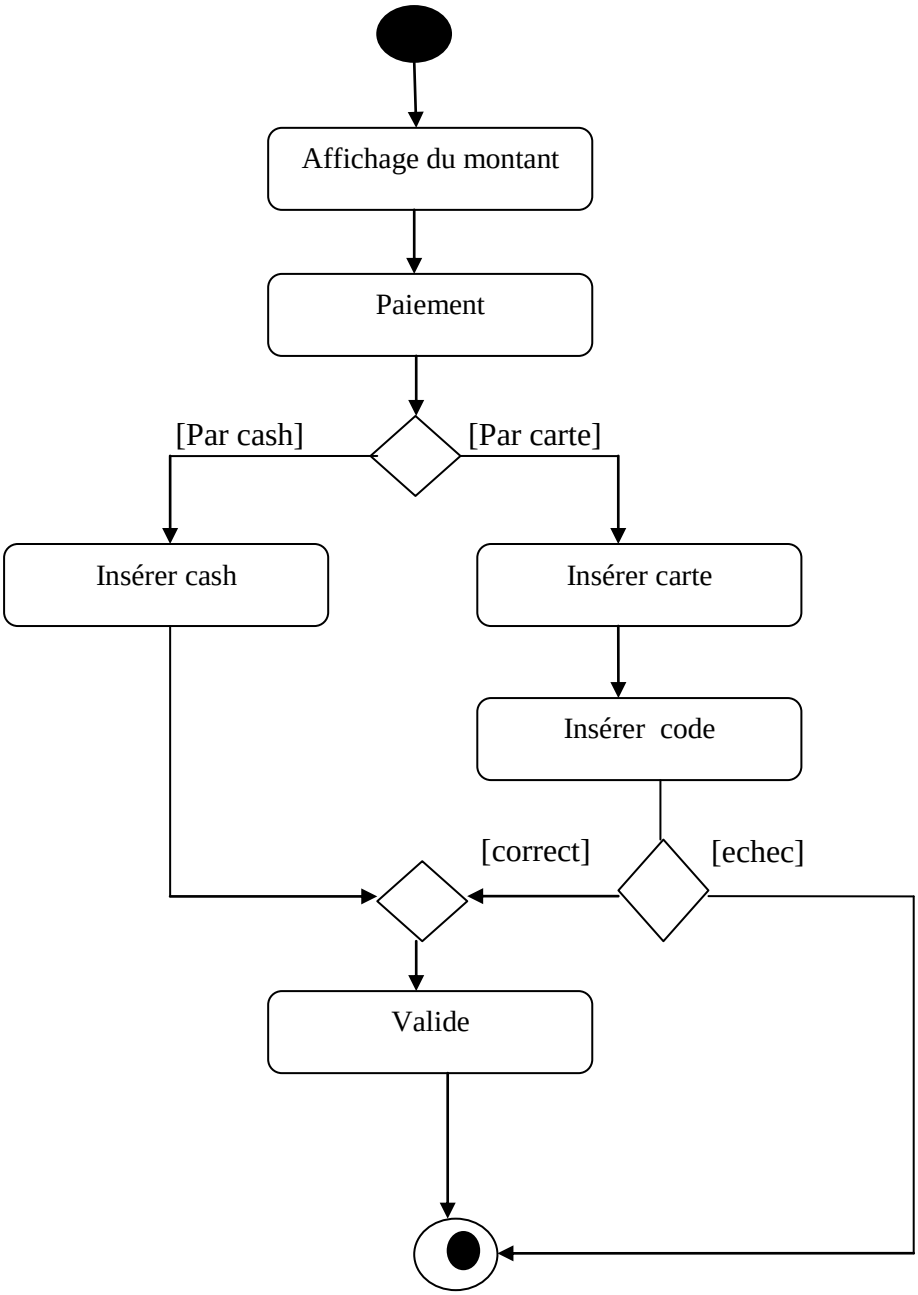
Figure A.6 L'action d'appelé (callAction)

