الجمهوريـة الجزائـريـة الديمقراطيـة الشعبيـة
République Algérienne Démocratique et Populaire
وزارة التعليـم العالـي والبحـث العلمـي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

N° Réf :……

Centre Universitaire
Abd Elhafid Boussouf Mila

Institut des Sciences et Technologie **Département de Mathématiques et Informatique**

Mémoire préparé en vue de l'obtention du diplôme de Master

En : Informatique
Spécialité: Sciences et Technologies de l'Information et de la Communication (STIC)

Le Développement d'un Outil de Transformation des Modèles Orientés Aspect vers les Réseaux de Pétri, Basé sur la Transformation De Graphes

Préparé par : Terchi Roqiya
Mezahi Bochra

Devant le jury :

Djaaboub Salim	**(MCB)**	**C.U.Abd Elhafid Boussouf**	**Président**
Aouag Mouna	**(MCB)**	**C.U.Abd Elhafid Boussouf**	**Rapporteur**
Hettab Abdelkamel	**(MAA)**	**C.U.Abd Elhafid Boussouf**	**Examinateur**

Année Universitaire : 2017/2018

REMERCIEMENT

Nous remercions d'abord et avant tout ***Allah*** *qui nous adonné le courage et la patience pour réaliser ce travail.*

Un remerciement particulier à notre encadreur ***Aouag Mouna*** *pour sa présence, son aide et surtout pour ses précieux conseils qui nous ont assistés pour l'accomplissement de notre projet,*

Nous adressons également nos vifs remerciements aux membres du jury qui ont bien voulu et accepté d'examiner ce modeste travail,

Sans oublier tous les enseignants qui ont contribué à notre formation durant notre vie Scolaire particulièrement les enseignants de notre institut.

Finalement, nous remercions très sincèrement tous nos famille pour leur encouragement sans limite.

Bouchra et Roqiya

Dédicace

Je dédie ce modeste travail:

A`
mes parents ***Abdelhamid et Sakina****, pour leurs encouragements*

A`
mes sœurs: ***Sabrina et Rahma et Souha,***

A`
mon frère : ***Yasser,***

À
mon binôme : ***Roqiya,***

A`
notre encadreur ***Aouag Mouna*** *et à nos enseignants.*

A`
toute ma famille,

A`
mes très chères amies.

Bouchra

Dédicace

Je dédie ce modeste travail:

A`
mes parents ***Mehieddine et Nacira****, pour leurs encouragements*

A`
mes sœurs: ***Asma et Hidaya,***

A`
mes frères : ***Mohamed et Alaeddine,***

À
mon binôme : ***Bouchra,***

A`
notre encadreur ***Aouag Mouna*** *et à nos enseignants.*

A`
toute ma famille.

Roqiya

Résumé

La Modélisation Orientée Aspect (MOA) est un nouveau paradigme développé par des scientifiques pour améliorer et résoudre les limites de l'orienté objet. Il existe plusieurs diagrammes UML2.0 qui sont orientés aspect, mais le problème de ces dernier est l'absence d'une sémantique formelle.
Pour cela, nous avons proposé une approche de transformation des diagrammes UML 2.0 orientés aspect vers des méthodes formelles.
Dans ce mémoire, nous avons présenté une transformation de Diagramme de Classes Orienté Aspect (DCOA)vers les réseaux de pétri (RdP) nous nous basant sur la transformation de graphes. Notre approche consiste à proposer une grammaire de graphes qui contient un ensemble de règles pour la transformation entre deux formalismes différents. Nous avons réalisé ce travail à l'aide de l'outil AToM, par la définition de les deux métas modèles (l'un pour le diagramme de classes orienté aspect et l'autre pour les réseaux de pétri) et la grammaire de graphes.
Finalement nous terminons notre contribution par une étude de cas bien illustrée, et nous représentons les résultats qui agrémentent notre approche.

Mots clés : UML 2.0 , DCOA, RdP, MOA, MDA, AToM^3

Abstract

Aspect-Oriented Modeling(AOM) is a new paradigm that developed by scientists to improve and resolve the limits of the object-oriented .There are several diagrams of UML 2.0 that are aspect-oriented, but the problems of theme are the absence of a formal semantics.
For this, we have proposed an approach for transforming UML 2.0 aspect oriented diagrams to formal methods.
In this memory, we presented the transformation of Aspect Oriented Class Diagram (AOCD) to Petri Net (PN), based on graph transformation. Our approach consists in proposing a graph grammar rules for the transformation between two different formalisms. We realized this work using the AToM^3 tool, to represent the rules of the transformation and meta- models (one for the Aspect Oriented Class Diagram and the other for petri net).
Finally, our contribution is will illustrated by a case study, and we have achieved a good result to validate our approach.

Key words: UML 2.0, AOCD, PN, AOM, MDA, AToM^3

مـلخــص

تصميم المظهر الموجه (orientée aspect) هو نموذج جديد طوره العلماء لتحسين وتصحيح حدود الشيء الموجه. هناك العديد من مخططات UML 2.0 الموجهة نحو الجوانب ، ولكن مشاكل هذه هي عدم وجود معنى.

لهذا ، اقترحنا طريقة لتحويل الرسوم البيانية لمخططات UML 2.0 الموجهة إلى شبكة بيتري (RdP).

في هذه المذكرة ، قدمنا منهجا من أجل تحويل الرسومات البيانية لمخططات UML 2.0 نحو شبكة بيتري (RdP) ، يتمثل عملنا في اقتراح قواعد نحوية وقواعد التحويل بين شكلين مختلفين.

لقد قمنا بهذا العمل باستخدام أداة AToM³ ، بالإضافة إلى قواعد التحويل بين النموذجين(metas واحد من أجل الرسم التوضيحي Classes المظهر الموجه (orientée aspect) و الآخر لشبكة بيتري(RdP))

وأخيرًا ، قمنا بتجربة عملنا على مجموعة من الامثلة وأعطت نتائج جيدة.

الكلمات المفتاحية: UML2.0 ، DCOA، RdP، MOA، MDA، AToM 3

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

Introduction Générale

Sommaire

1.1 Introduction

Aujourd'hui, les systèmes informatique que ce soit simples ou complexes sont de plus en plus utilisés au niveau de différents domaines tel que (la médecine , léavionique,le domaine militaire, l'architecteur, etc ....). La modélisation des systèmes est une phase laborieuse pour la conception et la validation des systèmes qui est l'objet de notre mémoire de fin d'étude .
En plus ,il existe plusieurs langage de modélisation par exemple (UML(unified modéling language) ,Merise ...etc).Dans notre travail, nous avons appliqué la modélisation

en utilisant les diagrammes UML 2.0 ,qui permet de modéliser les systèmes en utilisent treize diagrammes divisées en deux groupes statique et dynamique.

1.2 La problématique

Les diagrammes d'UML 2.0 sont des diagrammes orientés objet, ces derniers a prouver ces importance. Mais avec l'évolution de l'informatique ce paradigme a trouver des limites tels que : (la duplication , l'enchainement et l'éparpillant du modèle). Pour cela les développeurs et les programmeurs on pensés de définir un nouveau paradigme qui permet de résoudre les problèmes de l'orienté objet.Ce paradigme appelé « le paradigme orienté aspect » .
La Modélisation Orientée Aspect (MOA) est une nouvelle méthodologie qui permet la séparation entre les modèles fonctionnels (base) et les modèles non fonctionnels (aspect) pour obtenir des modèles composés (intégrés).
Les diagrammes UML2.0 orientés aspect sont classés dans la catégorie des décrispation semi formelles , ils permettent à travers différents diagrammes de mieux maitriser le développement d'un système , l'inconvénient majeur des diagrammes UML2.0 orientés aspect est l'absence d'une sémantique formelle ,ce qui rend impossibles de faire la vérification du comportement des systèmes d'écrit par plusieurs diagrammes.

1.3 Contribution

Pour résoudre ces problèmes ,l'idée générale est de transformer les diagrammes UML 2.0 orientés aspect vers des méthodes formelles . Les méthodes formelles offrent un cadre mathématique au processus de développement .Parmi le grand nombre de techniques formelles qui ont déjà été proposées on a les Réseaux de Pétri(RdP) .Ce dernier offre un outil formel et une bonne représentation graphique qui permet de modéliser les systèmes discrets, la facette graphique des réseaux de pétri, nous aide à comprendre aisément le système modélisé. Par ailleurs ,leur puissance d'expression mathématique permet de simuler des activités dynamiques.
La transformation des diagrammes UML 2.0 orientés aspect vers des RdP nécessite une

connaissance sur Modèle Driven Architecture (MDA),MDA est le domaine qui met à disposition des outils , concepts et langage pour créer et transformer des modèles afin de mécaniser le processus que les ingénieurs suivent habituellement à la main. Le MDA se concentre sur une préoccupation plus abstrait que la programmation classique ce qui permet d'obtenir plusieurs amélioration dans le développement de système complexe grâce à l'utilisation importante des modèles et des transformations automatique entre les modèles.
Notre travail se situe donc dans le contexte de la transformation des modèles .Plus précisément, il s'agit de transformation des diagrammes de classes orientés aspect vers les réseaux de pétri ,notre objectif est alors de proposer ,une approche basée sur la transformation de graphes pour générer un réseau de pétri à partir des diagrammes de classes orientés aspect d'une manière automatique. Pour atteindre cet objectif nous définissons des méta-modèles pour les modèles d'entrée ,sortie et une grammaire de graphes .Ces derniers sont implémenté avec l'outil Atom^3 et le langage de programmation Python.

1.4 Organisation de mémoire

Nous avons organisé notre mémoire comme la suite :
Chapitre 01 : Un aperçu sur la modélisation nous intéressons sur ULM 2.0 et nous représentons le différent diagramme d'UML 2.0 ainsi que la modélisation orienté aspect avec ces différent concepts de base.
Chapitre 02 : Sera consacré pour le formalisme de réseau de pétri avec ses propriétés et sa structure fondamentale permettent la modélisation .
Chapitre 03 : Sera dédie à la transformation de modèles à l'aide de transformation de graphes. Nous donnons un rappel sur l'architecture dirigée par les modèle ou MDA modèle driven architecture),avec une brève présentation de la théorie de transformation de graphes ,ainsi nous présentons l'environnement de travail Atom^3.
Chapitre 04 : Ce chapitre est organisé en deux grandes parties. La première est consacrée à la définition des règles de transformation qui constituant notre grammaire de graphes. La deuxième présentera une étude de cas qui permet de transformer le dia-

gramme de classe orienté aspect vers les réseaux de pétri d'une manière automatique.

Finalement, une conclusion résumant ce qui a été réalisé tout au long de notre travail.

CHAPITRE 2

La Modélisation Orientée Aspect

Sommaire

Introduction

La modélisation en informatique est l'étape la plus importante dans le développement d'un logiciel. Elle facilite la compréhension du fonctionnement d'un système avant sa réalisation en produisant un modèle. Ils existent plusieurs méthodes de modélisation comme la Modélisation Orientée Aspect(MOA) et la Modélisation Orientée Objet(MOO).

La modélisation orientée objet est un paradigme a une grande importance dans la résolution des problèmes complexe, mais avec l'évolution de l'information ce paradigme a trouvé des limites et n'a donné aucune solution tels que : (la duplication, l'enchainement et l'éparpillant du modèle).

Pour cela les développeurs et les programmeurs ont pensés de définir un nouveau paradigme qui permet de résoudre les problèmes de la modélisation orientée objet ce paradigme appelé « La modélisation orientée aspect ».

Dans ce chapitre nous commençons par des définitions de base de la modélisation avec ces différents types. Par la suite, nous présentons UML2.0 comme un langage de modélisation orientée objet. Nous présentons notre travail qui consiste la modélisation orientée aspect.

2.1 La modélisation

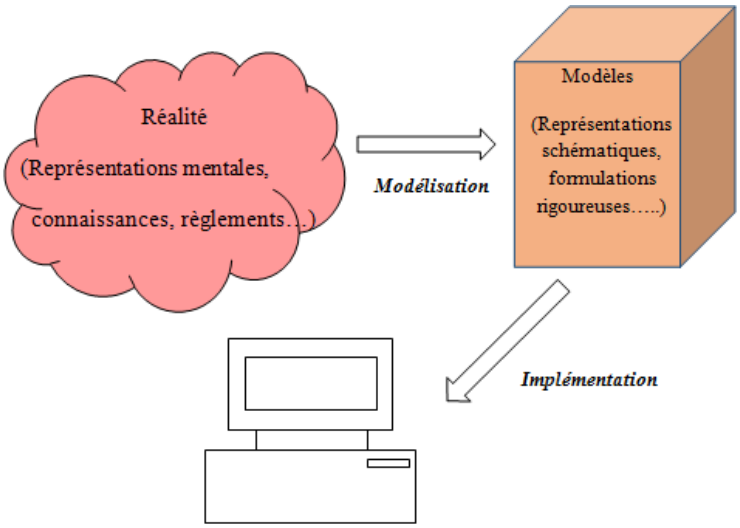

FIGURE 2.1 – Représentation de modélisation.

2.1.1 Définition d'un modèle

Un modèle est une abstraction d'un système construite dans un but précis. On dit alors que le modèle représente le système. Un modèle est une abstraction dans la mesure où il contient un ensemble restreint d'informations sur un système. Il est construit dans un but précis et les informations qu'il contient sont choisies pour être pertinentes vis-à-vis de l'utilisation qui sera faite du modèle. [Muller, 2006]

2.1.2 Définition de la modélisation

La modélisation est le processus de spécification du modèle, elle consiste à représenter le système sous forme de modèles en utilisant des concepts prédéfinis dans un langage de modélisation. La modélisation est une des tâches les plus importantes dans le processus de développement d'un système. La phase consacrée à l'analyse peut être considérée comme plus stratégique que celles dévolues à la conception et l'implémentation proprement dites. Il faut en effet fondamentalement représenter, comprendre et identifier les exigences du système afin de concevoir puis d'implémenter une applica-

tion stable et performante. Avec l'augmentation de la complexité des systèmes à élaborer, le choix d'une méthode de développement appropriée se révèle primordial pour le succès des travaux. Pour cela, il existe plusieurs orientations disponibles, c'est-à-dire des approches différentes pour comprendre, représenter, analyser et concevoir un système. Le problème sera décomposé en plusieurs modèles différents liés entre eux. [Wautelet et al, 2013]

2.1.3 Principes de la modélisation

L'histoire de la modélisation dans tous les domaines technologique est riche et l'expérience acquise dicte quatre principes de base [Menasra et Cherafa, 2013] :

• Les modèles doivent être choisis avec soin car, s'ils sont bons, ils feront ressortir les problèmes de développement les plus ardus et apporteront un éclaircissement inespéré, alors que s'ils sont mauvais, ils détourneront l'attention des développeurs des questions fondamentales.

• Tous les modèles peuvent avoir déférents niveaux de précision. Les meilleurs modèles sont ceux qui laissent le choix des niveaux de détails, en fonction du point de vue envisagé et du besoin, alors qu'un développement mettra l'accent sur leur réalisation. Tous veulent visualiser le système avec une précision variable et à des phases de développement différents.

• Les meilleurs modèles ne perdent pas le sens de la réalité. Toute modélisation simplifie la réalité, mais ces simplifications ne doivent dissimuler aucun détail important. Il est donc préférable de disposer de modèles qui restent en contact avec la réalité et de savoir exactement quand ils s'en éloignent.

• Aucun modèle n'est suffisant à lui seul, il est préférable de décomposer un système important en un ensemble de petits modèles presque indépendant. En effet, cela signifie qu'ils peuvent être construits et étudiés séparément, mais qu'ils doivent garder une interdépendance.

2.1.4 Objectif de la modélisation

La modélisation permet de mieux comprendre le système à développer. Donc il permet de [Laouar, 2013] :

- Visualiser le système comme il est ou comme il devrait l'être.
- Valider le modèle vis-à-vis des clients.
- Spécifier les structures de données et le comportement du système.
- Fournir un guide pour la construction du système.
- Documenter le système et les décisions prises.

2.1.5 Différent types de modélisation

La modélisation peut être classée selon le degré du formalisme des langages ou des méthodes utilisés. Ainsi, la modélisation peut être considérée comme étant formelle, semi-formelle ou informelle. La table de la figure 2.2 ci-dessous présente une définition des catégories de langages ainsi que des exemples de langages ou des méthodes utilisées dans la modélisation. [Dehimi, 2014]

Catégories de langages			
Langage Informel		Langage Semi-Formel	Langage Formel
Simple	Standardisé		
Langage qui n'a pas un ensemble complet de régles pour restreindre une construction.	Langage avec une structure, un format et des régles pour la composition d'une construction.	Langage qui a une syntaxe définie pour spécifier les conditions sur lesquelles les constructions sont premise.	Langage qui posséde une syntaxe et une sémantique définies rigoureusement.Il existe un modéle théorique qui peut étre utilisé pour valider une consultation .
Exemples de Langage ou Méthodes			
Langage naturel.	Texte structuré en langage naturel .	Diagramme entité-relation , Diagramme à objet	Réseaux de pétri , machines à états finis, VDM,Z

FIGURE 2.2 – Classification et utilisation de langages ou de méthodes.

2.1.5.1 Modélisation informelle

Le processus de modélisation informelle à base de langages informels :

- La facilité de compréhension du langage informel, qui permet des consensus entre les personnes qui spécifient et celles qui commandent un logiciel.
- Le langage informel représente une manière familière de communication entre les personnes.

Par contre, l'utilisation d'un langage informel rend la modélisation imprécise et parfois ambigüe. Toute tentative de standardisation est difficile, vue le caractère informel de cette approche. Néanmoins, Il est possible d'utiliser une modélisations informelle plus ou moins standardisée pour restreindre ce problème, c'est-à-dire une modélisation qui utilise un langage naturel tout en introduisant des règles d'utilisation de ce langage dans la construction de la modélisation. Un tel type de modélisation garde les avantages de la modélisation informelle en la rendant moins imprécise et moins ambigüe. [Dehimi, 2014]

2.1.5.2 Modélisation semi-formelle

Le processus de modélisation semi-formelle est basé sur un langage textuel ou graphique pour lequel une syntaxe précise est définie .Ce type de modélisation permet d'effectuer des contrôles et de réaliser des automatisations pour certaines tâches, bien que la sémantique des langages semi-formels soit souvent assez faible. Les méthodes de modélisation semi-formelles dans leur majorité, s'appuient fortement sur des langages graphiques. Cela, se justifie par l'expressivité que peut avoir un modèle graphique bien développé. Le langage textuel est utilisé normalement comme complément aux modèles graphiques.

Par ailleurs, la modélisation semi-formelle (tels que : UML) utilise fortement les langages graphiques, ceci permet la production de modèles assez faciles à interpréter. Néanmoins, les aspects sémantiques impliqués dans cette approche de modélisation souffrent d'une remarquable déficience . [Dehimi, 2014]

2.1.5.3 Modélisation formelle

Les méthodes formelles sont basées sur des techniques mathématiques pour la spécification, le développement et la validation des systèmes. L'utilisation des méthodes formelles est importante pour assurer la sécurité des systèmes. Les études pour la création d'autres outils formels qui couvrent toute la spécification, outre les outils de preuves déjà existants, montrent l'intérêt qu'on porte à ces méthodes. Bien qu'elles ne soient pas utilisées dans tous les types de modélisation, L'application des méthodes formelles, plus que souhaitable, commence à être imposée par des sociétés dans certains cas, qui nécessitent plus de rigueur. [Dehimi, 2014]

2.2 La modélisation orientée objet

2.2.1 Définition de modélisation orientée objet

La modélisation orientée objet a un rôle importante dans la conception et la modélisation de systèmes complexes. La modélisation et la conception orientée objet constitue une façon de penser les problèmes en appliquant des modèles organisés autour de concepts du monde réel. Le concept fondamental est l'objet qui combine à la fois une structure de données et un comportement. Les modèles orientés objet permettent de comprendre des problèmes, de communiquer avec des experts du domaine d'application, de modéliser les métiers d'une entreprise, de réparer la documentation et de concevoir des programmes et de bases de données [Aouag, 2014]. Ils existent plusieurs langages de modélisation qui sont orientés objet, parmi lesquelles on trouve UML 2.0 (Unified Mödling Language).

2.2.2 UML 2.0

UML2.0 (Unified Modeling Language) est un langage de modélisation graphique utilisé pour la spécification, la construction, et la documentation des artefacts d'un système. UML2.0 est utilisé pour la définition des modèles dans le processus de développement du système, afin de présenter les éléments du système et leur communication.

UML2.0 est non seulement un outil graphique intéressant mais une norme qui s'impose en technologie à objets et à laquelle se sont rangés tous les grands acteurs de domaine. Il fournit les éléments permettant de construire le modèle qui sera le langage du projet. C'est un outil de modélisation graphique commun qui permet de représenter et de communique les divers aspects d'un système d'information [Zahoui, 2014]. UML2.0 est un langage qui permet de modéliser non seulement des applications informatiques ou des structures de données, mais également les activités d'un domaine : mécanique, biologie, processus métier,...

La Figure (2.3) montre les déférentes étapes par les quelles UML2.0 est passé :

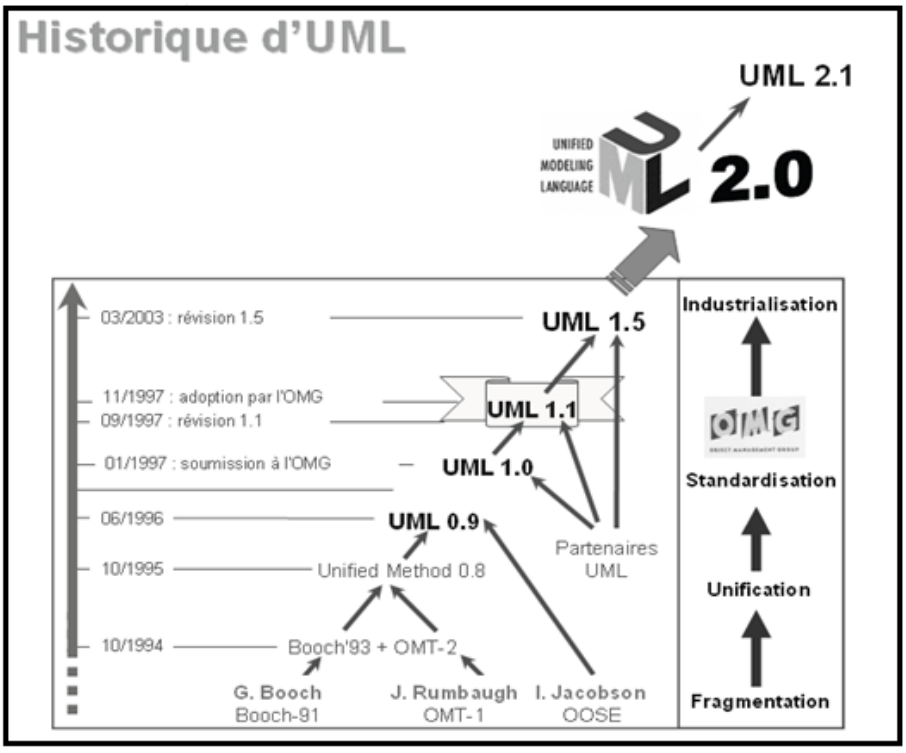

FIGURE 2.3 – Historique d'UML2.0.