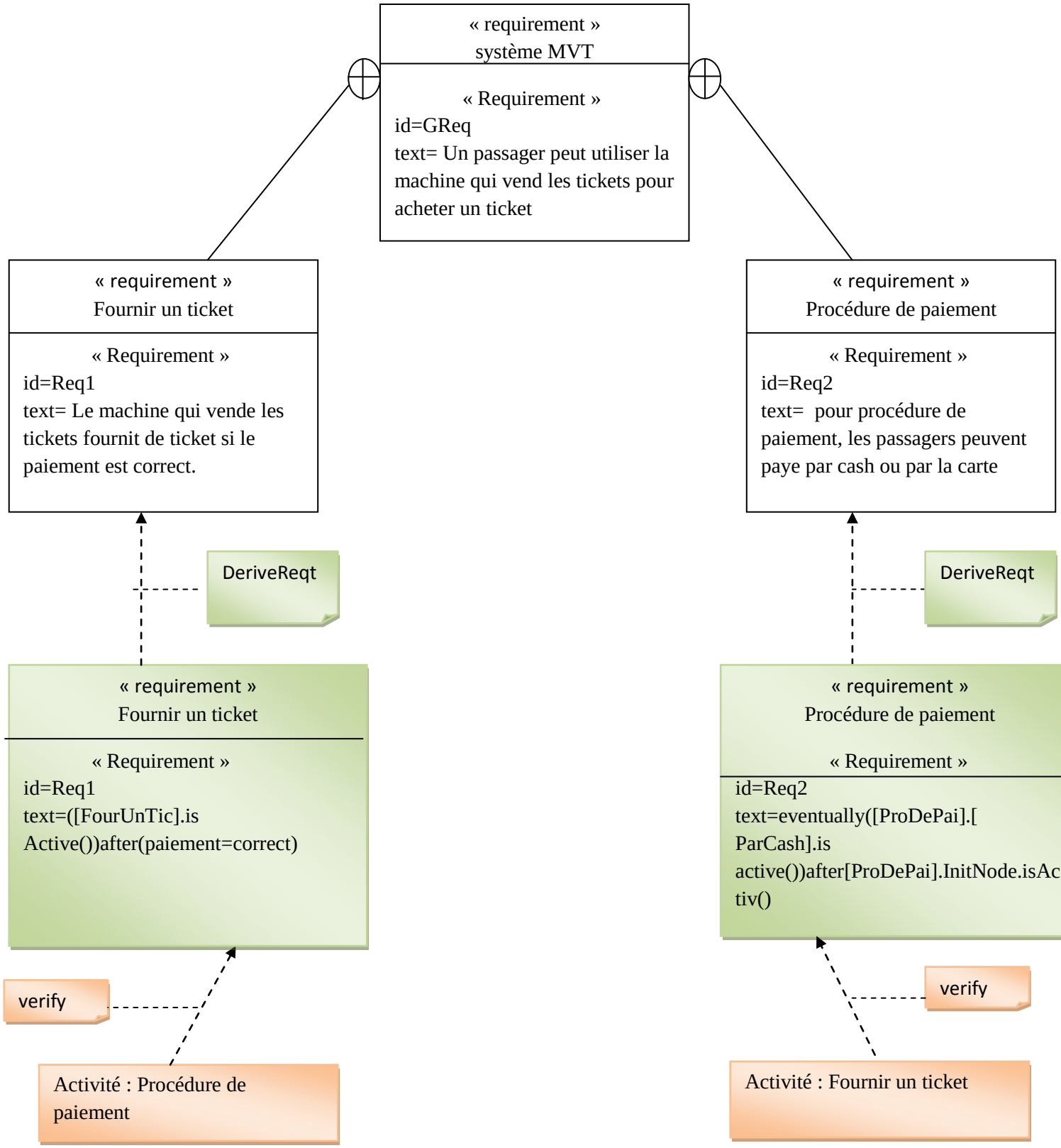
Machine qui vend les tickets (MVT)



la machine qui vend les tickets de métro

Procédure de paiement




« requirement »
système MVT
« Requirement »
id=GReq
text= Un passager peut utiliser la machine qui vend les tickets pour acheter un ticket
« requirement »
Fournir un ticket
« Requirement »
id=Req1
text= Le machine qui vende les tickets fournit de ticket si le paiement est correct.
« requirement »
Procédure de paiement
« Requirement »
id=Req2
text= pour procédure de paiement, les passagers peuvent paye par cash ou par la carte
DeriveReqt
DeriveReqt
« requirement »
Fournir un ticket
« Requirement »
id=Req1
text=([FourUnTic].is Active())after(paiement=correct)
« requirement »
Procédure de paiement
« Requirement »
id=Req2
text=eventually([ProDePai].[ParCash].is active())after[ProDePai].InitNode.isActiv()
verify
verify
Activité : Procédure de paiement
Activité : Fournir un ticket

Req1 :

[](paiement=echoue->[](!([VendeTickets].[FourUnTic].is active())))

Req2:

[](![ProDePai].InitNode.isActiv()<>([ProDePai].InitNode.isActiv() and ([ProDePai].[ParCash].is active()))