2.2.3 Diagrammes UML2.0

UML2.0 puise des avantages de la modélisation semi-formelle en offrant à l'utilisateur un moyen de visualiser et de manipuler des éléments de modélisation. Les

éléments de représentation sont le plus souvent des graphes connexes où les sommets correspondent aux éléments et les arcs aux relations. Ces graphes servent à visualiser un système sous différentes vues. UML2.0 propose un certain nombre de diagrammes qui servent à visualiser un système sous différentes perspectives. Pour les systèmes complexes, un diagramme ne représente qu'une vue partielle des éléments qui composent ces systèmes. UML2.0 comporte un ensemble de diagrammes(treize diagrammes) représentant des vues distinctes pour modéliser des concepts particuliers du système. Ils se répartissent en deux grands groupes

A. Diagrammes Structurels :

Ces diagrammes permettent de visualiser, spécifier, construire et documenter l'aspect statique ou structurel du système informatisé. [Roques et vallée , 2007]

A.1 Diagramme d'objets : Le diagramme d'objets sert à illustrer des structures de classes compliquées en montrant des exemples d'instances. Ce diagramme est utilisé en analyse pour vérifier l'adéquation d'un diagramme de classes à différents cas possibles.

A.2 Diagramme de packages : Le diagramme de packages est l'officialisation par UML 2.0 d'une pratique d'UML 1.x qui consiste à utiliser un diagramme de classes pour y représenter la hiérarchie des modules (catégories) d'un projet.

A.3 Diagramme de structure composite : Le diagramme de structure composite décrit la composition d'un objet complexe lors de son exécution. Ce diagramme est propre à UML 2.0 ; il introduit la notion de structure d'un objet complexe, tel qu'il se présente en phase de run-time.

A.4 Diagramme de composants : Le diagramme de composants représente les concepts connus de l'exploitant pour installer et dépanner le système. Il s'agit dans ce cas de déterminer la structure des composants d'exploitation que sont les librairies dynamiques, les instances de bases de données, les applications, les progiciels, les objets distribués, les exécutables, etc.

A.5 Diagramme de déploiement : Le diagramme de déploiement correspond à la fois à la structure du réseau informatique qui prend en charge le système logiciel, et la façon dont les composants d'exploitation y sont installés.

B. Diagrammes Comportementaux :

Les diagrammes comportementaux modélisent les aspects dynamiques du système. Ces aspects incluent les interactions entre le système et ses différents acteurs, ainsi que la façon dont les différents objets contenus dans le système communiquent entre eux. [Roques, 2006]

B.1 Diagramme de cas d'utilisation : Un diagramme de cas d'utilisation (use case) représente un ensemble de séquences d'actions qui sont réalisées par le système et qui produisent un résultat observable intéressant pour un acteur particulier. Un cas d'utilisation modélise un service rendu par le système. Il exprime les interactions acteurs/système et apporte une valeur ajoutée « notable » à l'acteur concerné.

B.2 Diagramme de vue d'ensemble des interactions : fusionne les diagrammes d'activité et de séquence pour combiner des fragments d'interaction avec des décisions et des flots.

B.3 Diagramme de séquence : Il représente séquentiellement le déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs. Le diagramme de séquence peut servir à illustrer un cas d'utilisation.

B.4 Diagramme de communication : C'est une représentation simplifiée d'un diagramme de séquence, en se concentrant sur les échanges de messages entre les objets.

B.5 Diagramme de temps : fusionne les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps et les messages qui modifient cet état.

B.6 Diagramme états-transitions : Le diagramme d'états représente le cycle de vie commun aux objets d'une même classe. Ce diagramme complète la connaissance des classes en analyse et en conception en montrant les différents états et transitions possibles des objets d'une classe à l'exécution.

B.7 Diagramme d'activité : représente les règles d'enchaînement des actions et décisions au sein d'une activité. Il peut également être utilisé comme alternative au diagramme d'états pour décrire la navigation dans un site web.

La figure 2.4 montre la hiérarchie des diagrammes d'UML2.0.

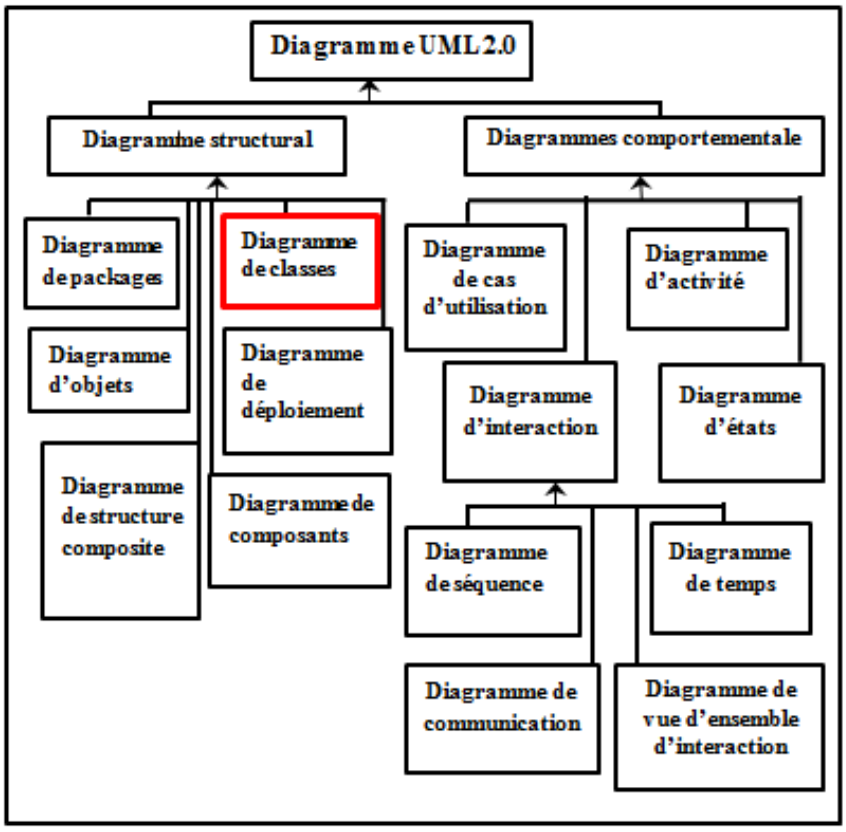

FIGURE 2.4 – La hiérarchie des diagrammes d'UML2.0.

Parmi les diagrammes UML2.0, nous sommes intéressés par le diagramme de classes pour réaliser notre travail.

2.2.4 Les diagrammes de classes

Le diagramme de classes est le point central dans un développement orienté objet. En analyse, il a pour objectif de décrire la structure des entités manipulées par les utilisateurs, en conception, le diagramme de classes représente la structure d'un code orienté objet ou, à un niveau de détail plus important et les modules du langage de développement. [Roques, 2006]
D'autre part le diagramme de classes constitue l'un des pivots essentiels de la modélisation avec UML. En effet, ce diagramme permet de donner la représentation statique

du système à développer. Cette représentation est centrée sur les concepts de classe et d'association. Chaque classe se décrit par les données et les traitements dont elle est responsable pour elle-même et vis-à-vis des autres classes. Les traitements sont matérialisés par des opérations. Le détail des traitements n'est pas représenté directement dans le diagramme de classe ; seul l'algorithme général et le pseudo-code correspondant peuvent être associés à la modélisation. La description du diagramme de classe est fondée sur :

- le concept d'objet,
- le concept de classe comprenant les attributs et les opérations,
- les différents types d'association entre classes. [Gabay et Gabay, 2008]

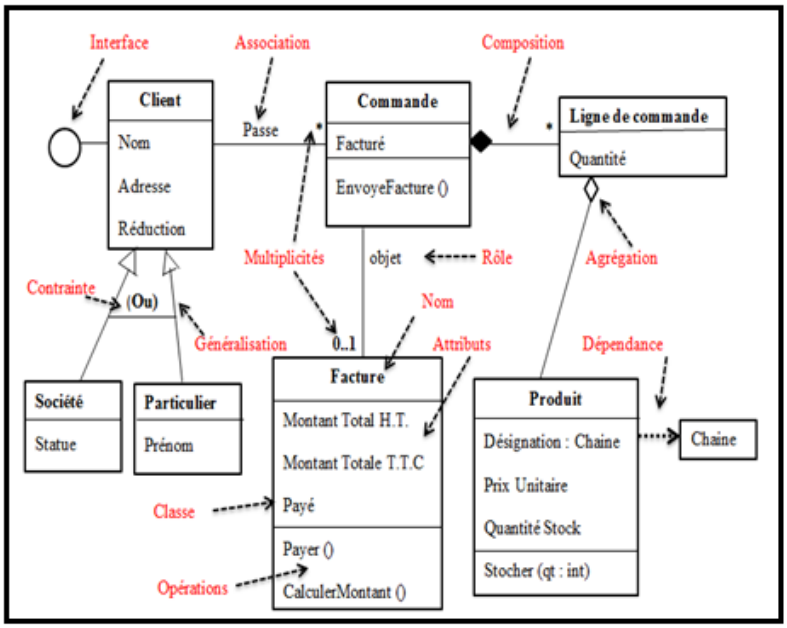

FIGURE 2.5 – Diagramme de classes.

Le diagramme des classes (Figure 2.5) contiennent les éléments suivants : des classes ; des attributs ; des relations d'association ; de composition ; Comme tous les autres diagrammes. Ils peuvent contenir des notes, des contraintes, des stéréotypes ou des étiquettes.

2.2.4.1 Classe et objet

Une classe : représente la description abstraite d'un ensemble d'objets possédant les mêmes caractéristiques. On peut parler également de type.

Un objet : est une entité aux frontières bien définies, possédant une identité et encapsulant un état et un comportement. Un objet est une instance (ou occurrence) d'une classe. [Roques, 2008]

2.2.4.2 Représentation d'une classe

La classe est définie par son nom, ses attributs et ses opérations comme ci-dessous. Étant donné que les classes vont être utilisées pour générer le code il est souhaitable d'utiliser une règle de nommage qui respecte les syntaxes des langages informatiques comme par exemple celle proposée par java qui consiste à (Figure 2.6)

- Commencer les noms des classes par des majuscules et tous les autres éléments par des minuscules .
- Séparer les mots composés des majuscules.
- Ne pas utiliser de caractères ou accentués qui pourraient ne pas être acceptés dans le langage informatique. [Menasra et Cherafa, 2013]

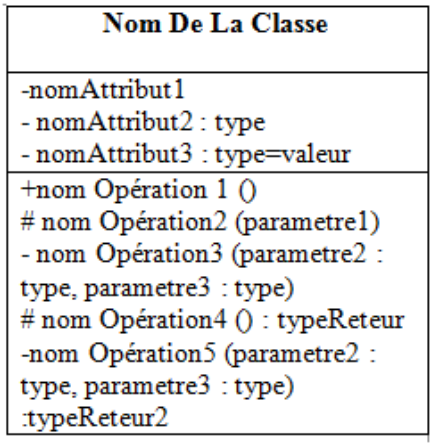

FIGURE 2.6 – Représentation graphique d'une classe.

2.2.4.3 Attributs de la classe

Les attributs définissent des informations qu'une classe ou un objet doivent connaître. Ils représentent les données encapsulées dans les objets de cette classe. Chacune de ces informations est définie par un nom, un type de données, une visibilité et peut être initialisé.Le nom de l'attribut doit être unique dans la classe. La syntaxe de la déclaration d'un attribut est la suivante :
<visibilité> [/] <nom_attribut> :
<type> ['['<multiplicité>']' [<contrainte>]] [= <valeur_ par_défaut>]
Le type de l'attribut (<type>) peut être un nom de classe, un nom d'interface ou un type de donné prédéfini. La multiplicité (<multiplicité>) d'un attribut précise le nombre de valeurs que l'attribut peut contenir. Lorsqu'une multiplicité supérieure à 1 est précisée, il est possible d'ajouter une contrainte (<contrainte>) pour préciser si les valeurs sont ordonnées. [Audibert, 2008]

2.2.4.4 Opération de la classe

Dans une classe, une opération doit être unique. Quand le nom d'une opération apparaît plusieurs fois avec des paramètres différents, on dit que l'opération est surchargée. En revanche, il est impossible que deux opérations ne se distinguent que par leur valeur retournée.
La déclaration d'une opération contient les types des paramètres et le type de la valeur de retour, sa syntaxe est la suivante :
<visibilité> <nom_opération > ([<paramètre1>, ... , <paramètreN>]) : [<type renvoyé>] [<Propriétés>].

- Visible : + public ; # protected ;− private .
- nom Opération : nom de l'opération.
    - [<paramètre1> ,. . . . <paramètreN>] : liste des paramètre .
    - [<type renvoyé>] : type de retour l'opération .
    - [<propriétés>] : contrainte(OCl) Object Constraint Language , indication ou mots clé comme «abstract» .

[<direction>] <nom-paramètre> :<type> ['['<multiplicité>']'] [=<valeur_ par_défaut>]
La direction peut prendre l'une des valeurs suivante :

- in : Paramètre d'entrée passé par valeur. Les modifications du paramètre ne sont pas disponibles pour l'appelant. C'est le comportement par défaut.
- out : Paramètre de sortie uniquement. Il n'y a pas de valeur d'entrée et la valeur finale est disponible pour l'appelant.
- in out : Paramètre d'entrée/sortie. La valeur finale est disponible pour l'appelant.

Le type du paramètre (<type>) peut être un nom de classe, un nom d'interface ou un type de donné prédéfini. [Audibert, 2008]

2.2.4.5 Relations entre classes

- **A. Association :**

Une association est une relation entre deux classes (association binaire) ou plus (association n-aire), qui décrit les connexions structurelles entre leurs instances. Une association indique donc qu'il peut y avoir des liens entre des instances des classes associées. [Audibert, 2008]

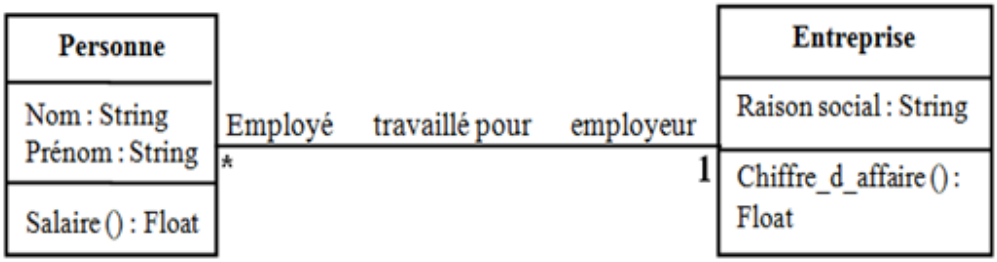

FIGURE 2.7 – Représentation d'association.

.

Multiplicité des associations : Chaque extrémité peut également porter une indication de multiplicité qui spécifie combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe. Les principales multiplicités normalisées sont décrites dans le table 2.1

Valeurs de multiplicité	Reformulation
0..1	Zéro ou un(e)
1	Un(e)unique
0..*ou*	Zéro ou plusieurs
1..*	Au mois un(e)
N	Exactement N (entier naturel)
N...M	Entre N et M (entier naturel)

TABLE 2.1 – Multiplicité des associations.

Relation d'association binaire : Une association binaire est matérialisée par un trait plein entre les classes associées .Elle peut être ornée d'un nom, avec éventuellement une précision du sens de lecture (ou) Quand les deux extrémités de l'association pointent vers la même classe, l'association est dite réflexive. [Audibert, 2008]

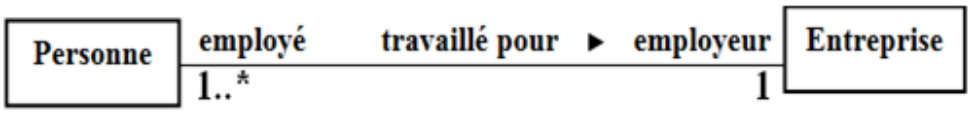

FIGURE 2.8 – Association binaire.

Relation d'association n-aire : Une association n-aire lie plus de deux classes. La ligne pointillée d'une classe-association peut être reliée au losange par une ligne discontinue pour représenter une association n-aire dotée d'attributs, d'opérations ou d'associations. On représente une association n-aire par un grand losange avec un chemin partant vers chaque classe participante (figure 2.9). Le nom de l'association, le cas échéant, apparaît à proximité du losange. [Audibert, 2008]

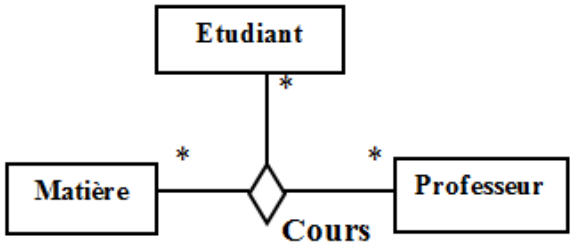

FIGURE 2.9 – Association n-aire.

• **B.Agrégation :**

Une agrégation est une association particulière. Elle représente la relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble. Dans la notation graphique d'UML, elle se distingue d'une association par l'ajout d'un losange vide du côté de l'agrégat (figure 2.10). La durée de vie de l'agrégat est indépendante des éléments qui le constituent : dans notre exemple, la destruction d'une instance de la classe 'Personne' n'implique pas la destruction des instances de la classe 'Immeuble'. Par ailleurs, un composant peut apparaître dans plusieurs agrégats : un immeuble peut appartenir à un ensemble de copropriétaires. [Alonso, 2009]

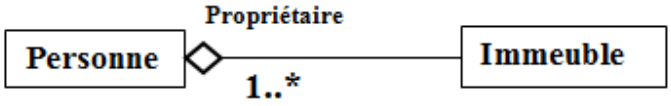

FIGURE 2.10 – Représentation UML d'une agrégation.

•**C.Composition :**

Une composition est une agrégation plus forte impliquant que la destruction de l'objet composite entraîne la destruction de ses composants. De plus une instance d'un composant appartient toujours à une instance de l'élément composite au plus. Une composition se distingue d'une association par l'ajout d'un losange plein du côté du composite (figure 2.11). [Alonso, 2009]

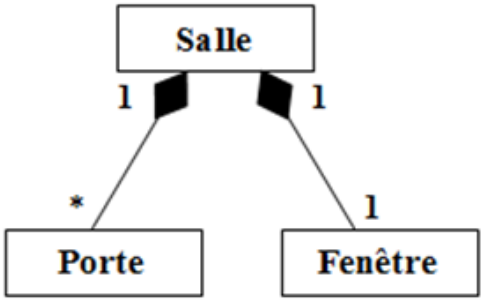

FIGURE 2.11 – Représentation UML d'une composition.

• **D.Dépendance :**

Une dépendance est une relation unidirectionnelle exprimant une dépendance sémantique entre des éléments du modèle. Elle est représentée par un trait discontinu orienté

.Elle indique que la modification de la cible peut impliquer une modification de la source. La dépendance est souvent stéréotypée pour mieux expliciter le lien sémantique entre les éléments du modèle). (Figure 2.12). [Audibert, 2008]

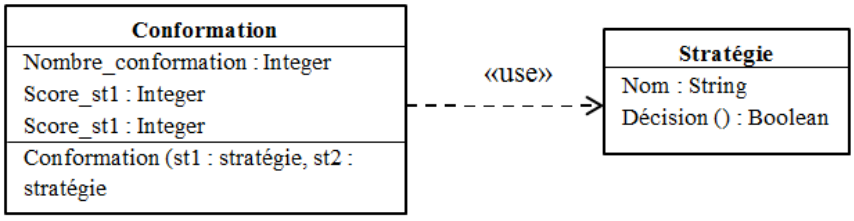

FIGURE 2.12 – Relation de dépendance.

2.2.4.6 Héritage

Un des intérêts de la modélisation orientée objet est la possibilité de manipuler des concepts abstraits. Ce principe d'abstraction est assuré par le concept d'héritage qui permet de simplifier le modèle en factorisant ses propriétés. L'abstraction consiste à définir une hiérarchie entre des éléments ayant des propriétés communes. Les propriétés communes sont rassemblées dans une super-classe, ou classe-parent, par le processus de généralisation et les propriétés spécifiques restent dans les sous-classes, ou classes-enfants, par le processus de spécification. L'héritage est représenté en UML au moyen d'une flèche orientée de la sous-classe vers la super-classe (figure 2.13). La liste suivante donne quelques propriétés de l'héritage :

- La classe-enfant possède toutes les propriétés de ses classes-parents mais elle n'a pas accès aux propriétés privées de celles-ci.
- Une classe-enfant peut redéfinir une ou plusieurs opérations de la classe-parent. Sauf indication contraire, un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes. La surcharge d'opérations (même nom, mais signature des opérations différentes) est possible dans toutes les classes.
- Toutes les associations de la classe-parent s'appliquent par défaut à ses sous-classes.
- Une instance d'une classe peut être utilisée partout où une instance de sa classe-parent est attendue (principe de la substitution). Par exemple, toute opération accep-

tant un objet d'une classe figure géométrique doit accepter tout objet de la classe rectangle (l'inverse n'est pas toujours vrai).

• Une classe peut avoir plusieurs classes-parents : on parle alors d'héritage multiple. [Alonso, 2009]

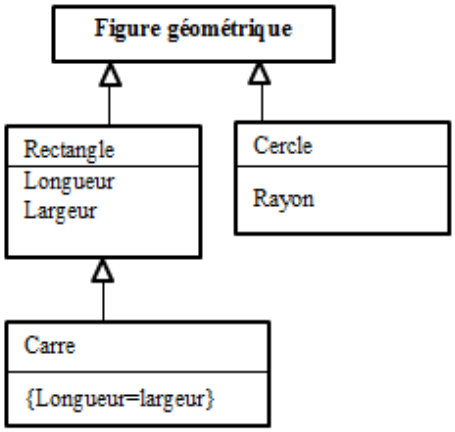

FIGURE 2.13 – Représentation UML d'une relation d'héritage.

2.2.4.7 Classe abstraite

Les classes abstraites ne sont pas instanciables directement : elles servent de spécification plus générale pour manipuler les objets instances d'une ou de plusieurs de leurs sous-classes. Par convention, le nom des classes abstraites est écrit en italique. Une classe abstraite se différencie d'une interface puisqu'elle peut posséder des attributs et que ses opérations peuvent posséder des implémentations. Une classe est abstraite lorsque au moins une des opérations qu'elle définit ou dont elle hérite est abstraite, c'est-à-dire qu'elle ne possède pas d'implémentation. [Alonso, 2009]

La figure 2.14 montre comment le procédé d'abstraction permet de compléter l'exemple précédent.

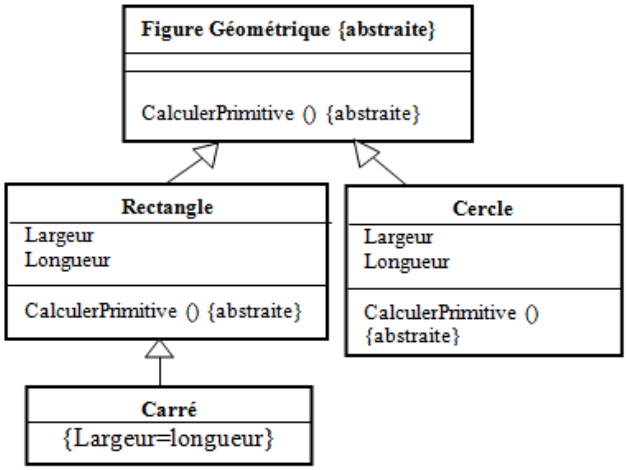

FIGURE 2.14 – Représentation UML d'une classe abstraite.

2.2.4.8 Package

Package : mécanisme général de regroupement d'éléments en UML, qui est principalement utilisé en analyse et conception objet pour regrouper des classes et des associations. Les packages sont des espaces de noms : deux éléments ne peuvent pas porter le même nom au sein du même package. Par contre, deux éléments appartenant à des packages différents peuvent porter le même nom. La structuration d'un modèle en packages est une activité délicate. Elle doit s'appuyer sur deux principes fondamentaux : cohérence et indépendance. Le premier principe consiste à regrouper les classes qui sont proches d'un point de vue sémantique. Un critère intéressant consiste à évaluer les durées de vie des instances de concept et à rechercher l'homogénéité. Le deuxième principe s'efforce de minimiser les relations entre packages, c'est-à-dire plus concrètement les relations entre classes de packages différents. [Roques, 2008]

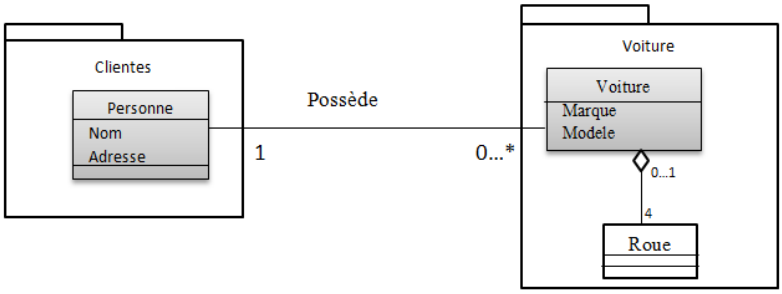

FIGURE 2.15 – Notation de package.

2.2.4.9 Interface

Une interface doit être réalisée par au moins une classe et peut l'être par plusieurs. Graphiquement, cela est représenté par un trait discontinu terminé par une flèche triangulaire et le stéréotype « realize ». Une classe peut très bien réaliser plusieurs interfaces. Une classe (classe cliente de l'interface) peut dépendre d'une interface (interface requise). On représente cela par une relation de dépendance et le stéréotype « use ». Attention aux problèmes de conflits si une classe dépend d'une interface réalisée par plusieurs autres classes. La notion d'interface est assez proche de la notion de classe abstraite avec une capacité de découplage plus grand. En C++ (le C++ ne connaît pas la notion d'interface), la notion d'interface est généralement implémentée par une classe abstraite. [Audibert, 2008]

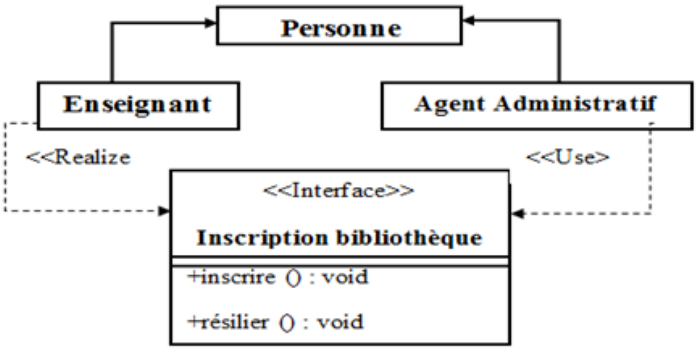

FIGURE 2.16 – Diagramme mettant en œuvre une interface.

2.2.4.10 L'Applications des diagrammes de classes

Les diagrammes de classes présentent plusieurs avantages pour n'importe quel type d'organisation. Essayez de les utiliser pour : Web

- Illustrer des modèles de données pour des systèmes d'information.
- Comprendre l'aperçu général des schémas d'une application.
- Exprimer les besoins d'un système et diffuser cette information dans toute l'entreprise.
- Créer des schémas détaillés qui mettent l'accent sur le code de programmation nécessaire pour mettre en œuvre la structure décrite.
- Fournir une description indépendante de la mise en œuvre des types utilisés dans un système et transmis entre ses composants.

2.2.5 Les avantages de la modélisation orientée objet

L'approche orientée objet possède une capacité d'intégration ou d'unification qui implique les avantages suivants : [Aouag, 2014]

- La stabilité de la modélisation par rapport aux entités du monde réel.
- La construction itérative facilitée par le couplage faible entre composants et la possibilité de réutiliser des éléments d'un développement à un autre.
- La simplicité du modèle qui fait appel à cinq concepts fondateurs (les objets,les messages,les classes, la généralisation et le polymorphisme).
- L'approche orientée objet est largement adoptée, tout simplement parce qu'elle a montré son efficacité lors de la construction de systèmes dans une diversité de domaines métier et qu'elle englobe les dimensions et les degrés de complexité.

2.2.6 Les limites de la modélisation orientée objet

Dans le paradigme orienté objet on trouve des limites pour les qu'elles ce paradigme n'a donné aucune solution à ce type de problème on peut citer : [Aouag, 2014]

- Le modèle de résolution est difficile à lire et à comprendre en plus il peut provoquer des erreurs.
- L'objet ne peut pas faire la suppression.

- La dispersion (duplication), Les modèles transversales se transverses dans le modèle d'application.
- Il n'existe pas un enchainement et un éparpillant du modèle.
- La réutilisation des modèles est complexe.

2.3 La modélisation orientée aspect (MOA)

2.3.1 L'approche orientée aspect

2.3.1.1 Ingénierie des exigences orientée aspect

ces approches fournissent une représentation des préoccupations transverses dans les artefacts d'exigences. Elles reconnaissent explicitement l'importance d'identifier et de traiter les préoccupations transversales d'une manière précoces, les préoccupations transverses peuvent être des exigences non fonctionnelle aussi bien que des exigences fonctionnelles, leurs identifications précoces permettent l'analyse précoce de leurs interactions. Ces approches se concentrent sur le principe de la composition de toutes les préoccupations pour avoir le système complet en cours de construction. Ainsi, il est possible de comprendre les interactions et les compromis entre les préoccupations. La composition des besoins permet non seulement d'examiner les exigences dans leur ensemble, mais aussi la détection des conflits potentiels très tôt, dans l'ordre de prendre des mesures correctives ou décisions appropriées à la prochaine étape. [Boubendir, 2011]

2.3.1.2 Les approches d'architecture orientée aspect

un aspect architectural est un module architectural qui a une grande influence sur d'autres modules architecturaux .Les approches de conception d'architecture orientées aspect donc décrit les étapes d'identification des aspects architecturaux et leurs composants enchevêtrés, cette information est utilisée pour refaire une conception d'architecture donnée tout en mettant les aspects architecturaux explicites. Ceci est différent des approches traditionnelles où les aspects architecturaux sont une information implicite dans la spécification de l'architecture .l'étude menée dans donne une vue assez

exhaustive de ces approches. [Boubendir, 2011]

2.3.1.3 Les approches de conception orientées aspect

ces approches de conception se concentrent sur la représentation explicite des préoccupations transversales en utilisant des langages de conception adéquates. Dans les premiers temps, les concepteurs utilisaient simplement des méthodes et des langages orientées objet (tel que UML) pour la conception de leurs aspects. Cela s'est révélé difficile puisque l'UML n'a pas été conçu pour fournir des constructeurs pour décrire les aspects, la principale contribution de la conception orientée aspect a été donc de fournir aux concepteurs des moyens explicites pour modéliser les systèmes par aspect. [Boubendir, 2011]

2.3.1.4 Les approches de vérification et de test des programmes orientées aspect

L'approche orientée aspect lève de nouveaux défis dans les techniques de vérification et validation des logicielles afin de s'assurer que la fonctionnalité désirée est satisfait par le système. des aspects peuvent potentiellement endommager la fiabilité d'un système auquel ils sont tissés, et peuvent rendre invalide des propriétés essentielles du système qui étaient corrects avant le tissage d'aspect .Pour assurer l'exactitude du logiciel par aspects, il y a beaucoup de recherche sur l'utilisation de méthodes formelles et techniques de tests spécialement adaptés aux aspects. [Boubendir, 2011]
Mais dans notre travail nous sommes intéressés par l'approche de conception orientée aspect, puisque les diagrammes de classes orientés aspect que nous avons utilisé pour réaliser notre implémentation est basé sur cette approche.

2.3.2 Pourquoi l'approche orientée aspect ?

L'approche aspect a été utilisée pendant les premières années dans les langages de programmation dans la phase de codage, le paradigme orienté aspect ne s'est plus restreint au niveau de la programmation, et il s'étend maintenant aux phases amonts du développement logiciel. Des approches orientées aspects sont aujourd'hui disponibles à chaque phase du développement d'un logiciel : analyse des exigences, concep-

tion, ou encore implémentation. Passer d'une phase à l'autre en conservant les aspects identifiés au préalable reste un défi majeur, pourtant peu étudié. Une approche itérative et dirigée par les préoccupations permettant de transformer un modèle d'exigences orienté aspect en un modèle de conception lui aussi orienté aspect, et ceci de manière automatique. [Menasra et Cherafa, 2013]

2.3.3 Définition de la modélisation orientée aspect (MOA)

La modélisation orientée aspect permet la séparation entre un modèle métier (fonctionnel) et un modèle technique (non fonctionnel), et utilise un mécanisme d'intégration (Weaver) pour obtenir le modèle intégré (métier et aspect). Pour cela il existe plusieurs travaux ont été proposés pour définir l'aspect (comme entité) dans les phases d'analyse et de conception [Aouag, 2014]. La principale difficulté des techniques de modélisation orientée aspect est le tissage des modèles d'aspect qui permet de construire le modèle complet. Le processus de tissage il contient le modèle de base correspond au modèle dans lequel l'aspect doit être tissé. Un aspect est composé de deux parties : un point de coupure et un advice. Le point de coupure décrit les structures ou les comportements du modèle de base qui doivent être modifiés et l'advice spécifie la structure ou le comportement qui doit être composé dans le modèle de base. Lors de la phase de détection, le point de coupure sert donc à trouver l'ensemble des points du modèle de base auxquels l'advice doit être composé lors de la seconde phase. [Abdellah et Tiga, 2014] Voici un exemple qui explique le processus de la modélisation orientée aspect tels que : S1, S2 et S3 sont des états, vérification et sécurité sont des aspects.

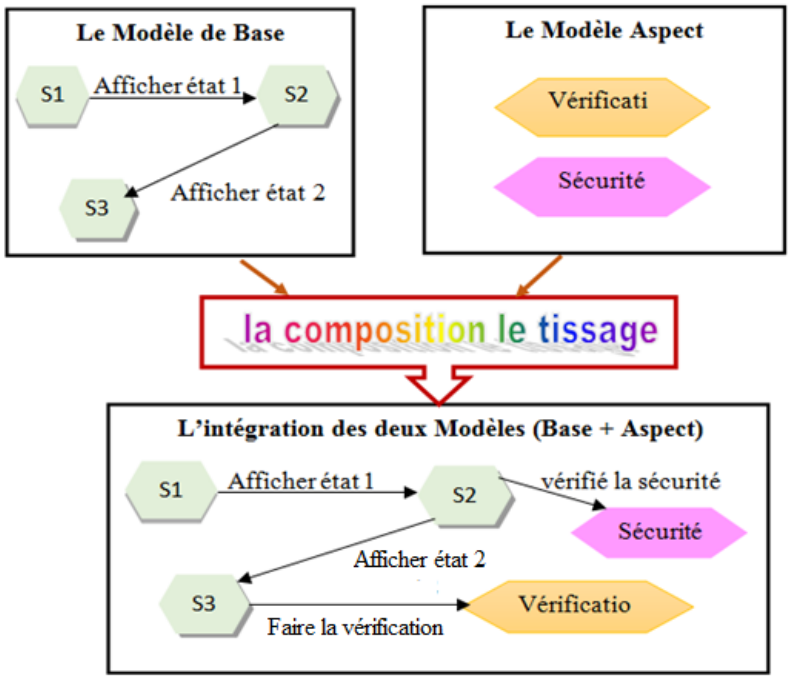

FIGURE 2.17 – Le processus de la modélisation orientée aspect.

La modélisation orientée aspect est basée sur les points suivantes :

•**Point de jointure :** Un point de jonction est un endroit dans le modèle d'aspect où seront utilisés un ou plusieurs aspects. Ce point de jonction est déterminé lors de la composition par le développeur. L'utilisation d'un aspect pour un point de jonction se fait de manière dynamique, c'est-à-dire une fois le modèle en exécution (weaver). Il existe plusieurs types de points de jonction, selon ce que le développeur souhaite intercepter. Ainsi, un point de jonction peut intervenir pour :

- Une classe
- Une interface
- L'exécution d'une méthode
- L'accès à une variable de classe. [Otman Rachedi, 2015]

•**Point de coupure :** une coupe désigne un ensemble de point de jonctions. Une coupe est définie à l'intérieur d'un aspect. Dans le cas simple, une seule coupe suffit pour définir la structure transversale d'un aspect, dans les cas plus complexes, un aspect est as-

socié à plusieurs coupes. Les notions de coupe et de point de jonction sont liées par leur définition. Pourtant, leur nature est très différente. Une coupe est un choix de points de jonction défini dans un aspect, alors qu'un point de jonction est un emplacement dans l'exécution d'un modèle. De ce fait un point de jonction peut appartenir à plusieurs coupes dans un même aspect ou dans des aspects différents. [Amroune, 2014]

•**Advice :**Une consigne (advice en anglais) représente un comportement technique particulier d'un aspect. Concrètement, c'est un modèle qui sera greffé dans l'application à l'exécution d'un point de jonction définit dans une coupe. Un aspect peut contenir une à plusieurs consignes. De même qu'un aspect peut être assimilé à une classe en modélisation orientée objet, un modèle de consigne pourra être comparé à une méthode de l'aspect. Comme nous l'avons vu précédemment, une coupe peut contenir plusieurs points de jonction, qui eux-mêmes peuvent référencer plusieurs aspects. Une consigne est donc susceptible d'être utilisée à plusieurs endroits dans l'application et un point de jonction peut éventuellement greffer plusieurs consignes au moment de l'interception. [Otman Rachedi, 2015]

•**Tisseur (Weaver en anglais) :**Le tissage (weaving en anglais) est une opération automatique qui est absolument nécessaire pour obtenir une application opérationnelle, cette étape sert à intégrer les fonctionnalités des classes et celles des aspects. Le programme qui réalise cette opération est appelé un tisseur d'aspects. L'application obtenue à l'issue du tissage est dite tissée. [Otman Rachedi, 2015]

2.3.4 Travaux de modélisation orientée aspect

Il existe un grand nombre de travaux de modélisation orientées-aspect, Nous allons présenter six de ces approches qui nous apparaissent les plus représentatives :

2.3.4.1 Approche de Modélisation Orientée-Aspect de France et al

L'approche de modélisation orientée-aspect de France et al est une des approches existantes les plus avancées. Ils proposent des mécanismes de composition évalués qui sont réellement implantés et donc utilisable. Nous pouvons tout de même faire deux

critiques majeures à leur approche .Premièrement, ils ne proposent pas de mécanisme de détection de point de jonction. Deuxièmement, les mécanismes de composition sont restreints aux modèles statiques. [Klein, 2006]

2.3.4.2 Approche de Modélisation Orientée-Aspect de Clarke et al

Thème est une approche dont l'intérêt majeur porte sur la méthodologie qu'elle propose. Thème utilisent des thèmes qui comportent une partie structurelle et un partie comportementale. Les inconvénients de Thème sont : premièrement qu'il n'existe pas le non existence d'outils et de formalisations des mécanismes de composition. Deuxièmement thème ne propose pas de mécanisme de détection de points jonction. [Klein, 2006]

2.3.4.3 Approche de Modélisation Orientée-Aspect de Muller et al

L'approche de Muller et al n'est pas directement liée à une approche de modélisation orienté-aspect, mais comme l'approche de France et al, elle propose des mécanismes de composition essentiellement sur des modèles structuraux. L'avantage majeur de cette approche est la définition formelle d'opérateurs de compositions, et la proposition de propriétés permettant de garantir la cohérence des compositions, l'approche de Muller et al ne proposent pas de mécanisme de détection de points de jonction. [Klein, 2006]

2.3.4.4 Approche de Modélisation Orientée-Aspect de Stein et al

L'objectif de l'approche de Stein et al .et de modéliser des programme Aspect, elle ne propose pas de mécanismes de composition et de détection de points de jonction. [Klein, 2006]

2.3.4.5 Approche de Modélisation Orientée-Aspect d'Aldawud et al

L'approche d'Aldawud et al est principalement utilisée pour présenter des aspects à un niveau de modélisation. Mais une fois encore, cette approche ne propose pas de réels mécanismes de tissages d'aspects à un niveau de modélisation. [Klein, 2006]

2.3.4.6 Approche de Modélisation Orientée-Aspect d'Aouag.M

L'objectif de l'approche de Aouag.M.et de modéliser des modèles d'aspect. Elle est principalement utilisée pour présenter des aspects à un niveau de modélisation. L'avantage majeur de cette approche est la définition formelle d'opérateurs de compositions, et la proposition de propriétés permettant de garantir la cohérence des compositions. En plus l'approche d'Aouag.M assure des mécanismes de détection de points de jonction. [Aouag, 2014]

2.4 UML2.0 et la MOA

Il existe plusieurs diagrammes UML2.0 qui sont orientés aspect on peut citer : les diagrammes de classes, de communication, et d'activité. Dans notre travail, on s'intéresse au diagramme de classes orienté aspect pour cela il est nécessaire de donner qu'elle que définition de base.

2.5 Définition de diagramme de classes orienté aspect

Un diagramme de classes orienté aspect est un simple diagramme de classes orienté objet comme présenter dans la section 2.2.4. Mais, on trouve des aspects (non fonctionnel) qui sont ajoutés au niveau des points de jointure sélectionné par les points de coupure dans le modèle de base pour fournir un modèle orienté aspect(modèle base plus modèle d'aspect). [Aouag et al, 2013]

Exemple : Pour illustrer notre approche, nous l'avons appliquée au comportement d'un système de recherche simple dans le réseau internet. Par conséquent, nous avons utilisé un diagramme de classes, où les classes représentent les différentes entités du système et les liens représentent la relation entre ces classes.

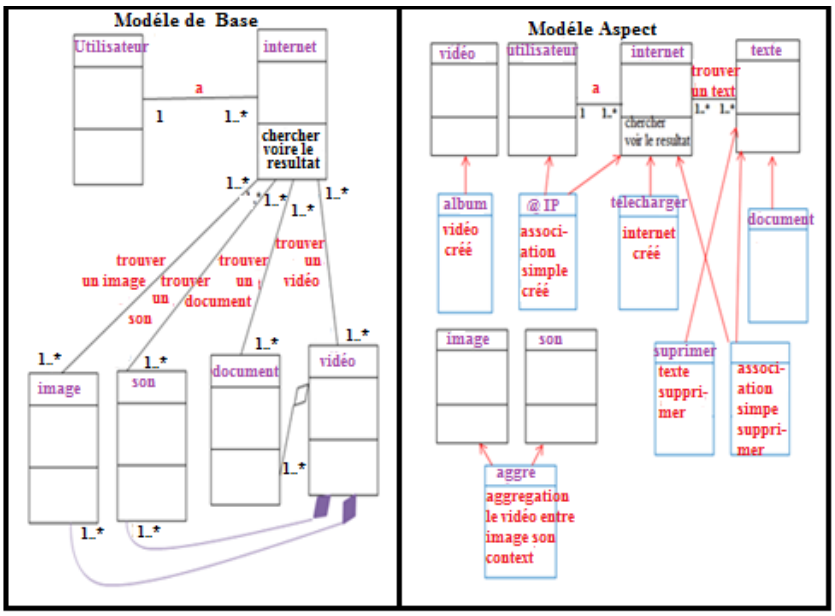

FIGURE 2.18 – modèle de base et modèle d'aspect pour l'exemple de recherche dans le réseau internet.

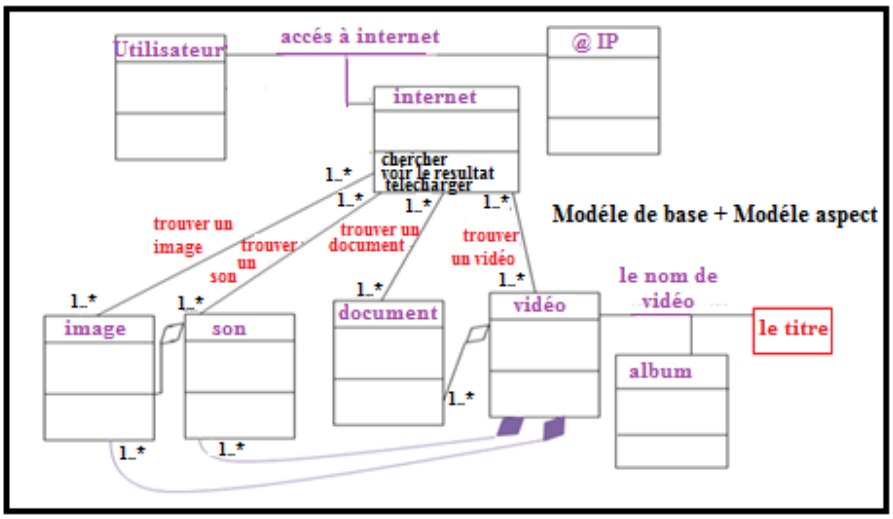

FIGURE 2.19 – Diagramme de classes orienté aspect.

Conclusion

Dans ce chapitre nous avons présenté brièvement le langage de modélisation unifié UML 2.0 et ses diagrammes, Ainsi les avantages et les limites de la modélisation orientée objet. Et nous avons vu la définition de la modélisation orientée aspect et les éléments de ce modèle. Ensuite nous parlons sur notre modèle source « diagramme de classes orienté aspect » et nous avons vu les compartiments de base, les principes et les formalismes de la représentation graphique d'un diagramme de classes. Nous avons insisté sur les associations entre la classe, qui seront modélisés dans le formalisme source de la transformation constituant l'objet de notre travail.

Alors les diagrammes de classes orientés aspect ont des mécanismes pour exprimer le séquencement, le choix et le parallélisme. Mais ils souffrent du manque des outils de vérification de comportement, qui sont disponible dans les réseaux de pétri « notre modèle cible », qui est défini dans le chapitre suivant.

CHAPITRE 3

Les Réseaux de Pétri(RdP)

Sommaire

Introduction

Les systèmes dynamiques ne peuvent pas être décrits en ne prenant en compte que leurs états initiaux et finaux. En effet, on doit tenir compte de leur comportement permanent qui est une séquence d'états, pour qu'on puisse parler d'une description bien fondée. Parmi le grand nombre des techniques formelles qui ont déjà été proposées pour spécifier, analyser et vérifier ce genre de systèmes, les réseaux de pétri sont l'une des plus utilisés.

Les réseaux de pétri offrent un outil formel et une bonne représentation graphique qui permettent de modéliser et d'analyser les systèmes discrets, particulièrement les systèmes concurrents et parallèles. La facette graphique des réseaux de pétri, nous aide à comprendre aisément le système modélisé.

Par ailleurs, leur puissance d'expression mathématique permet de simuler des activités dynamiques et concurrentes. L'intérêt majeur de ces réseaux réside dans leurs possibilité d'analyser les systèmes modélisés, grâce aux modèles de graphes et aux équations algébriques. Les réseaux de pétri sont des outils de modélisation utilisés généralement en phase préliminaire de conception de système afin de réaliser leur spécification fonctionnelle, modélisation et suivre leur évaluation. Grâce à leur expressivité et à leur souplesse, ils sont utilisés dans une large variété de domaines. Ils permettent notamment : La modélisation des systèmes informatiques, l'évaluation des performances des systèmes discrets, des interfaces homme-machine, la commande des ateliers de fabrication, la conception de systèmes temps réel, La modélisation des protocoles de communication, la modélisation des chaines de production (de fabrication). Dans le présent chapitre, nous nous intéressons au formalisme RdP. Nous présentons dans un premier lieu les concepts de base de réseau de pétri. Ensuite, nous rappelons certaines de leurs extensions, les propriétés de RdP, les méthodes d'analyse par RdP .

3.1 Historique

Historiquement le réseau est présenté par **Carl Adam Pétri** dans sa thèse "Communication avec des Automates" en **Allemagne à Bonn en 1962**. Ce travail a continué à être développé par Anatol W. Holt, F. Commoner, M. Hack et leurs collègues dans le groupe de recherche de Massachussetts Institute Of Technology (MIT) dans des années 70s. En 1975 la première conférence de réseau de pétri et des méthodes relationnels ont été organisés à MIT. En 1981 le premier livre du réseau de pétri a été publié en Anglais par J. Peterson. Aujourd'hui, suivre pétri-Net Newsletter, chaque année il y a des 600 aux 800 d'œuvres des réseaux de pétri sont publiés. [Hettab, 2009]
Ce formalisme a été proposé comme un outil mathématique permettant la modélisation des systèmes dynamiques à événements discrets. Les réseaux de pétri offrent un outil formel avec une bonne représentation graphique permettant de modéliser et d'analyser les systèmes discrets, notamment les systèmes concurrents et parallèles. L'intérêt majeur de ces réseaux réside dans leur possibilité d'analyser les systèmes modélisés. En effet, Ce formalisme bénéficie d'une multitude de techniques d'analyse et d'outils. [Benmoussa et Houidi, 2015]

La figure suivante représente la méthode générale de modélisation des RdP.

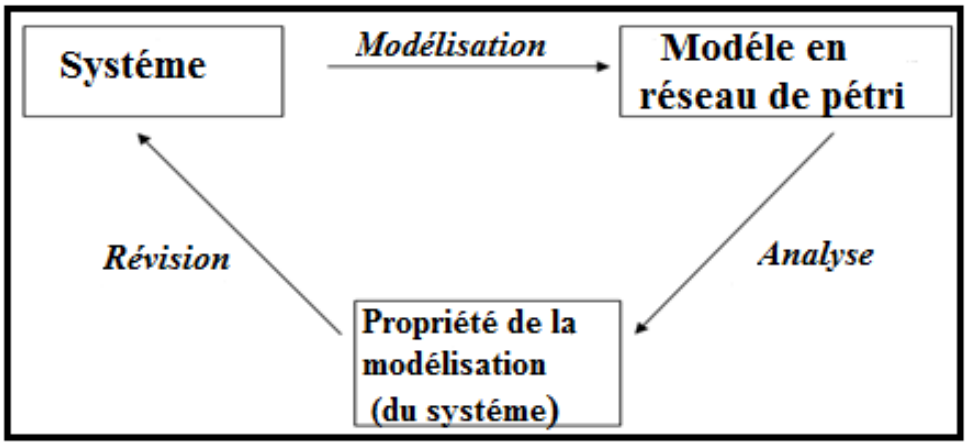

FIGURE 3.1 – Méthode générale de modélisation et d'analyse basée sur les réseaux de pétri.

3.2 Le réseau de pétri

3.2.1 Définitions informelles de RdP

Intuitivement, un réseau de pétri est un graphe orienté biparti (ayant deux types de nœuds) : des places représentées par des cercles et des transitions représentées par des rectangles. Les arcs du graphe ne peuvent relier que des places vers des transitions, ou des transitions vers des places (pas d'arcs entre places ni entre transitions) .
D'autre part un réseau de pétri décrit un système dynamique à événements discrets. Les places permettent la description des états possibles du système (qui sont discrets), les transitions permettent la description des événements ou les actions qui causent le changement de l'état. Ce réseau est un graphe muni d'une sémantique opérationnelle, c'est-à-dire qu'un comportement est associé au graphe, ce qui permet de décrire la dynamique du système représenté. Pour cela un troisième élément est ajouté aux places et aux transitions qui est les jetons. Une répartition des jetons dans les places à un instant donné est appelée marquage du réseau de pétri. Un marquage donne l'état du système. Le nombre de jetons contenus dans une place est un entier positif ou nul, il ne peut pas être négatif. Un marquage donné, permet à une transition d'être sensibilisée ou non. Une transition est sensibilisée si chacune de ses places d'entrée contient au moins un jeton. L'ensemble des transitions sensibilisées pour un marquage donné défini l'ensemble des changements d'états possibles du système depuis l'état correspondant à ce marquage. C'est un moyen de définir l'ensemble des événements auxquels ce système est réceptif dans cet état. [Benmoussa et Houidi, 2015]

Un exemple de réseau de pétri est illustré par la figure 3.2.

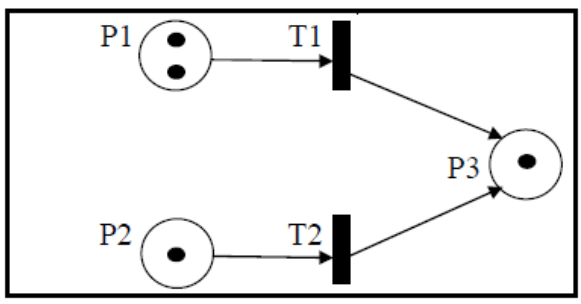

FIGURE 3.2 – Réseau de pétri marqué.

• **Concepts de base pour les RdP :**

Il existe trios concepts de base : [Benmoussa et Houidi, 2015]

- Condition : Une condition est un prédicat ou une description logique d'un état du système.Une condition peut être vraie ou fausse. Un état du système peut être décrit comme un ensemble des conditions.

- Evénement : Les événements sont des actions se déroulant dans le système. Le déclenchement d'un événement dépend de l'état du système.

- Déclenchement, pré-condition, post-condition : Les conditions nécessaires au déclenchement d'un événement sont les pré-conditions de l'événement. Lorsqu'un événement se produit,certaines de ses pré-conditions peuvent cesser d'être vraies alors que d'autres conditions, appelées post-conditions de l'événement, deviennent vraies.

3.2.2 Définitions formelles de RdP

Formellement, un réseau de pétri (R) est un triplet R= (P, T, W) où P est l'ensemble des places (les places représentent les conditions) et T l'ensemble des transitions (les transitions représentent les événements ou les actions) tel que P $\cap$ T = ϕet W est la fonction définissant le poids porté par les arcs où W : ((P $\times$ T) $\cup$(T $\times$ P))$\longrightarrow$ N=$\{$0, 1, 2,. . . $\}$.

- Le réseau R est fini si l'ensemble des places et des transitions est fini, c.-à-d.$|$ P $\cup$ T $| \in$ N.
- Un réseau R = (P, T, W) est ordinaire si pour tout (x, y) $\in$ ((P $\times$ T) $\cup$ (T $\times$ P)) : W(x, y) $\leq$ 1.

Dans un réseau ordinaire la fonction W est remplacée par F où :

F $\subseteq$((P$\times$T)$\cup$(T $\times$ P)) tel que (x, y)$\in$ F $\Longleftrightarrow$ w (x, y) $\neq$ 0 . Pour chaque x$\in$P $\cup$ T :

- *x représente l'ensemble des entrées de x :*x= $\{$ y $\in$P $\cup$ T $|$ W (y, x) $\neq$ 0$\}$
- x* représente l'ensemble des sorties de x :x*= $\{$ y $\in$P $\cup$ T $|$ W (y, x) $\neq$ 0 $\}$

Remarque :

- si *x$\neq$ ϕ, x est dite source, si x* $\neq$ ϕ, x est dite puits.

Le comportement d'un réseau de pétri est déterminé par sa structure et par son état. Pour exprimer l'état d'un réseau de pétri, les places peuvent contenir des jetons qui ne

sont que de simples marques. [Elmansouri, 2009]

3.2.3 Représentation graphique de RdP

Le réseau de pétri est représenté par deux type de sommets alternés, les places *Pi* et les transitions *Ti*. Ces sommets sont reliés par des arcs orientés. Tout arc doit relier une place à une transition ou bien une transition à une place. [Bahri, 2010]

La figure 3.3 montre la représentation graphique d'un RdP comportant 7 places, 8 transitions et 17 arcs orientés.

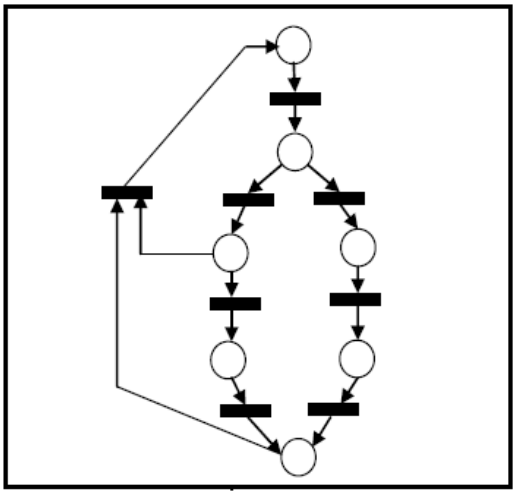

FIGURE 3.3 – Représentation graphique de RdP.

3.2.4 Le marquage d'un réseau de pétri

Le marquage d'un réseau de pétri permet de définir l'état d'un système modélisé par ce réseau. Le marquage consiste à placer initialement un nombre*mi* entier (positif ou nul) de jetons dans chaque place *Pi* du réseau. Le marquage du réseau sera défini par un vecteur **M = mi**. [Bahri, 2010]

Un marquage donné, a une transition peut être sensibilisée ou non. Si chacune des places en entrée d'une transition contient au moins un jeton, elle est ***sensibilisée***. Pour un marquage donné, L'ensemble des transitions sensibilisées défini l'ensemble des changements d'états possibles du système depuis l'état correspondant à ce marquage.

C'est un moyen de définir l'ensemble des événements auxquels ce système est ***réceptif*** dans cet état. [Dehimi, 2014]

La figure 3.4 représente un réseau de pétri marqué avec un vecteur de marquage M tel que : M = (1, 0, 1, 0, 0, 2,0).

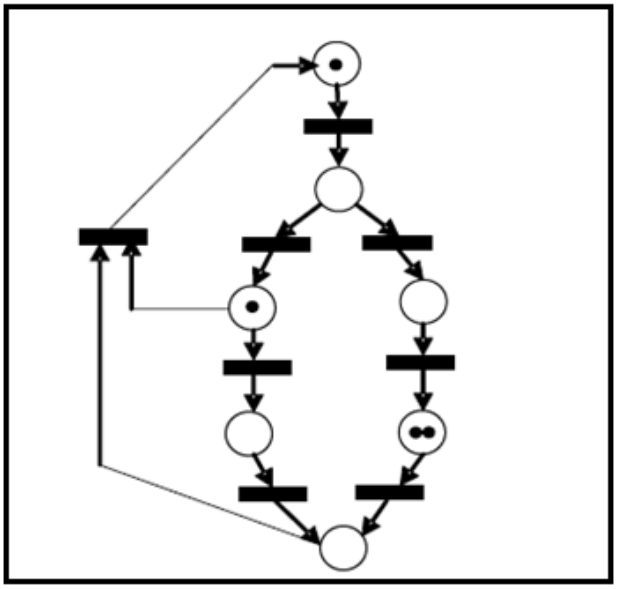

FIGURE 3.4 – Le marquage d'un réseau de pétri.

3.2.5 Evolution d'états d'un réseau de pétri

L'évolution d'état du réseau de pétri correspond à une évolution de marquage. Les jetons qui indiquent l'état du réseau à un instant donné, peuvent passer d'une place à une autre par un **franchissement** ou par un **tir** d'une transition. Dans le cas des réseaux dits à arcs simples ou de poids égal à 1, le franchissement d'une transition consiste à retirer un jeton dans chacune des places d'entrée de la transition et à en ajouter un dans chacune de sorties de places de celle-ci. (La figure 3.5). [Bahri, 2010]

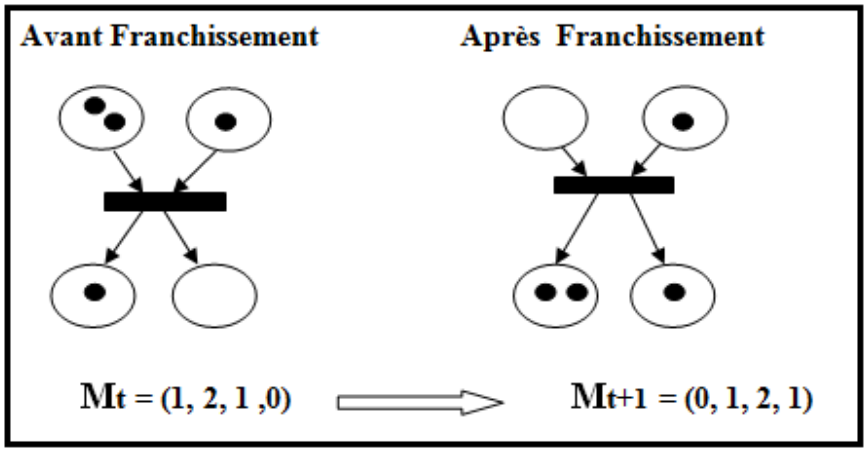

FIGURE 3.5 – Evolution d'états d'un réseau de pétri.

En général l'évolution des états d'un réseau de pétri marqués simple obéit aux règles suivantes : [Bahri, 2010]

• Une transition est franchissable ou sensibilisée ou encore tirable lorsque chacune des places d'entrées possède au moins le nombre de jetons correspondant au poids de l'arc le reliant à la transition.

• Le réseau ne peut évoluer que par franchissement d'une seule transition à la fois, transition sélectionnée parmi toutes celles validées au moment du choix.

• Le franchissement d'une transition est indivisible et de durée nulle. Ces règles introduisent un certain indéterminisme dans l'évolution des réseaux de Pétri, puisque ceux-ci peuvent passer par différents états dont l'apparition est conditionnée par le choix des transitions tirées. Ce fonctionnement représente assez bien les situations réelles où il n'y a pas de priorité dans la succession des événements.

3.2.6 Représentation matricielle

Une représentation matricielle d'un RdP est offerte afin de simplifier les tâches d'analyse et de vérification effectuée sur un modèle RdP. Agir sur une représentation graphique d'un modèle RdP est une tâche délicate en la comparant avec une représentation matricielle. Il est possible de représenter la fonction W (fonction de poids) par des matrices. [Elmansouri, 2009]

D'autre part un réseau de pétri R= (P, T, W) avec P={p1, p2, . . . , pm} et T= {t1, t2, . . . ,tn}, on appelle matrice des pré-conditions *pré*, la matrice m × n à coefficients dans N telle que pré(i,j)= W(pi, tj) indique le nombre de marques que doit contenir la place pi pour que la transition tj devienne franchissable. De la même manière on définit la matrice des post-conditions*post*, la matrice n × m telle que post (i,j)= W(tj, pi) contient le nombre de marques déposées dans pi lors du franchissement de la transition tj. La matrice C= post – pré est appelée matrice d'incidence du réseau (m représente le nombre de places d'un réseau de pétri et n le nombre de transitions). Le marquage d'un réseau de pétri est représenté par un vecteur de dimension m à coefficients dans N. La règle de franchissement d'un réseau de pétri est définie par : M' (p) =M (p) + C (p, t) .

La figure 3.6 représente un réseau de pétri marqué avec un vecteur de marquage M tel que : M = (1, 0, 0,0).

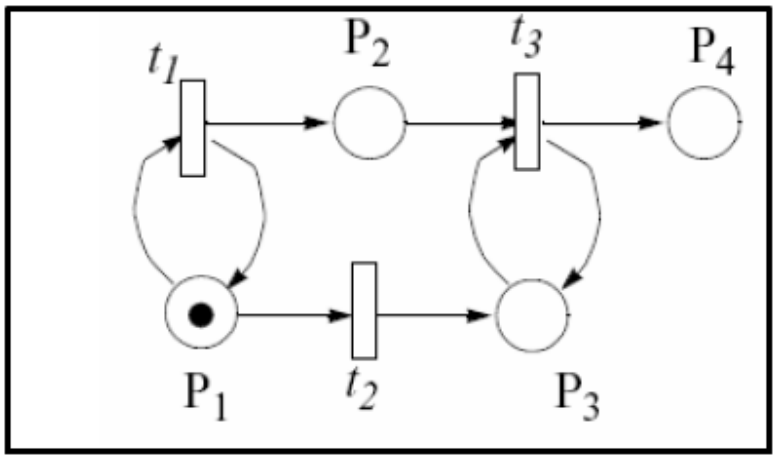

FIGURE 3.6 – Un réseau de pétri marqué.

Pour le réseau ci-dessus (la figure 3.6), P= p1, p2, p3, p4 T= t1, t2, t3, la représentation matricielle est donnée ci-dessous (la figure 3.7). du RdP de la figure 3.6.

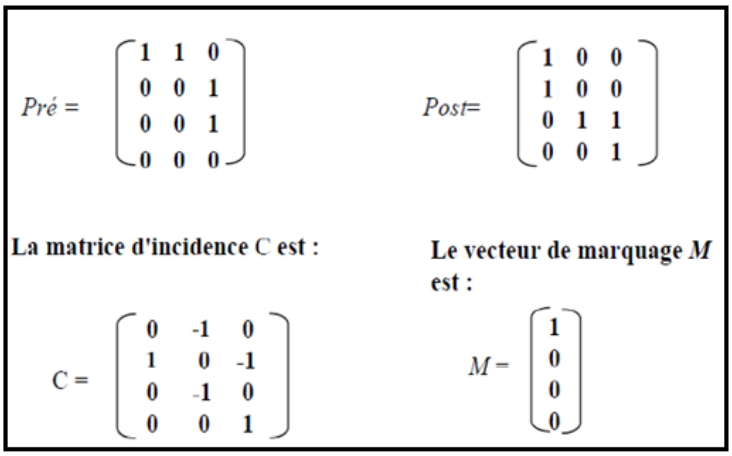

FIGURE 3.7 – Matrice d'incidence .

3.3 Méthodes d'analyse pour les réseaux de pétri

La modélisation d'un système doit permettre l'analyse de ses propriétés. Les réseaux de pétri offrent des techniques d'analyse puissantes pour valider des modèles de comportement de systèmes à événements discrets. Parmi ces techniques, nous citons le graphe de marquages, l'équation de matrice et la réduction des réseaux de pétri [Bahri, 2010].

• **Le graphe de marquage :** Il s'agit de construire le graphe de tous les marquages du réseau.Les propriétés sont par la suite déduites grâce aux techniques de la théorie des graphes.

• **L'équation de matrice :** Cette méthode consiste à trouver une représentation matricielle du réseau, les techniques d'algèbre linéaire permettent alors d'obtenir les propriétés du réseau.

• **La réduction des RdPs :** Pour l'analyse des propriétés d'un RdP de taille significative, l'utilisation du graphe de marquage ou de l'équation de matrice s'avère insuffisante. L'objectif de la technique par réduction est de présenter des règles permettant d'obtenir à partir d'un RdP marqué, un RdP marqué plus simple, avec un nombre réduit de places et de transitions.

Les méthodes d'analyse des réseaux de pétri prennent pleine puissance avec leurs mise en œuvre par le biais d'un ensemble d'outils d'analyse tels que : INA (Integrated Net Analyzer), PEP (Programming Environment based on Pétri nets), TINA (TIme Pétri Net Analyzer) etc.

3.4 Réseaux de pétri particuliers

3.4.1 Graphe d'état

Un réseau de pétri non marqué est un graphe d'état si et seulement si toute transition a exactement une seule place d'entrée et une place de sortie .Exemple : les transitions T1, T2, T3, T4 et T4 possède une place d'entrée et une seule place de sortie. Comme il est détaillé dans cette figure 3.8 : [Laouar, 2013]

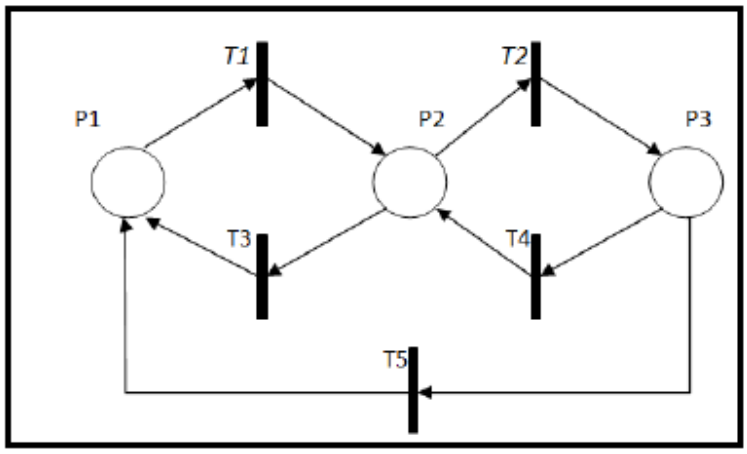

FIGURE 3.8 – Graphe d'état.

3.4.2 Graphe d'événement

Un réseau de pétri est un graphe d'événement si et seulement si chaque place possède exactement une seule transition d'entrée et une seule transition de sortie comme il est schématisé ci-dessous (Figure 3.9).un graphe d'évènements est parfois appelé graphe de transition. [Saggadi, 2007]

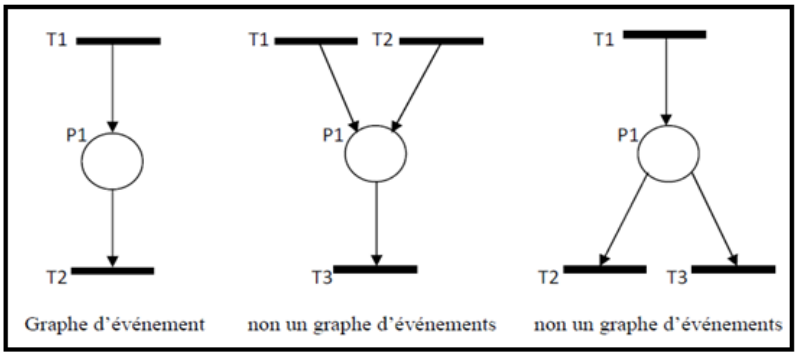

FIGURE 3.9 – Graph d'événement .

3.4.3 RdP sans conflit

dire qu'un RdP est un réseau sans conflit si et seulement si chaque place a au plus une transition de sortie. Un RdP avec conflit est un réseau qui possède donc une place avec au moins deux transitions de sorties. Un conflit est noté : [Pi, T1, T2,..... Tn] ; avec T1, T2,.... Tn étant les transitions de sortie de la place Pi. Comme il est montré dans la figure 3.10 [Laouar, 2013]

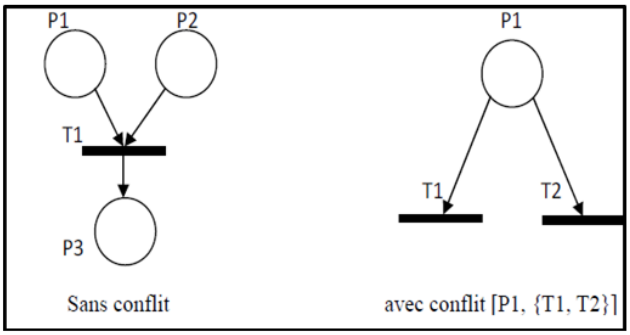

FIGURE 3.10 – Graphe RdP sans conflit .

3.4.4 RdP a choix libre

un RdP est à choix libre est un réseau dans lequel pour tout conflit [Pi, T1, T2,....Tn] aucun des transitions T1, T2,..., Tn ne possède aucune autre place d'entrée que Pi. Comme il est exprimé dans la figure 3.11. [Saggadi, 2007]

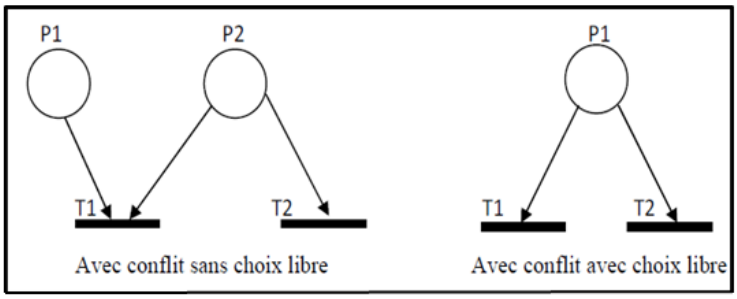

FIGURE 3.11 – RdP avec choix libre.

3.4.5 RdP simple

les réseaux de pétri simples sont des RdP ordinaires tels que chaque transition a au plus une place d'entre qui peut être reliée a d'autre transition (Toute transition appartient à un seul conflit au plus). Tel qu'il est présenté dans la figure 3.12. [Saggadi, 2007]

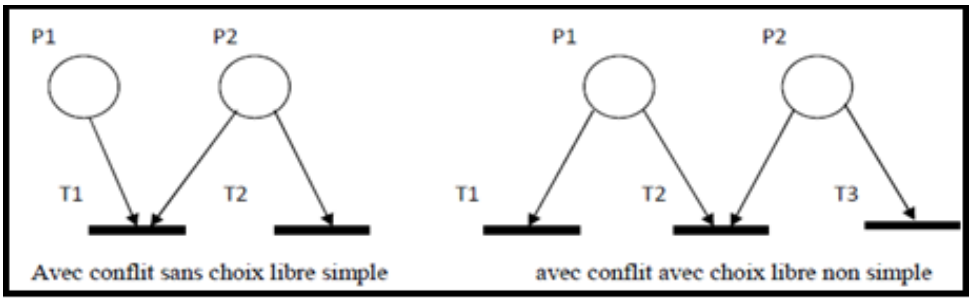

FIGURE 3.12 – RdP simple .

3.4.6 RdP pur

dans un RdP une transition est dite pure si elle ne possède aucune place qui est à la fois place d'entrée et place de sortie. Si toutes les transitions du RdP est pur. Dans un RdP une transition est dite impure si elle possède une place qui est à la fois place d'entrée et place de sortie. Si les transitions du RdP sont impures le RdP est impur. [Saggadi, 2007]

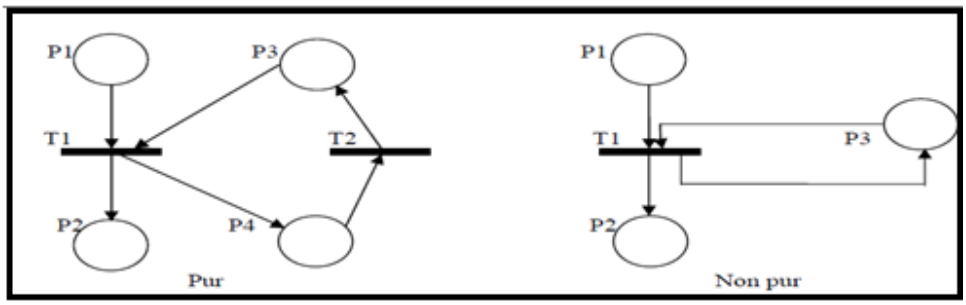

FIGURE 3.13 – RdP pur .

3.4.7 RdP généralisés

Un RdP généralisé est un RdP dans lequel des poids (nombres entiers strictement positifs) sont associés aux arcs. Si un arc (Pi,Tj) a un poids K : la transition Tj n'est franchie que si la place Pi possède au moins K jetons. Le franchissement consiste à retirer K jetons de la place Pi. Si un arc (Tj, Pi) a un poids K : le franchissement de la transition rajoute K jetons à la place Pi. Lorsque le poids n'est pas signalé, il est égal à un par défaut. Comme il est exprimé dans la figure suivante 3.14 . [Laouar, 2013]

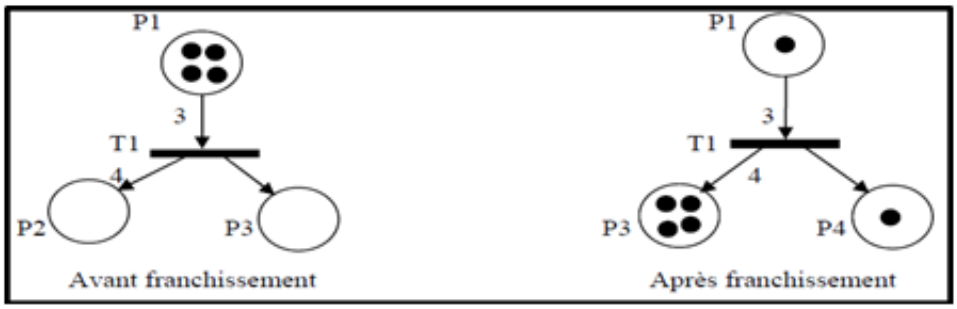

FIGURE 3.14 – RdP généralisé .

3.4.8 RdP a capacité

Un RdP à capacités est un RdP dans lequel des capacités (nombre entiers strictement positifs) sont associées aux places. Le franchissement d'une transition d'entrée d'une place Pi dont la capacité est cap(Pi) n'est possible que si le franchissement ne conduit pas à un nombre de jetons dans Pi qui est plus grand que cap(Pi).

La Figure 3.15 montre le franchissement de Ti conduit à 3 jetons dans P2 d'où T1 ne peut plus être franchie. [Laouar, 2013]

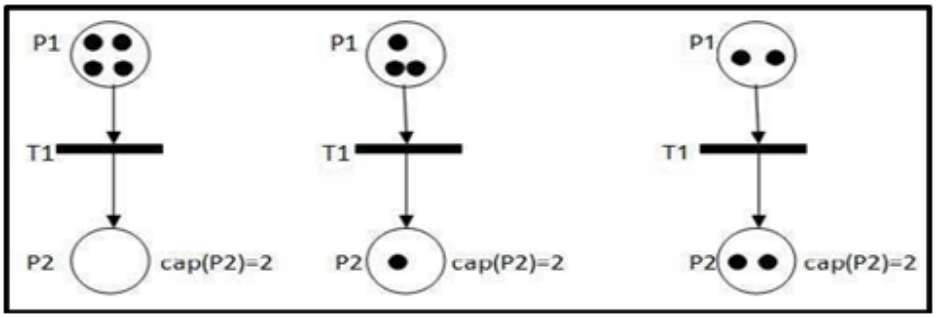

FIGURE 3.15 – RdP à capacité .

3.4.9 RdP à priorités

Dans un tel réseau si on atteint un marquage tel que plusieurs transitions sont franchissable, on doit franchir la transition qui a la plus grande priorité. Dans l'exemple suivant on a présenté un RdP à priorité comme le schématise dans la figure suivante 3.16. [Laouar, 2013]

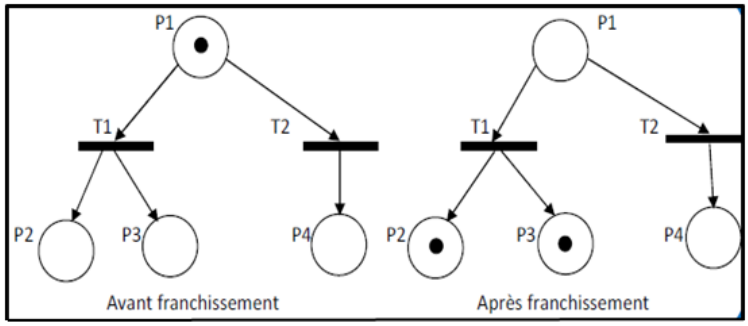

FIGURE 3.16 – RdP à priorité .

3.5 Extensions des réseaux de pétri

Les systèmes réel impose beaucoup de contraintes, Leur modélisation peut donc engendrer des réseaux de pétri de taille importante dont la manipulation et l'analyse est

très difficile. En plus, les RdPs usuels ne permettent pas d'exprimer certaines propriétés, telle que la mobilité, rendant ainsi leur analyse assez difficile. Dans ce contexte, plusieurs extensions de réseaux de pétri ont vu le jour, entre autres on a : les réseaux de pétri colorés, les réseaux de pétri temporisés etc. Dans la section suivante, nous présentons un aperçu de quelques extensions des RdPs [Dehimi, 2014].

3.5.1 Les réseaux de pétri colorés

Lorsque le nombre d'entités du système à modéliser est important, la taille du réseau de pétri devient rapidement énorme ; et si les entités présentent des comportements similaires, l'usage des réseaux colorés permet de condenser le modèle. Les réseaux de pétri colorés sont des réseaux de pétri dans lesquels les jetons portent des couleurs. Une couleur est une information attachée à un jeton. Cette information permet de distinguer des jetons entre eux et peut être de type quelconque. Ainsi, les arcs ne sont pas seulement étiquetés par le nombre de jetons mais par leurs couleurs. Le franchissement d'une transition est alors conditionné par la présence dans les places en entrée du nombre de jetons nécessaires, qui en plus satisfont les couleurs qui étiquettent les arcs. Après le franchissement d'une transition, les jetons qui étiquettent les arcs d'entrée sont retirés des places en entrée tandis que ceux qui étiquettent les arcs de sortie sont ajoutés aux places en sortie de cette transition. Les réseaux colorés n'apportent pas de puissance de description supplémentaire par rapport aux réseaux de pétri, ils permettent juste une condensation de l'information. A tout réseau de pétri coloré marqué correspond un réseau de pétri qui lui est isomorphe. La relation entre le RdP coloré et le RdP ordinaire vu comme une relation entre le langage de programmation de haut niveau et le code assembleur. Théoriquement les deux niveaux d'abstraction ont la même sémantique. De plus le langage de haut niveau offre une grande puissance de modélisation par apport au langage assembleur ; car il est bien structuré, bien typé et modulé [Hettab, 2009]. Le passage du RdP (Figure 3.17 gauche) au RdP coloré (Figure 3.17 droite) est appelé pliage et le passage du RdP coloré au RdP dépliage.

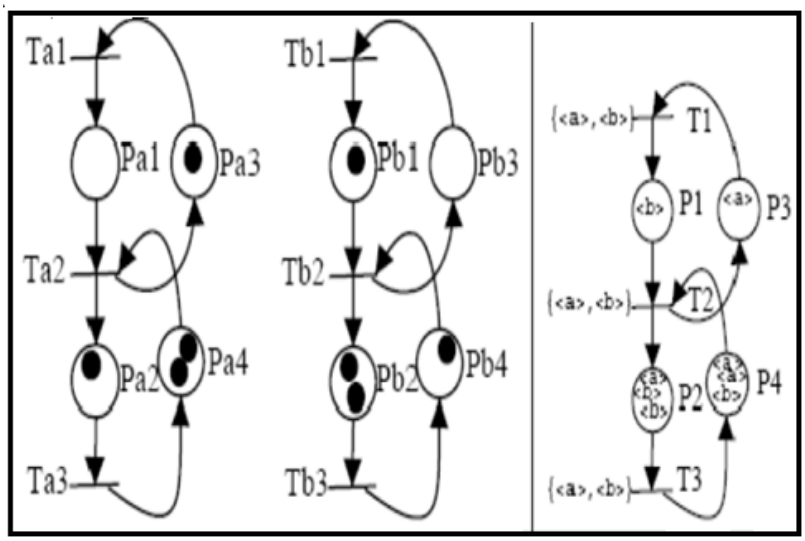

FIGURE 3.17 – RdP (gauche) et RdP coloré (droite) .

3.5.2 Les réseaux de pétri temporisés

Les RdP temporisés RdPT introduits étendent les RdP avec des intervalles temporels associés aux transitions, spécifiant les bornes de délai de tir des transitions. Ils permettent la modélisation des systèmes avec des contraintes temporelles tel que : les protocoles de communication, certains cas des systèmes à temps réels, etc. L'extension temporelle peut s'envisager sur les places, sur les arcs ou sur les transitions .Merlin définit le RdPT comme un RdP avec deux valeurs du temps a et b associées aux transitions ; avec ($0 \leq a \leq b$) et b peut être illimité, spécifiant les bornes de délai de franchissement des transitions. En considérant que la transition t est devenue sensibilisée pour la dernière fois à l'instant :θ .alors t ne peut être franchie plus tôt qu'à l'instant :θ + a ; et ne doit pas être franchie plus tard qu'avant (ou exactement à) : θ + b . L'intervalle [a, b] est le temps pendant lequel les jetons des places en entrée ne sont plus présents (ils sont réservés), mais pendant lequel les jetons produits ne sont pas encore visibles dans les places de sortie. [Haiouni, 2010]

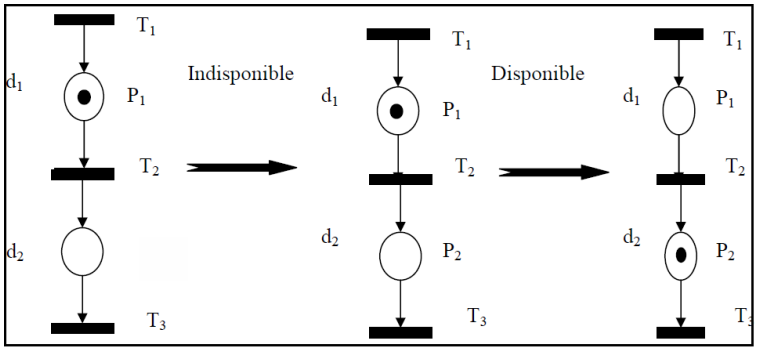

FIGURE 3.18 – RdP temporisés.

3.5.3 Les réseaux de pétri synchronisés

Dans les modélisations RdPS que nous avons vues précédemment, le fait qu'une transition soit franchissable indique que toutes les conditions sont réunies pour qu'elle soit effectivement franchie. Le moment où se produira le franchissement n'est pas connu. Un RdP synchronisé est un RdP où à chaque transition est associée un événement. La transition sera alors franchie si elle est validée et en plus, quand l'événement associé se produit. Dans un RdP synchronisé, une transition validée n'est pas forcement franchissable. La transition est validée quand la condition sur les marquages est satisfaite. Elle deviendra franchissable quand l'événement externe associé à la transition se produit : elle est alors immédiatement franchie. Si en fonction du marquage de ses places d'entrée, plusieurs franchissements sont possibles, un seul se produira effectivement, celui dont l'événement associé se produit en premier. [Dehimi, 2014]

3.5.4 Les réseaux de pétri stochastique :

Pour pouvoir utiliser la puissance de l'analyse *Markovienne*, il faut que les systèmes soient sans mémoire du passé, c'est-à-dire que si un événement produit un franchissement de transitions t et transforme le marquage M1 en M2, l'évolution future des transitions qui étaient sensibilisées par M1 avant le franchissement de t doit être identique à celle qu'elle subiraient si elles venaient juste d'être sensibilisées par M2. Seules les distributions géométriques et exponentielles vérifient ce fait. Les réseaux

de pétri stochastiques sont définis par de telles distributions afin de pouvoir construire un processus Markovien équivalent et ainsi analyser les comportements du réseau. Les réseaux de pétri stochastiques ajoutent de l'indéterminisme et des probabilités de franchissement de transitions. [Belounnar, 2011]

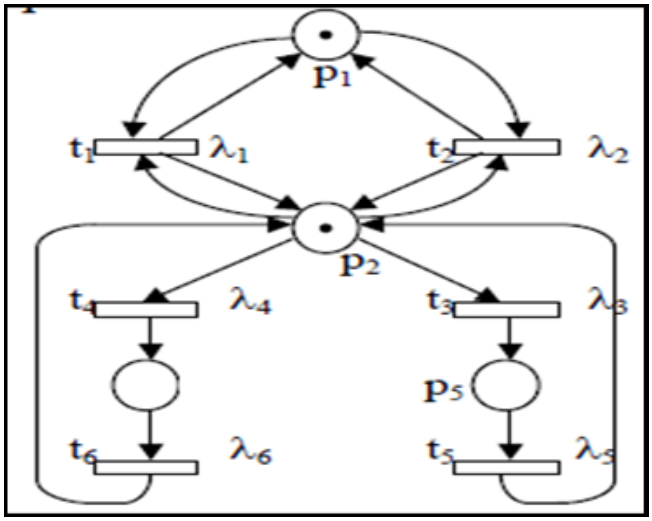

FIGURE 3.19 – Réseaux de pétri stochastique.

3.5.5 Les réseaux de pétri avec arc inhibiteur :

Une autre extension des réseaux ordinaires consiste à permettre de tester l'absence de marques dans une place, alors que lors d'un franchissement classique, on vérifie au contraire la présence d'une marque qui est consommée. Lorsqu'une place en entrée est reliée à une transition par un arc inhibiteur, cette transition n'est franchissable que si la place est vide (à ceci peut s'ajouter les conditions sur les autres places naturellement). Lors du franchissement la place en question reste vide. [Rahab, 2011]

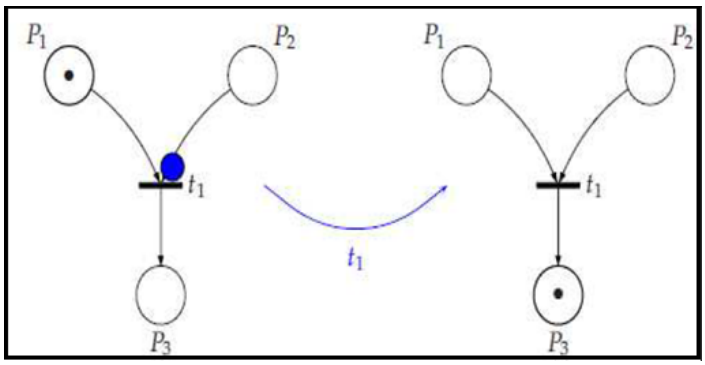

FIGURE 3.20 – Un réseau d pétri à arc inhibiteur.

3.6 Utilisation des réseaux de pétri

Dans certains cas, les réseaux de pétri ordinaires ne peuvent exprimer toutes les propriétés que nous voudrons modéliser, et de ce fait, des extensions s'avèrent utiles afin de pallier à ces insuffisances. Parmi les extensions les plus utilisées, nous citons : [Bouarioua, 2013]

• Les réseaux de pétri généralisés : Un réseau de pétri généralisé est un réseau de pétri dans lequel des poids (nombres entiers strictement positifs) sont associés aux arcs.

• Les réseaux de pétri temporisés : C'est une extension temporelle des réseaux de pétri.

• Les réseaux de pétri colorés : Dans un réseau de pétri coloré, on associe une valeur à chaque jeton.

• Les réseaux de pétri continus : Le nombre de jetons dans un réseau de pétri continu est un réel positif. Le franchissement s'effectue comme un flot continu en introduisant la notion de vitesse traduite par le nombre de marques franchies pendant une unité de temps.

Quelques utilisations des réseaux de pétri dans les différents domaines, non seulement informatiques, ont été résumées dans la table 3.1

Réseau de Pétri ordinaire	- Modélisation des systèmes - Modélisation des processus d'affaires - Gestion des flux. - Programmation concurrente. - Génie de la qualité - Diagnostic
Réseau de Pétri généralise	- Gestion des flux complexes. -Modélisation de chaines logistiques. -Utilisation pour les techniques quantitatives.
Réseau de Pétri temporisé	- Gestion du temps. -Modélisation d'attentes.
Réseau de Pétri coloré	- Modélisation des systèmes de collaboration.
Réseau de Pétri continu	- Modélisation des réactions chimiques.

TABLE 3.1 – L'utilisation des réseaux de pétri.

3.7 Propriétés des réseaux de pétri

Les propriétés génériques informent le développeur sur le comportement général du réseau de pétri. Celles-ci doivent être complétées par l'analyse des propriétés spécifiques du système modélisé. Ces propriétés sont souvent spécifiées par des formules logiques telles que la logique temporelle linéaire (LTL) et la logique temporelle arborescente (CTL). La vérification des propriétés nécessite la construction du graphe d'accessibilité. [Bouarioua,2013]

3.7.1 Propriétés génériques

La vivacité, le caractère borné et la réinitialisabilité sont les principales propriétés génériques qui peuvent être automatiquement vérifiées dans un réseau de pétri.

a . Vivacité et blocage :

Un réseau de pétri est vivant si et seulement si toutes ses transitions sont vivantes. Une transition t d'un réseau de pétri ayant M0 comme marquage initial est dite vivante si pour tout marquage M du réseau de pétri atteignable à partir de M0, nous pouvons toujours trouver une séquence franchissable de transitions dans laquelle la transition t figure. La vivacité d'un réseau de pétri dépend de son marquage initial M0. Un réseau

de pétri vivant garantit l'absence de blocages et de parties mortes (non atteintes) dans la structure du réseau. Il garantit ainsi la possibilité de toujours atteindre les services du système modélisé dans le réseau.

b . Réseau borné , Réseau Sauf :

Un réseau de pétri est borné si et seulement si toutes ses places sont bornées. Une place p est bornée si pour tout marquage M, le nombre de jetons dans la place est inférieur à une constante k : $\forall$ M, M(p) < k. Le caractère borné d'un réseau de pétri renseigne sur les valeurs limites des ressources demandées par le système. Si un réseau est borné, et la borne est égale à 1, alors il est dit sauf.

c . Conflit :

Un conflit structurel correspond à un ensemble d'au moins deux transitions t1 et t2 avec une place d'entrée en commun. Ceci est noté comme suit : k = < p , t1,t2,. . . ,tn >. Un conflit effectif est l'existence d'un conflit structurel k, et d'un marquage M, tel que le nombre de marques dans p est inférieur au nombre de transitions de sortie de p qui sont validées par M.

d . Réinitialisabilité :

Un réseau de pétri est réinitialisable si et seulement si pour tout marquage M, il existe une séquence de transitions qui permet de revenir au marquage initial M0. Cette propriété renseigne sur le fonctionnement répétitif, ce qui est pertinent pour la majorité des systèmes interactifs.

3.7.2 Propriétés spécifiques

Les propriétés spécifiques sont regroupées en quatre types : Les propriétés d'accessibilité, de sûreté, de vivacité et d'équité.

• L'accessibilité (reachability) : détermine si une situation est accessible ou non à partir du graphe d'accessibilité.

• La sûreté (safety) : certaines situations ne devront jamais être atteintes.

• La vivacité (liveness) : une situation finira tôt ou tard par avoir lieu.

• L'équité (fairness) : une situation aura lieu une infinité de fois.

3.8 Modélisation des systèmes concurrents

L'avantage des réseaux de pétri réside dans leur capacité à modéliser un grand nombre de comportements dans les systèmes complexes. Parmi ces comportements, nous trouvons le parallélisme, la synchronisation, le partage de ressources, la mémorisation, la lecture d'informations, la limitation de capacité de stockage, etc.
[Bouarioua, 2013]

• **Le parallélisme :** Le parallélisme est défini comme l'évolution simultanée de plusieurs processus dans un même système. Dans un réseau de pétri, le parallélisme est déclenché avec une transition ayant plusieurs places de sortie, comme présenté dans la figure 3.21.

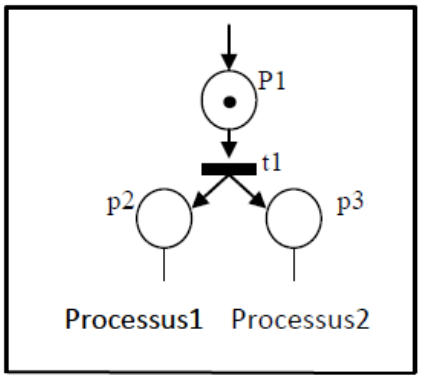

FIGURE 3.21 – Réseau de pétri avec parallélisme .

Le franchissement de la transition t1 met un jeton dans la place P2, et un jeton dans la place P3, ce qui marque le déclenchement du processus 1 et du processus 2 en parallèle.

• **La synchronisation :** Il existe deux types de synchronisation : la synchronisation mutuelle (rendez-vous) et la synchronisation par signal (sémaphores).

Synchronisation mutuelle : Cette synchronisation permet de synchroniser les opérations de deux processus comme le montre la figure 3.22.

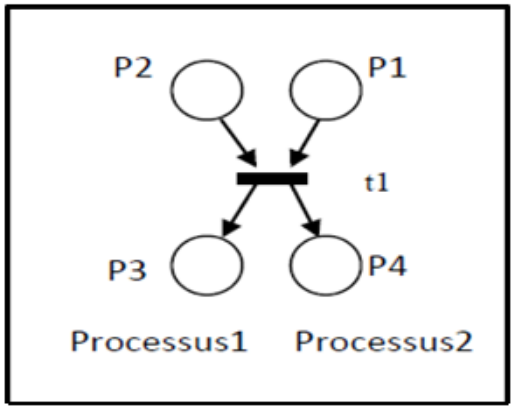

FIGURE 3.22 – Réseau de pétri avec synchronisation mutuelle.

.

Pour que la transition t1 franchisse, il faut que la place P1 qui correspond au processus1 et la place P2 qui correspond au processus 2 contiennent chacune au moins un jeton. Autrement, si par exemple la place P1 ne contient pas de jetons, le processus2 reste bloqué sur la place P2 ; il attend que le processus 1 réussisse à obtenir un jeton dans la place p1 au cours de son évolution.

Synchronisation par signal : Les opérations du processus 2 se poursuivent à condition que le processus 1 ait atteint un certain niveau dans son évolution. Ceci n'est pas le cas du processus 1 qui ne dépend pas de l'avancement des opérations du processus 2. Un exemple est illustré dans la figure 3.23.

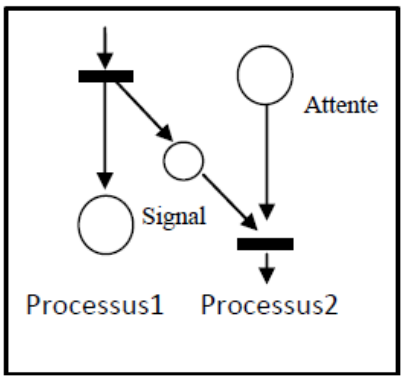

FIGURE 3.23 – Réseau de pétri avec synchronisation par signal.

Lorsque la place "Signal" est marquée et la place "Attente" ne l'est pas, ceci se traduit par un processus 1 qui a envoyé le signal que le processus 2 n'a pas encore reçu. Si, à l'inverse, la place "Signal" n'est pas marquée et la place "Attente" est marquée, cela signifie que le processus 2 est en attente du signal.

• **Le partage de ressources :** C'est un type de modélisation lié à un système au sein duquel plusieurs processus partagent une même ressource en utilisant le principe de l'exclusion mutuelle.

Dans la Figure 3.24, Le jeton dans la place P0 présente une ressource mise en commun entre le processus 1 et le processus 2. Le franchissement de la transition T17 lors de l'évolution du processus 1 entraîne la consommation du jeton présenté dans la place P0. La ressource que constitue ce jeton n'est alors plus disponible pour l'évolution du processus 2. Lorsque la transition T18 est franchie, un jeton est alors est placé dans la place P0 : la ressource devient alors disponible pour l'évolution des deux processus. [Dehimi, 2014]

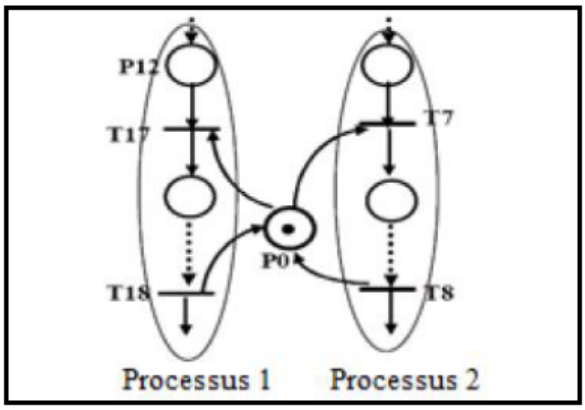

FIGURE 3.24 – Réseau de pétri avec partage de ressource.

• **La mémorisation :** Une place sans entrée ou sans sortie peut être utilisée dans tous les modèles pour compter le nombre de tirs d'une transition.

Dans la figure 3.25, les places "Attente" et "Compteur" peuvent indiquer combien d'instances d'un processus sont en attente. [Bouarioua, 2013]

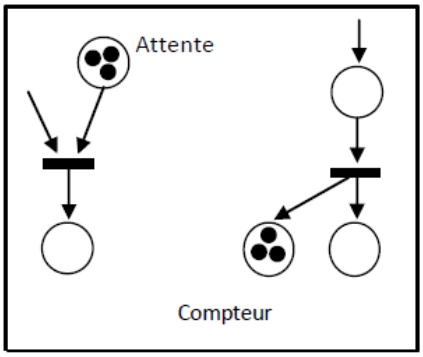

FIGURE 3.25 – Réseau de pétri illustrant le cas de la mémorisation.

Conclusion :

Dans ce chapitre, nous avons présenté les réseaux de pétri de haut niveau .On a vus qu'un réseau de pétri aide à représenter un système, en associant les états et les activités à des places et les événements à des transitions. Les arcs des réseaux de pétri servent à lier la transition avec des places et les places avec des transitions. Ils ne peuvent pas lier deux places ou deux transitions. Ainsi, le marquage consiste à disposer un nombre entier (positif ou nul) de marques ou jetons dans chaque place du réseau de Pétri. Les RdP sont particulièrement bien adaptés à la description des aspects dynamiques ou comportementaux d'un système. Des concepts tels que la concurrence ou la synchronisation entre interactions s'expriment aisément dans le cadre de ce formalisme. Toutes les notions clarifiées dans ce chapitre vont aider à comprendre le noyau de notre projet. Le chapitre suivant va se concentrer sur la transformation de graphes.

CHAPITRE 4

Transformation De Graphes

Sommaire

Introduction

L'Ingénierie Dirigée par les Modèles (IDM, ou MDE : Model Driven Engineering) offre un cadre méthodologique et technologique qui permet d'unifier différentes façons de faire dans un processus homogène. Il est ainsi possible d'utiliser la technologie la mieux adaptée à chacune des étapes du développement du logiciel, tout en ayant un processus global de développement qui soit unifié dans un paradigme unique. L'IDM permet cette unification grâce à l'utilisation importante des modèles (qui peuvent être exprimés dans des formalismes différents) et des transformations automatiques entre les modèles. L'utilisation intensive de modèles permet un développement souple et itératif, grâce aux raffinements et enrichissements par transformations successives. Par ailleurs, les transformations permettent de passer d'un espace technique à un autre (par exemple d'UML vers les graphes ou vers du XML). Ainsi, les transformations permettent de

choisir l'espace technique et le formalisme le plus adapté à chaque activité.

Dans le cadre de ce travail nous intéressons à la transformation de modèles. Les diagrammes de classes orientés aspects seront transformés vers les réseaux de pétri. Le passage du modèle source vers le modèle cible sera réalisé par la transformation de graphes à l'aide de l'outil AToM^3. Le but de ce chapitre est de donner une idée sur la transformation de modèles, notamment la transformation de graphes, sur les grammaires de graphes et un aperçu sur l'outil utilisé l'AToM^3.

4.1 L'architecture dirigée par les modèles

L'architecture dirigée par les modèles ou MDA ("Model Driven Architecture") est une démarche de réalisation de logiciel, proposée et soutenue par l'OMG. C'est une variante particulière de l'ingénierie dirigée par les modèles. [Elmansouri, 2009]

La figure suivante représente les différentes couches de spécification de la démarche MDA.

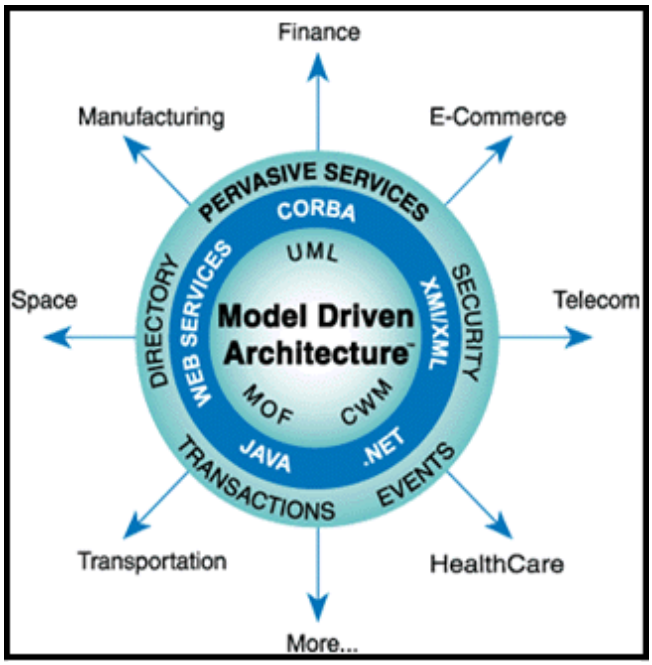

FIGURE 4.1 – Model Driven Architecture (OMG).

4.1.1 Le principe du MDA

L'approche MDA (Model Driven Architecture), définie par l'OMG en 2000, est basée sur l'utilisation des modèles et essaye de répondre aux comment, quand, quoi et pourquoi modéliser. MDA inclut la définition de plusieurs standards, notamment UML (Unified Modeling Language), MOF (Meta Object Facility) et XMI (XML Metadata Interchange). Le principe clé de MDA consiste en l'utilisation de modèles aux différentes phases du cycle de développement d'une application. Plus précisément c'est :

- modèles d'exigences (CIM : Computation Independent Model).
- d'analyse et de conception (PIM : Platform Independent Model).
- de code (PSM : Platform Specific Model). [Aouag, 2014]

a) Le modèle des besoins CIM :

La première chose à faire lors de la construction d'une nouvelle application est bien entendu de spécifier les exigences du client. Bien que très en amont, cette étape doit fortement bénéficier des modèles. L'objectif est de créer un modèle d'exigences de la future application. Un tel modèle doit représenter l'application dans son environnement afin de définir quels sont les services offerts par l'application et quelles sont les autres entités avec lesquelles elle interagit. La création d'un modèle d'exigences est d'une importance capitale. Cela permet d'exprimer clairement les liens de traçabilité avec les modèles qui seront construits dans les autres phases du cycle de développement de l'application, comme les modèles d'analyse et de conception. [Blanc, 2005]

b) Le modèle d'analyse et de conception abstraite PIM :

Une fois le modèle d'exigences réalisé, le travail d'analyse et de conception peut commencer. Dans l'approche MDA, cette phase utilise elle aussi un modèle. L'analyse et la conception sont les étapes où la modélisation est la plus présente, d'abord avec les méthodes **Merise** et **Coad/Yourdon** puis avec les méthodes **objet Schlear et Mellor**, **OMT, OOSE** et **Booch**. Ces méthodes proposent toutes leurs propres modèles. Aujourd'hui, le langage UML s'est imposé comme la référence pour réaliser tous les modèles d'analyse et de conception. [Blanc, 2005]

c) Le modèle de code PSM :

Une fois les modèles d'analyse et de conception réalisés, le travail de génération de code peut commencer. Cette phase, la plus délicate du MDA, MDA considère que le code d'une application peut être facilement obtenu à partir de modèles de code. La différence principale entre un modèle de code et un modèle d'analyse ou de conception réside dans le fait que le modèle de code est lié à une plate-forme d'exécution. Ces modèles de code sont appelés des PSM. Les modèles de code servent essentiellement à faciliter la génération de code à partir d'un modèle d'analyse et de conception. Ils contiennent toutes les informations nécessaires à l'exploitation d'une plate-forme

d'exécution. Pour MDA, le code d'une application se résume à une suite de lignes textuelles, comme un fichier Java, alors qu'un modèle de code est plutôt une représentation structurée incluant, par exemple les concepts de boucle, condition, instruction, composant, événement, etc. L'écriture de code à partir d'un modèle de code est donc une opération assez triviale. [Blanc, 2005]

4.1.2 Transformation de modèles

Elle est définie comme étant le processus de convertir un modèle d'un système à un autre modèle du même système. [Elmansouri, 2009]

La figure suivante donne un aperçu global sur le processus de transformation de modèles de l'approche MDA.

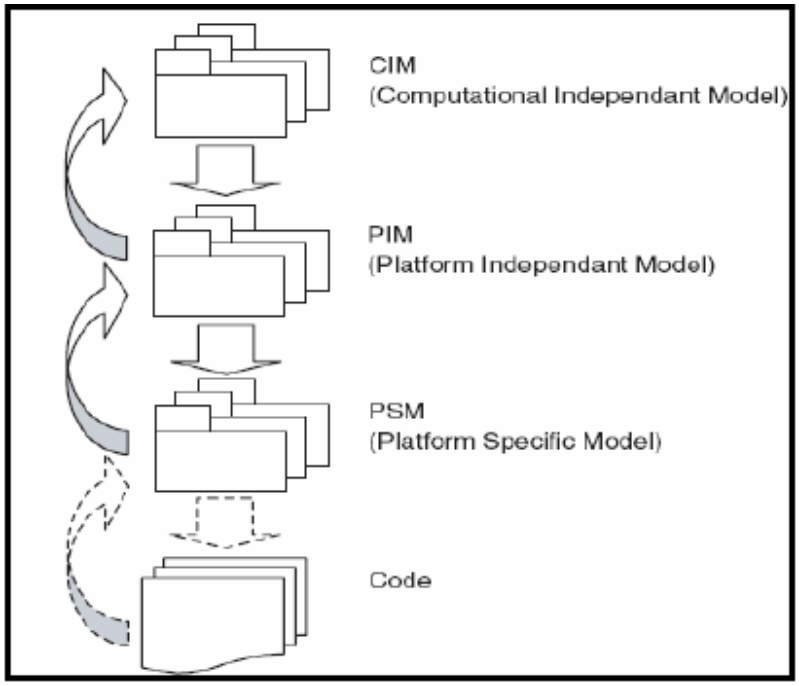

FIGURE 4.2 – Le processus de transformation de modèles dans l'approche MDA .

Par exemples :

• Transformer un automate d'état fini non déterministe vers un automate d'état fini déterministe .

• Transformer un diagramme UML vers un réseau de pétri.
• Transformer un processus métier vers un réseau de pétri.
• .....

4.1.3 Méta-modélisation et transformation

Comme un programme, un modèle destiné à être utilisé par une machine ne doit pas avoir une ambigüité d'interprétation. Il doit être exprimé selon des règles bien définies. Justement, la démarche de méta-modélisation s'attelle à définir des formalismes pour les langages de modélisation [kholladi, 2009].
Ce qui garantit le traitement correct de tout modèle conforme à ce formalisme.
• **Méta-modèle :** Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle. Autrement dit, le méta-modèle représente (modélise) les entités d'un langage, leurs relations ainsi que leurs contraintes, c'est-à-dire une spécification de la syntaxe du langage.[Kerkouche, 2011]
• **Le méta-méta-modèle :**
Le méta-méta-modèle est un méta-modèle pour les méta-modèles utilisé tout naturellement pour désigner ce méta-modèle particulier. Le langage utilisé au niveau du méta-méta-modèle doit être suffisamment puissant pour spécifier sa propre syntaxe abstraite et ce niveau d'abstraction demeure largement suffisant (méta-circulaire).Chaque élément du modèle est une instance d'un élément du méta-modèle .[Kerkouche, 2011]

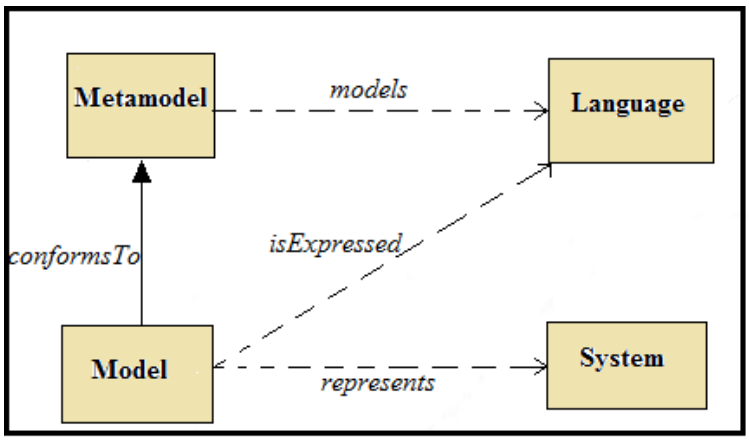

FIGURE 4.3 – Relations entre système, modèle, méta-modèle et langage .

4.1.4 Modèle d'architecture MDA à quatre niveaux

Le MDA se résume à la pyramide suivante avec 4 niveaux d'abstraction.
[Bahri, 2011]

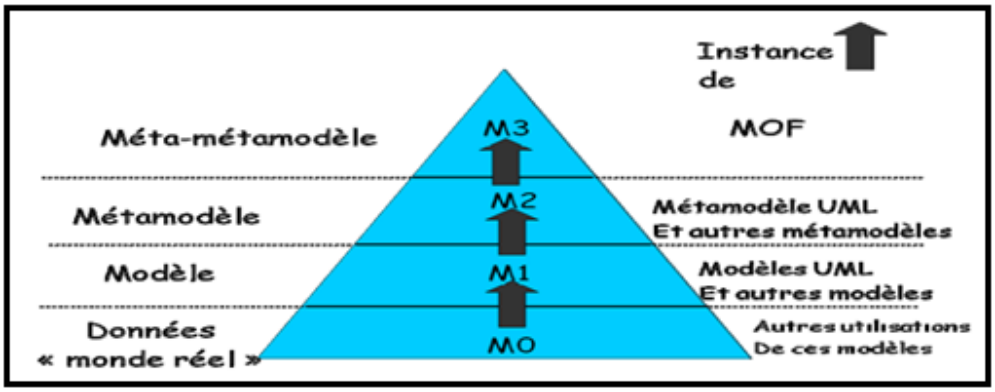

FIGURE 4.4 – Les quatre niveaux d'abstraction pour MDA.

Le niveau M0 : Niveau des instances des modèles. Il définit des informations pour la modélisation des objets du monde réel.

Le niveau M1 : Ce niveau représente toutes les instances d'un méta-modèle. Les modèles du niveau M1 doivent être exprimés dans un langage défini au niveau M2. UML est un exemple de modèles du niveau M1.

Le niveau M2 : Ce niveau représente toutes les instances d'un méta-méta-modèle. Il est composé de langages de spécifications de modèles d'information. Le méta-modèle UML qui est décrit dans le standard UML et qui définit la structure interne des modèles UML, appartient au niveau M2.

Le niveau M3 : Ce niveau définit un langage unique pour la spécification des méta-modèles. Le MOF élément réflexif du niveau M3, définit la structure de tous les méta-modèles du niveau M2.

Dans l'architecture proposée par l'OMG, UML se situe au troisième niveau ; un langage générique de description des modèles est proposé : le MOF (Meta Object Facility). La figure 4.5 présente un simple scénario d'une transformation avec un modèle en

entrée (source) et un modèle en sortie (cible). Les deux modèles sont conformes avec leurs méta-modèles. Une transformation est définie en respectant les méta-modèles. L'outil de transformation exécute cette définition sur des modèles concrets.

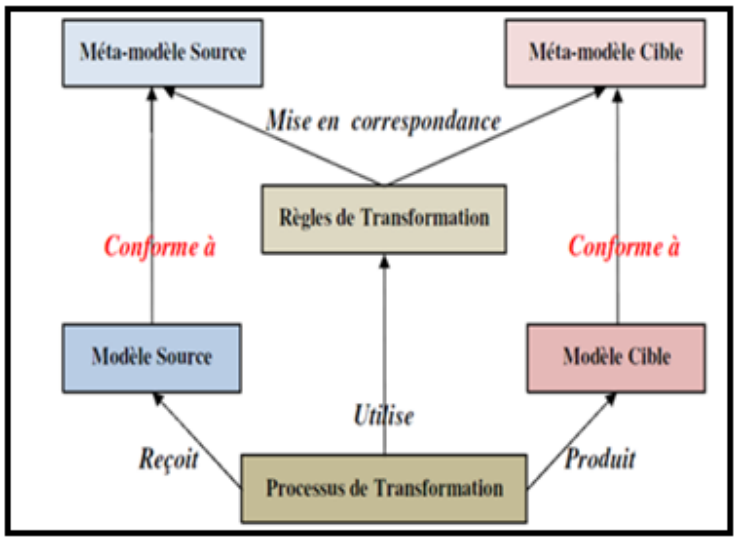

FIGURE 4.5 – Concepts de base de la transformation de modèles .

Généralement, une transformation peut avoir des modèles sources et cibles multiples. En outre, les méta-modèles source et cible peuvent être les mêmes dans certaines situations. Il existe dans la littérature trois types de transformations : [Hachichi, 2013]

1. Les transformations verticales : La source et la cible d'une transformation verticale sont définies à différents niveaux d'abstraction. Une transformation qui baisse le niveau d'abstraction est appelée un raffinement (PIM vers PSM). Une transformation qui élève le niveau est appelée une abstraction (PSM vers PIM).

2.Les transformations horizontales : Une transformation horizontale modifie la représentation source tout en conservant le même niveau d'abstraction (PIM vers PIM) ou (PSM vers PSM). La modification peut être l'ajout, la modification, la suppression ou la restructuration d'informations.

3. Les transformations obliques : Une transformation oblique combine une transformation horizontale et une verticale. Ce type de transformation est notamment utilisé par les compilateurs, qui effectuent des optimisations du code source avant de générer le code exécutable.

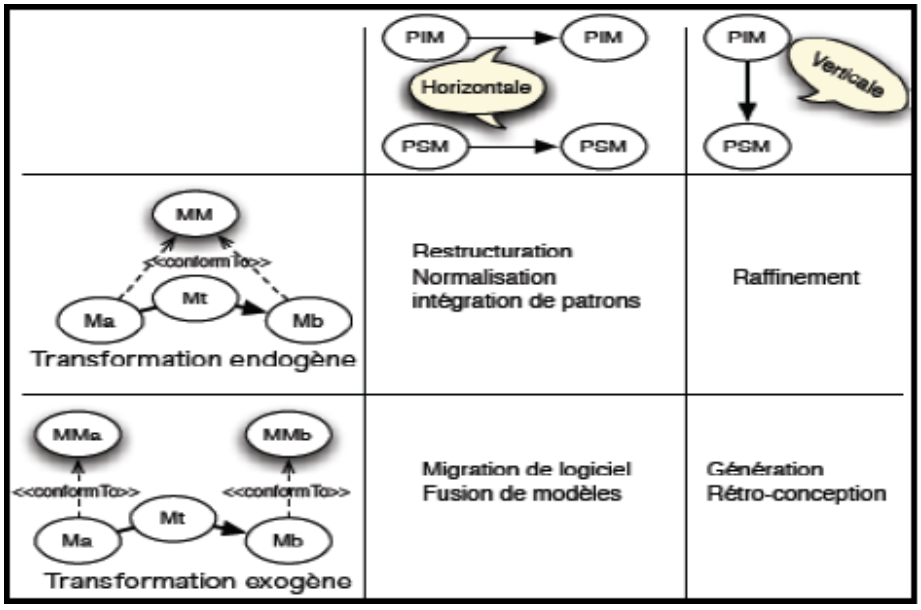

FIGURE 4.6 – Les types de transformation et leurs principales utilisations.

4.1.5 Classification des approches de transformation de modèles

On va présenter une classification des approches de transformation en se basant sur le travail de Czarnecki qui décompose la transformation de modèle en deux catégories : les transformations de type **modèle vers code** et les transformations de type **modèle vers modèle**.

4.1.5.1 Une transformation de type modèle vers code

Il existe deux approches de transformations de type **modèle vers code** : les approches basées sur le principe du ***visiteur*** **(Visitor-based approach)** et les approches basées sur le principe des ***patrons*** **(Template-based approach)**. [Hettab, 2009]
A.Les approches basées sur le principe du visiteur consistent à réduire la différence de sémantique entre le modèle et le langage de programmation cible en se basant sur mécanismes visiteurs dont le quel en ajoutant des éléments dans le modèle et le code est obtenu en parcourant le modèle enrichi pour créer un flux de texte.

B.Les approches basées sur le principe des patrons sont les plus utilisés actuellement et la majorité des outils MDA couramment disponibles supporte ce principe de génération de code à partir de modèle. Parmi les outils basés sur ce principe, on peut citer : **OptimalJ**, **XDE** (qui fournissent la transformation modèle vers modèle aussi), **JET**, **ArcStyler** et **AndroMDA**(un générateur de code qui se repose notamment sur la technologie ouverte **Velocity** pour l'écriture des patrons). Le principe de ces approche repose sur l'utilisation des morceaux de méta-code obtenu a partir du code cible et les utilisés pour accéder aux informations du modèle source.

4.1.5.2 Une transformation de type modèle vers modèle

Depuis l'apparition de MDA, Les transformations de type modèle vers modèle ne cessent évolué jusqu'a nos jours. Il existe un grand espace d'abstraction entre un PIM et un PSM, alors l'utilisation des modèles intermédiaires au lieu d'aller directement d'un PIM vers un PSM est une chose recommandée, ces modèles utilisés pour des raisons d'optimisation ou de déboguage. Ces transformations sont utiles pour le calcul des différentes vues du système, leur synchronisation et pour la vérification et la validation. [Hettab, 2009]

4.1.5.3 Structure d'une transformation

Une transformation de modèle est principalement caractérisée par la combinaison des éléments suivants : des règles de transformation, une relation entre la source et la cible, un ordonnancement des règles, une organisation des règles, une traçabilité et une direction. [Hettab, 2009]

A. Spécification : les pré-conditions et les post-conditions exprimées en OCL (Object Constraint Language) ou un autre langage ce spécification est utilise dans certaines approches.

B. Règles de transformation : une règle de transformation qui possède une logique de forme déclarative ou impérative pour exprimer des contraintes, elle se compose en deux parties : la partie gauche qui s'appelle **LHS** (Left HandSide) et qui accède au modèle source, et une partie droite **RHS** (Right Hand Side) qui accède au modèle cible.

C. Organisation des règles : il existe plusieurs façons pour organiser les règles de

transformation ; on peut les organiser de façon modulaire, d'un ordonnancement explicite ou d'une structure dépendante du modèle source ou du modèle cible.
D. Ordonnancement des règles : il existe deux types d'ordonnancements des règles ; l'ordonnancement implicite et l'ordonnancement explicite. Dans le premier type l'ordre des règles est défini par l'outil de transformation elle-même, par contre dans le deuxième type (ordonnancement explicite) il existe des mécanismes permettent de spécifier l'ordre d'exécution des règles.
E. Relation entre les modèles source et cible : pour certains types de transformations, la création d'un nouveau modèle cible est nécessaire, par contre dans d'autres type, la source et la cible forme le même modèle, ce qui revient en fait à une modification de modèle.
F. Direction : dans le point de vue de la direction de transformation, deux types de direction sont existe : les transformations unidirectionnelles et les transformations bidirectionnelles. Dans le premier cas, l'obtention du modèle cible est basé uniquement sur modèle source, par contre dans le second cas, une synchronisation entre les modèles source et cible est possible.
G. Traçabilité : la traçabilité est le processus d'archivage des corrélations existes entre les éléments des modèles source et cible. Certaines approches supportent la traçabilité en fournissant des mécanismes dédiées pour elle. Dans les autres cas, le développeur doit implémenter la traçabilité de la même manière qu'il crée n'importe quel autre lien dans un modèle.

4.1.5.4 Les approches pour la définition des transformations

la transformation modèle à modèle se base sur une structure variée. Par conséquent, Plusieurs approches tentent de définir ce type de transformation. Dans la littérature, on distingue généralement cinq approches :

- les approches par manipulation directe.
- les approches relationnelles.
- les approches basées sur la transformation de graphes.
- les approches basées sur la structure.
- les approches hybrides. [Bahri, 2011]

Dans notre travail, on s'intéresse particulièrement aux approches basées sur la transformation de graphes, du fait que notre objectif est de transformer les diagrammes UML 2.0 qui sont des graphes vers les réseaux de pétri qui eux aussi sont sous forme de graphes.

4.1.6 Les outils d'MDA

Pour obtenir une telle efficacité, plusieurs outils conceptuels sont mis à disposition. La technologie MDA (Model Driven Architecture) est supportée par l'OMG (Object-Management Group), qui propose également UML (Unified Modeling Language) et Corba (Object Request Broker). Ces outils sont :

A. UML largement utilisé par ailleurs qui permet une mise en œuvre aisée de MDA en offrant un support connu.

B. XMI (XML Metadata Interchange) qui propose un formalisme de structuration des documents XML de telle sorte qu'ils permettent de représenter des méta-données d'application de manière compatible.

C. MOF (Meta Object Facility) spécification qui permet le stockage, l'accès,la manipulation, la modification de méta-données. Le MOF permet une unification de l'expression des méta-modèles, qu'ils soient ensuite utilisés comme profils UML ou non .

D.CWM base de données pour méta-données. [Hettab, 2009]

4.2 Transformation de graphes

Dans cette partie, nous allons présenter quelques notions sur les graphes, ensuite nous allons décrire avec détaille les transformations et les grammaires de graphes.

4.2.1 Notion de graphe

Un graphe est constitué de sommets ou nœuds qui sont reliés par des arêtes. Plus formellement, on appelle graphe G = < S ; A >

- S : ensemble fini non vide d'éléments appelés sommets ou nœuds.
- A : ensemble fini non vide de paires de sommets appelés arcs.
- $A \subseteq X \times X = \{(s, t) \mid s , t \in S \}$

Chaque arc de A relie deux sommets de S. Il existe deux types principaux de graphes : les graphes non orientés (la figure 4.8) et les graphes orientés (la figure 4.7).
[Aouag, 2014]

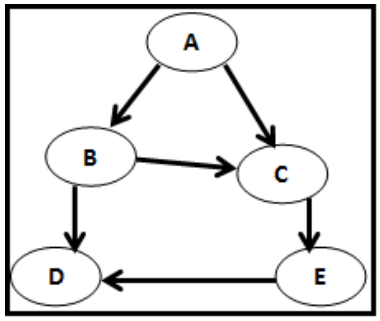

FIGURE 4.7 – Graphe orienté .

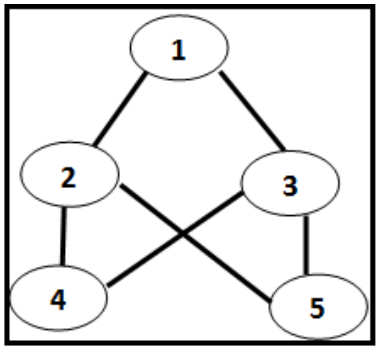

FIGURE 4.8 – Graphe non orienté .

Graphe étiqueté : Un graphe étiqueté est un graphe orienté dans lequel les arcs possèdent un ensemble non vide d'étiquettes.
graphe attribué : est un graphe qui peut contenir un ensemble prédéfini d'attributs.
les sommets adjacents : Deux sommets sont adjacents s'ils sont reliés par une arête. .
l'ordre du graphe : le nombre de sommets présents dans un graphe.
le degré du graphe : Le degré d'un sommet est le nombre d'arêtes dans ce sommet. Si le graphe est orienté le degré de sommet est le nombre d'arcs entrants et sortants dans

ce sommet.

Le sous-graphe : un sous-graphe d'un graphe G est un graphe G' composé de certains sommets de G, ainsi que toutes les arêtes qui relient ces sommets. [Aouag, 2014]

4.2.2 Grammaires de graphes

En générale, la transformation de graphes est le processus de choisir une règle d'un ensemble indiqué, appliquer cette règle à un graphe et réitérer le processus jusqu'à ce qu'aucune règle ne puisse être appliquée. La transformation de graphes est spécifiée sous forme d'un modèle de grammaires de graphes, ces dernières sont une généralisation, pour les graphes, des grammaires de Chomsky. Elles sont composées de règles dont chacune est composée d'un graphe de coté **gauche (LHS)** et d'un graphe de coté **droit (RHS).** [Elmansouri, 2009]

Définition : Une grammaire de graphes est une structure GG , généralement définie par un triplet : GG = (P ; S ; T) où :

P : ensemble de règles.

S : un graphe initial.

T : ensemble de symboles terminaux T. [Aouag, 2014]

Exemple : Règle permettant d'éliminer le non déterminisme dans un automate d'état fini déterministe.

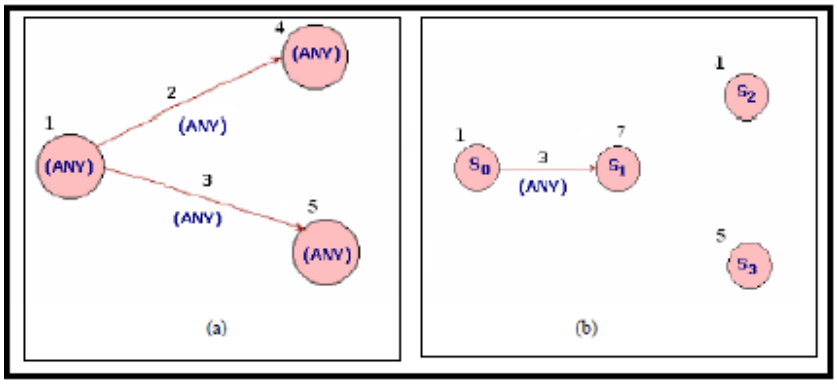

FIGURE 4.9 – (a) Partie gauche d'une règle (b) Partie droite d'une règle .

Cette règle permet d'éliminer le non déterminisme lors de la conversion d'un automate non déterministe en un automate déterministe. Une grammaire de graphes distingue les

graphes non terminaux, qui sont les résultats intermédiaires sur lesquels les règles sont appliquées, des graphes terminaux dont on ne peut plus appliquer de règles, on dit que ces derniers sont dans le langage engendré par la grammaire. Pour vérifier si un graphe G est dans les langages engendrés par une grammaire de graphe, il doit être analysé. Le processus d'analyse va déterminer une séquence de règles dérivant G.

4.2.2.1 Le principe de règles

Une règle de transformation de graphe est définie par : r = (L, R, K, glue, emb, cond) Elle consiste en :

- Deux graphes L graphe de côté gauche et R graphe de côté droit.
- Un sous graphe K de L.
- Une occurrence glue de K dans R qui relie le sous graphe avec le graphe de côté droit.
- Une relation d'enfoncement qui relie les sommets du graphe de côté gauche et ceux du graphe du coté droit.
- Un ensemble cond qui spécifie les conditions d'application de la règle.

[kholladi, 2009]

4.2.2.2 Application des règles

L'application d'une règle r= (L, R, K, glue, emb, cond) à un graphe G produit un graphe résultant H. Le graphe H fourni, peut être obtenu depuis le graphe d'origine G en passant par les cinq étapes suivantes :

1. Choisir une occurrence du graphe de coté gauche L dans G.

2. Vérifier les conditions d'application d'après cond.

3. Retirer l'occurrence de L (jusqu'à K) de G ainsi que les arcs pendillé, c-à-d tout les arcs qui ont perdu leurs sources et/ou leurs destinations. Ce qui fourni le graphe de contexte D de L qui a laissé une occurrence de K.

4. Coller le graphe de contexte D et le graphe de coté droit R suivant l'occurrence de K dans D et dans R. c'est la construction de l'union de disjonction de D et R et, pour chaque point dans K, identifier le point correspondant dans D avec le point correspondant dans R.

5. Enfoncer le graphe du côté droit dans le graphe de contexte de L suivant la relation d'enfoncement emb : L'application de r sur un graphe G pour fournir un graphe H est appelée une dérivation directe depuis G vers H à travers r, elle est dénotée par G$\Longrightarrow$H ou simplement par G $\Longrightarrow$H .
En donnant les notions de règle et de dérivation directe comme étant les concepts élémentaires de la transformation de graphe, ont peut définir les systèmes de transformation de graphe, les grammaires de graphe et la notion de langages engendrés. [kholladi, 2009] [Elmansouri, 2009]

4.2.3 Outils de transformation de graphes

Il existe plusieurs outils de transformation de graphes. On peut citer quelques exemples d'outils de transformation de graphes : AGG , TGG, FUJABA, ATOM^3 , VIATRA , Eclipse Modeling Galileo, GreAT, Progres, Booggie etc. Notre choix est porté sur ATOM^3 à cause des avantages qu'il présente. Nous pouvons citer parmi ces avantages :

1. sa simplicité.
2. sa disponibilité.
3. il est multi paradigmes. [kholladi, 2009]

4.3 ATOM ^{3}

ATOM^3[Atom3] est un outil visuel pour la modélisation et la méta-modélisation multi formalismes. Comme il a été implémenté en Python, il peut être exécuté, sans aucun changement, sur toutes les plateformes où un interpréteur de Python est disponible (Linux, Windows et MacOS). Ces deux tâches principales sont la méta-modélisation et la transformation de modèles. Pour la méta-modélisation, il supporte la modélisation visuelle en utilisant le formalisme Entité-Relation (ER) ou le formalisme de diagrammes de classes d'UML2.0. Cela signifie que pour méta-modéliser de nouveaux formalismes, on peut utiliser le modèle ER ou le modèle des diagrammes de classes. Ici, le choix est porté sur l'utilisation du modèle des diagrammes de classes. Afin de pouvoir spécifier entièrement les formalismes de modélisation, les méta-formalismes peuvent être étendus par l'expression de contraintes, ce que les diagrammes de classes

d'UML2.0 ou le modèle ER ne peuvent pas exprimer. Les contraintes fournissent une vue sur la manière dont un constructeur est lié à un autre pour qu'il ait un sens. Les contraintes sont exprimées sous une forme textuelle. De ce fait, certains systèmes, dont AToM^3, utilisent OCL (Object Constraint Language) d'UML2.0. On peut utiliser du code Python pour exprimer les contraintes du moment que AToM^3 est implémenté en Python. Pour la transformation de modèles, AToM^3 supporte la réécriture de graphes qui utilise les règles de grammaire de graphes pour guider visuellement la procédure de transformation. Les règles sont spécifiées par l'utilisateur et ordonnées selon des critères dépendants des caractéristiques du modèle à transformer. AToM^3 offre la possibilité à l'utilisateur de créer une nouvelle grammaire, de charger une grammaire et de la modifier et d'exécuter une grammaire. L'exécution de la grammaire sur un modèle en entrée produit un modèle de sortie. [kholladi, 2009]

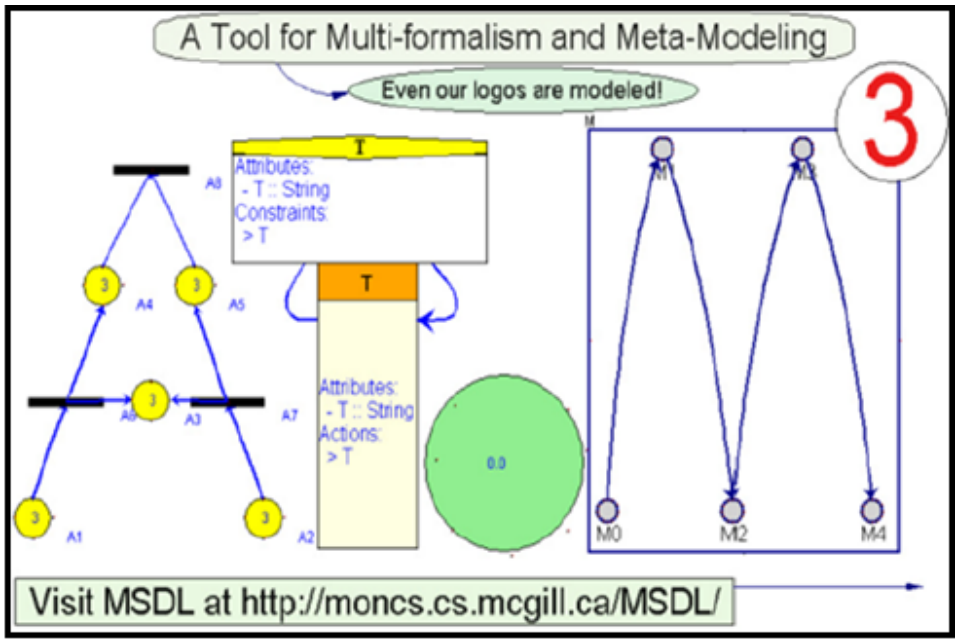

FIGURE 4.10 – Présentation de l'outil AToM^3.

4.3.1 La méta-modélisation avec AToM^3

Le méta-formalisme utilisé est le diagramme entités-associations. Un méta-modèle est spécifié par la fourniture de deux syntaxes. D'une part, une syntaxe abstraite et formelle. Elle sert à définir les entités, les relations et toutes les contraintes associées. Les entités et les relations sont définies en donnant tous les attributs nécessaires, alors que les contraintes sont exprimées textuellement en langage OCL . D'autre part, une syntaxe graphique permettant d'ajuster l'apparence graphique selon les notations appropriées. Une fois le méta-modèle d'un formalisme donné est défini, AToM^3 génère un environnement interactif pour manipuler ses entités. Ainsi, la conformité des modèles créés vis-à-vis la syntaxe spécifiée est assurée automatiquement. [Khalfaoui, 2014]

La Figure 4.11 montre le méta-modèle des automates à états finis, alors que la Figure 4.12 présente l'éditeur graphique généré automatique avec AToM^3 pour le formalisme des automates à états finis . [Kerkouche, 2011]

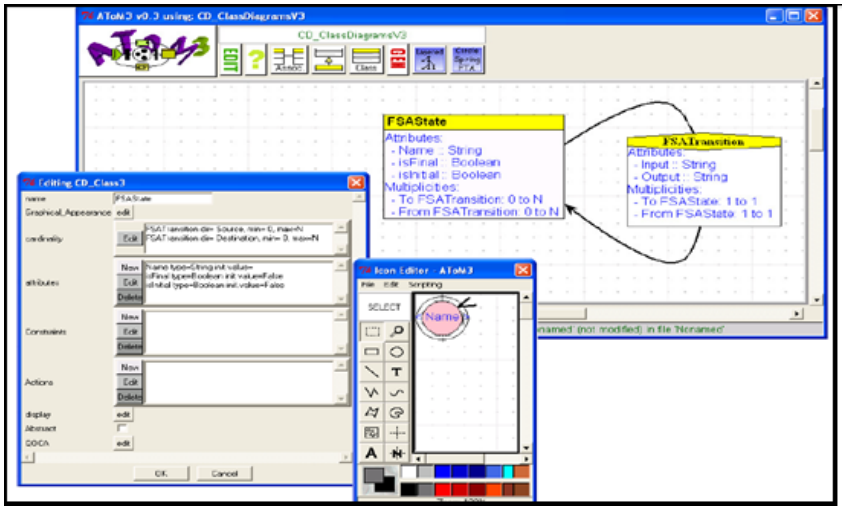

FIGURE 4.11 – Méta-modèle des automates à états fini.

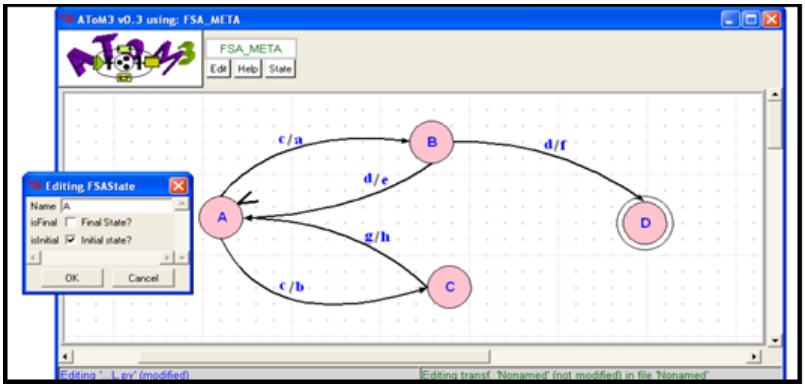

FIGURE 4.12 – Editeur graphique généré pour les automates à états finis.

4.3.2 La transformation de modèles

Dans AToM^3 un système de réécriture de graphes qui applique itérativement les règles d'une grammaire de graphes pour guider la procédure de transformation, est utilisé pour la transformation de modèles. Pour diriger le choix de la règle à appliquer l'utilisateur spécifie et classe ses règles selon des priorités bien étudiées. Selon l'ordre ascendant de leurs priorités toutes les règles sont testées, à chaque itération. Lors de l'application des règles, les attributs des éléments de la partie LHS doivent avoir des valeurs qui seront comparées avec les attributs des éléments du modèle pendant le processus de correspondance (matching). Un attribut peut prendre une valeur spécifique ou n'importe quelle valeur (<ANY>). En outre, les liens de traçabilité entre les éléments source et cibles sont représentés par des étiquettes sous forme de numéros. Si une étiquette d'un élément apparaît dans la partie LHS mais pas dans la partie RHS, cet élément sera supprimé lors de l'application de la règle. Si au contraire, une étiquette d'un élément n'apparaît que dans la partie RHS, cet élément sera créé suite à l'application de cette règle. Le nœud sera maintenu, si son étiquette apparaît à la fois dans la partie LHS et dans la partie RHS de la règle à appliquer, Suite à l'application d'une règle, les valeurs des attributs des éléments maintenus ou nouvellement crées par la règle seront définis. On distingue, dans l'outil AToM^3 plusieurs possibilités de le faire. Si l'élément est déjà présent dans la partie LHS, les valeurs de ses attributs peuvent copiées (<COPIED>). On a également la possibilité de leurs donner des valeurs spéci-

fiques ou de définir un programme Python pour calculer ces valeurs (<SPECIFIED>), éventuellement, en utilisant les valeurs des attributs source. A chaque règle peuvent être attaché, des actions à effectuer et des conditions supplémentaires, liées à son application. Outre les règles de transformations, dans la grammaire de graphe, on peut utiliser aussi une action initiale et une autre finale. L'action initiale (ou finale) spécifie les actions à exécuter avant (ou après) l'application des règles. AToM^3 offre à l'utilisateur la possibilité de créer, charger, modifier et exécuter une grammaire. Un modèle en sortie est le résultat de l'exécution de la grammaire sur le modèle en entrée. [Dehimi, 2014]

La Figure 4.13 représente l'application d'une règle dans une grammaire de graphes (1^{er} Règle) sur le modèle de la Figure 4.11 .

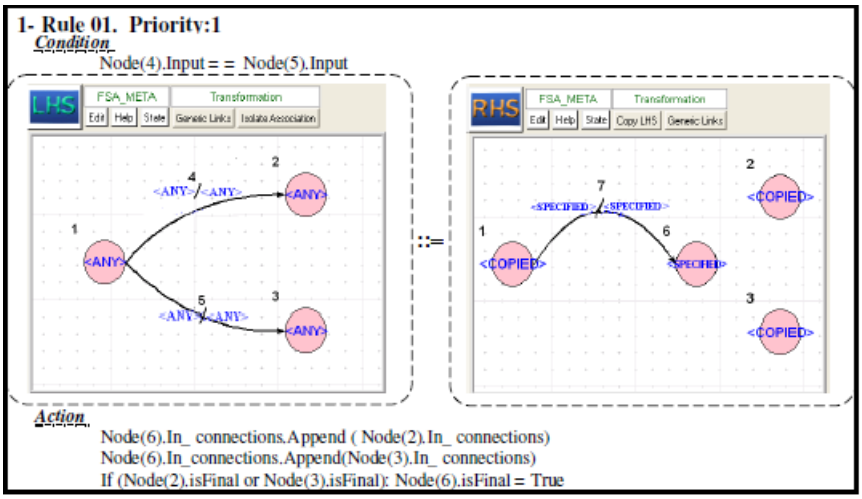

FIGURE 4.13 – une règle dans une grammaire de graphes.

Conclusion

Dans ce chapitre, nous avons présenté les concepts de base de la transformation de modèles qui est considéré comme la clé de la démarche MDA. Nous avons présenté les différentes approches existantes en se basant sur une classification proposée dans la littérature. Une introduction aux transformations de graphes qui représentent une approche de transformation de modèles a été donnée. Nous avons également brièvement présenté AToM^3, l'outil de transformation utilisé dans notre travail.Les concepts présentés dans ce chapitre constituent un package nécessaire pour la compréhension de nos contributions dans le cadre de ce mémoire et qui seront présentées dans le chapitre quatre « l'implémentation de notre travail ».

CHAPITRE 5

L'approche De Transformation Proposée

Sommaire

Introduction

L'utilisation d'UML 2.0 et RdP simultanément dans une méthode intégrée de développement de logiciel est justifié par leur complémentarité. UML 2.0 est utilisé pour la modélisation alors que le RdP est utilisée pour la vérification formelle.

Dans ce chapitre, nous proposons une approche de transformation de modèles UML 2.0 orienté aspect vers les réseaux de pétri (RdPs) en se basant sur la grammaire de graphes. Cette approche est automatisée à l'aide de l'outil de méta-modélisation AToM^3. La transformation de graphes est un système de réécriture de graphes qui applique les règles de grammaire de graphes sur son graphe initial jusqu'à ce que plus aucune règle ne soit applicable.

Les Diagrammes de Classes Orienté Aspect (DCOA) sont les diagrammes les plus courants dans la modélisation des systèmes orientée aspect. On les utilise pour modéliser la vue de conception statique d'un système .Pour l'essentiel, ils englobent la

modélisation de vocabulaire du système, la modélisation de collaboration ou la modélisation de schémas. Le diagramme de classe orienté aspect permet de modéliser les classes du système et leur relation indépendamment d'un langage de programmation particulier.

Tout d'abord pour atteindre notre objectif, nous allons définir des méta-modèles pour les diagrammes de classes orientés aspect et les réseaux de pétri avec l'outil AToM^3 qui va générer automatiquement un outil de modélisation visuel pour chaque formalisme, puis nous définissons la grammaire de graphes pour transformer ce diagramme vers le réseau de pétri correspondant.

5.1 L'approche proposée

L'idée de base de la transformation consiste à remplacer chaque ***classe*** ou ***aspect*** d'un diagramme de classes orienté aspect par ***une place*** dans le réseau de pétri, chaque ***association*** d'un diagramme de classes orienté aspect par une ***transition*** dans le réseau de pétri et les ***cardinalités*** d'un diagramme de classes orienté aspect par les ***jetons*** dans le réseau de pétri.

La structure de cette transformation est résumée dans la figure 5.1.

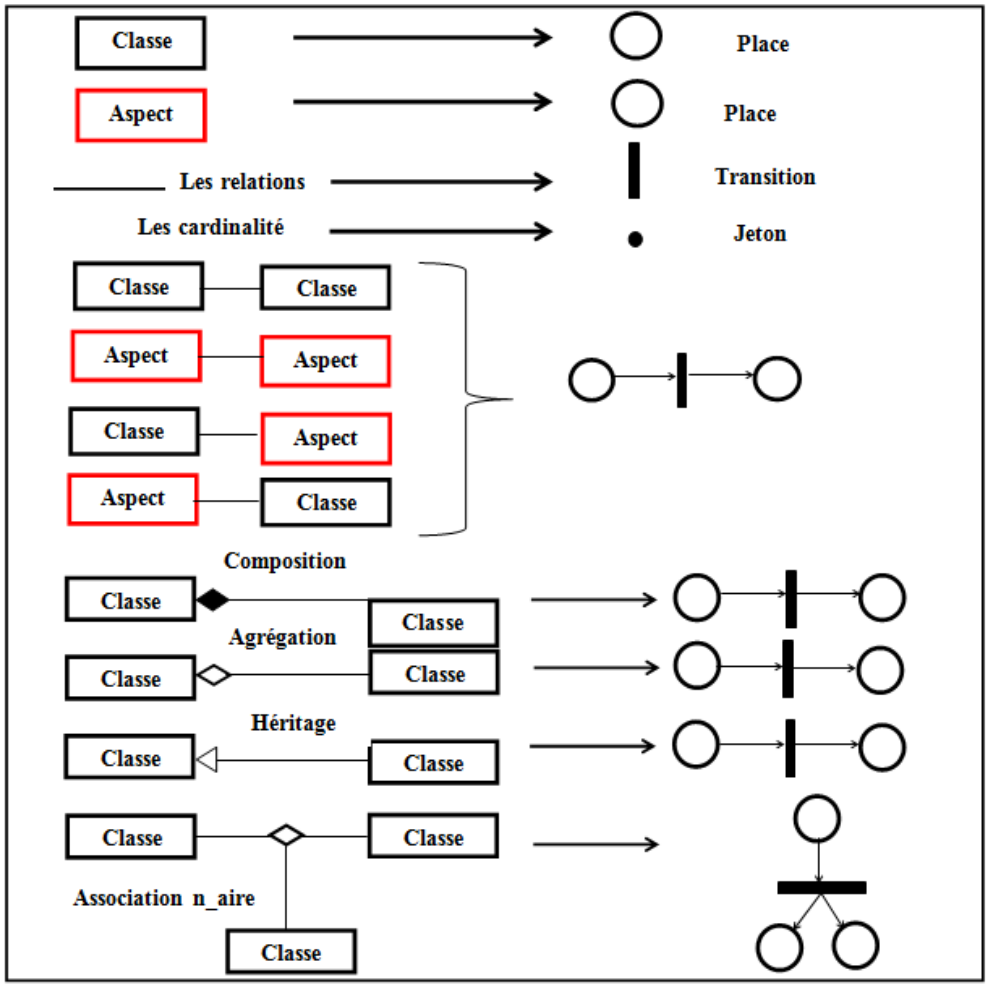

FIGURE 5.1 – Structure de transformation d'un diagramme de classes orienté aspect vers RdP.

Pour transformer un diagramme de classes orienté aspect vers RdP, nous avons proposé deux méta-modèles (DCOA et RdP) et une grammaire de graphe. Cette transformation est réalisée à l'aide de l'outil AToM^3.
L'AToM^{3}est utilisé pour la modélisation, le méta-modélisation et la transformation de modèles à l'aide des grammaires de graphes. D'ailleurs, il peut être étendu pour manipuler la simulation et la génération du code à partir des modèles. Les modèles ne sont pas simplement dessinés, mais ils sont construits par des règles présentées par une spécification de formalisme.

5.1.1 Méta-modélisation des diagrammes UML 2.0

Le concept de méta-modèle est une base formelle, qui permet de décrire plus précisément la syntaxe de la notation UML2.0 en utilisant UML2.0 elle-même.

5.1.1.1 Méta-modélisation des diagrammes de classes orientés aspect

Le méta-modèle du diagramme de classes orienté aspect composé de huit classes (DiagrammeDeClasseAspect, Classe_simple, Composition, Agrégations, Héritage (inheritance), Association, Association_naire, Aspect) et sept Associations (Association_1, Association_2, Association_3, Association_4 et Association_5, Association_6,Composer).

Dans la Figure (5.2), nous présentons l'outil généré pour la manipulation des diagrammes de classes.

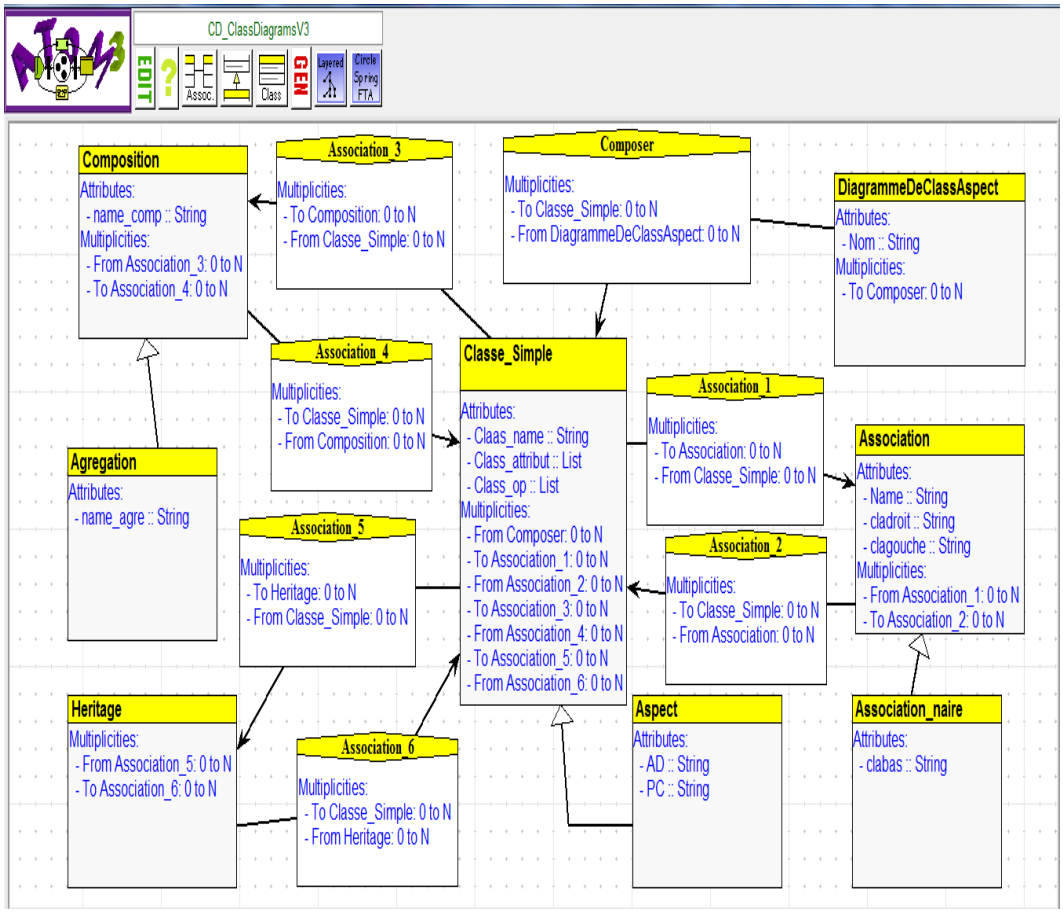

FIGURE 5.2 – Meta-modèle pour le diagramme de classes orienté aspect.

1. Les classes :

• **DiagrammeDeClasseAspect :** cette classe représente le diagramme de classe orienté aspect. Elle possède un attribut « Nom» de type String. Graphiquement, elle est représentée par un grand rectangle noir.

• **Classe_simple :** cette classe représente les classe_simple. Elle possède trois attributs «Class_name» de type string, «Class_attribut» et «Class_op» de type list. Graphiquement, elle est représentée par une classe noire,

• **Aspect :** est une classe qui hérite la classe «Classe_simple», elle hérite tous les attributs de la «Classe_simple» et plus deux attributs «AD» et «PC» de type string. Graphiquement, elle est représentée par une classe rouge.

• **Association :** cette classe représente les associations entre deux classes. Graphiquement elle est représentée par un lien, elle possède trois attributs «Name» «cladroit» et «Clagouche» de type string.
• **Association-naire :** est une classe qui hérite la classe «Association»,elle hérite tous les attributs de la classe «association» et plus l'attribut «clabase» de type string. Cette classe représente les associations entre plusieurs classes. Graphiquement elle est représentée par un lien .
• **Composition :** cette classe représente la composition entre deux classes. Elle possède l'attribut «name_comp» de type string. Graphiquement elle est représentée par une losange plaine.
• **Agrégation :** est une classe qui hérite la classe composition, cette classe représente l'agrégation entre deux classes. Elle possède l'attribut «Namagre» de type string. Graphiquement elle est représentée par un losange vide.
• **Héritage :** cette classe représente l'héritage entre deux classes. Graphiquement elle est représentée par une flache.

2. Les Associations :

• **Associations_1 :** relie la classe «Classe_simple» et la classe «Association», les cardinalités de cette association sont :
-To association : 0 to N.
-From classe_simple :0 to N .
• **Associations_2 :** relie la classe «Association» et la classe «Classe_simple», les cardinalités de cette association sont :
- To classe_simple : 0 to N.
- From association : 0 to N.
• **Associations_3 :** relie classe «Classe_simple» et la classe «Composition», les cardinalités de cette association sont :
- To composition : 0 to N .
- From classe_simple :0 to N .
• **Associations_4 :** relie la classe «Composition» et la classe «Classe_simple», les cardinalités de cette association sont :

- To classe_simple :0 to N .
- From composition : 0 to N .

• **Associations_5 :** relie la classe «Classe_simple» et le classe «Héritage», les cardinalités de cette association sont :
- To heritage : 0 to N .
- From classe_simple 0 to N.

• **Associations_6 :** relie le classe «Héritage» et la classe «Classe_simple», les cardinalités de cette association sont :
- To classe_simple 0 to N.
- From heritage : 0 to N .

• **Composer :** relie le classe «DiagrammeDeClasseAspect» et la classe «Classe_simple», les cardinalités de cette association sont :
-To classe_simple :0 to N.
-From DiagrammeDeClasseAspect : 0 to N .

La figure 5.3 représente l'outil généré pour la manipulation des diagrammes de classes orientes aspect.

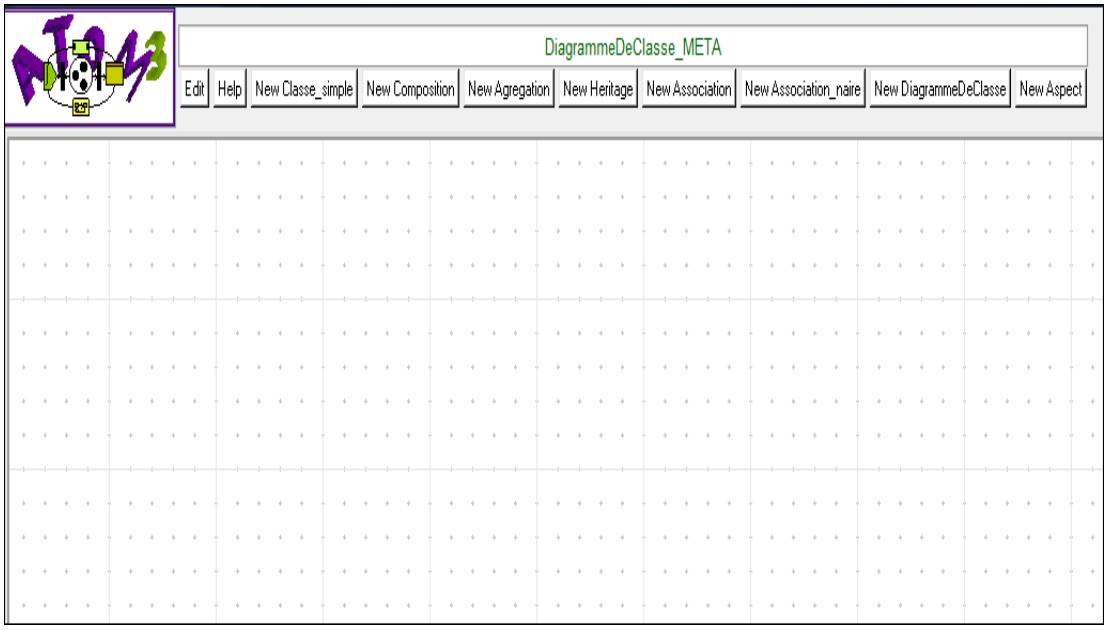

FIGURE 5.3 – l'outil généré pour les diagrammes de classes orientés aspect.

5.1.1.2 Méta-modèle du réseaux de pétri

Le méta-modèle des réseaux de pétri présenter dans la Figure 5.4 contient trois classes et trois associations.

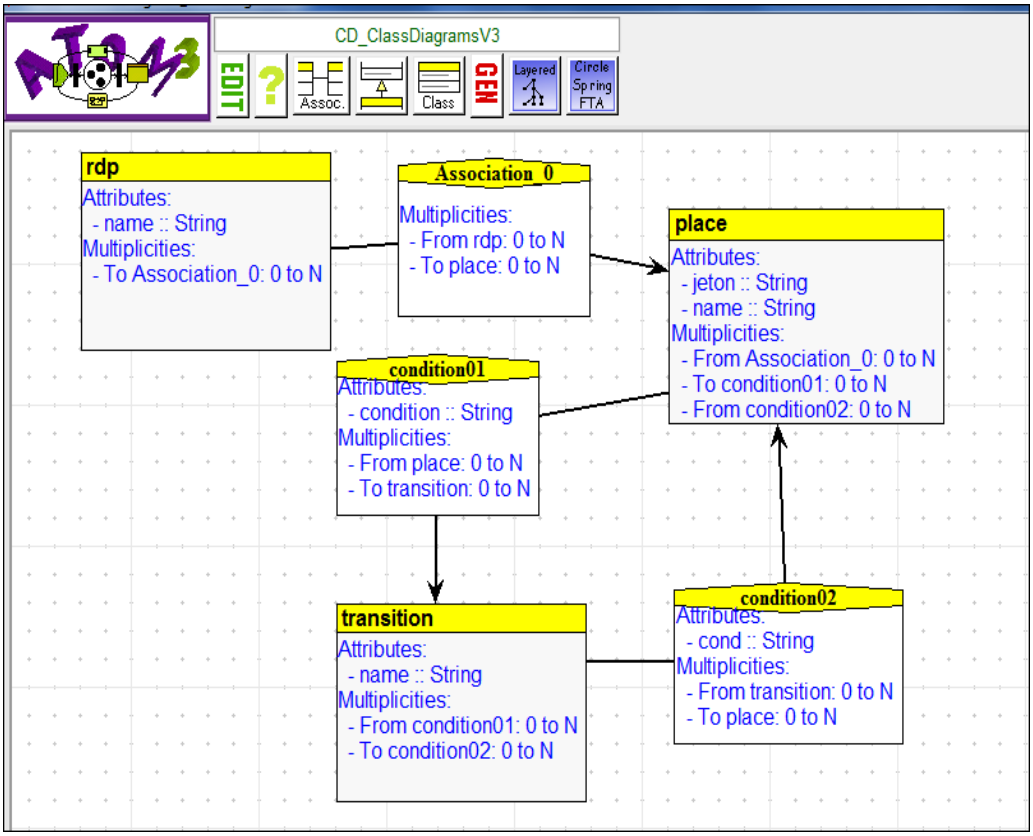

FIGURE 5.4 – Méta-modèle pour les RdPs .

1. Les classes :

• **RDP :** cette classe représente le réseau de pétri. Elle possède un attribut « name » de type String. Graphiquement, elle est représentée par un grand rectangle noir.

• **place :** cette classe représente les places. Graphiquement, elle est représentée par un cercle, elle possède deux attributs « name »et « jeton » de type String.

• **transition :** cette classe représente les transitions , elle possède l'attribut « name » de type String. Graphiquement elle est représentée par un petit rectangle noir.

2. Les associations :

• **Association_0 :** relie la classe « RDP » et la classe « place » avec les cardinalités :
- To place : 0 to N .
- From RPD : 0 to N.

• **Condition01 :** relie la classe « place » et la classe « transition ». Elle est visualisée par une flèche. Elle possède un attribut « condition» de type String avec les cardinalités :
- To transition 0 to N .
- From place : 0 to N.

• **Condition02 :** relie la classe « transition » et la classe « place ».Elle est visualisée par une flèche. Elle possède un attribut « cond » de type String avec les cardinalités :
- To place 0 to N.
- From transition : 0 to N.

La figure 5.5 représente l'outil généré pour la manipulation des réseaux de pétri

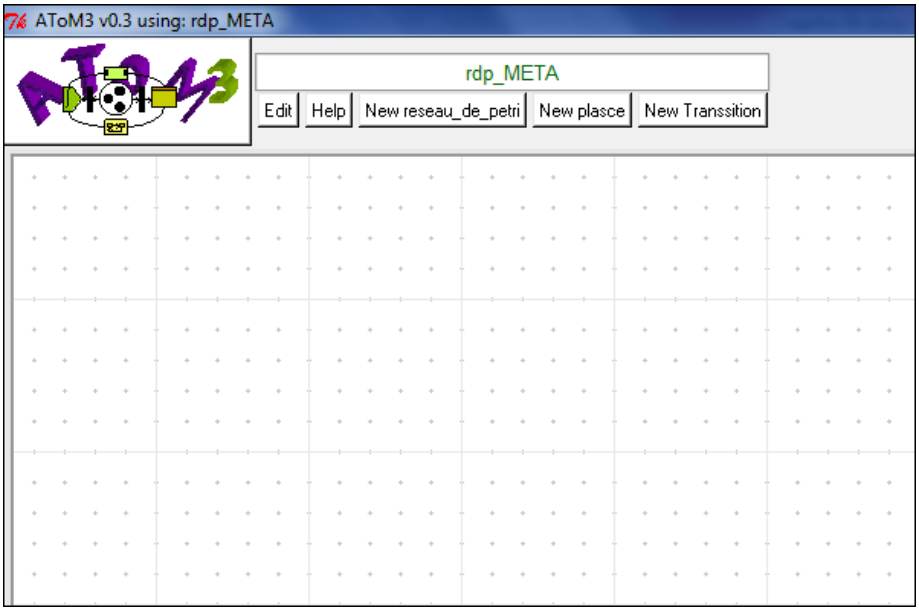

FIGURE 5.5 – Outil généré pour les RdPs .

5.1.2 Grammaire de graphes pour la transformation des diagrammes de classes orientés aspect vers les réseaux de pétri

Nous avons défini une grammaire de graphes composée de **39 règles** qui seront exécutées dans un ordre ascendant selon leurs priorités. Notre technique de transformation est basée sur la transformation de graphes, alors chaque règle est composée de deux parties : ***une partie gauche (LHS)*** et ***une partie droite (RHS)***. Ces deux parties peuvent contenir des éléments des diagrammes de classes orientés aspect, des RdPs ou bien les deux formalismes ou même temps (DCOA/RdP).

Notre grammaire de graphe est nommée « Aspectclass_RdP_GG », elle est divisée en **«3 catégories » :**

Catégorie 01 : cette catégorie pour la création des composants UML 2.0 vers RdP, cette classe englobe **3 règles**, chaque règle contient son numéro et son ordre.

La 1 ère règle : «diagramme de classes orienté aspect»

- **L'ordre d'exécution : 1**

Cette règle permet de transformer le diagramme de classes orienté aspect de la partie gauche LHS vers le réseau de pétri dans la partie droite RHS. Ce DCOA possède comme attribut «Nom» et le RdP possède comme attribut « name ».

La structure de cette règle est résumée dans la figure 5.6.

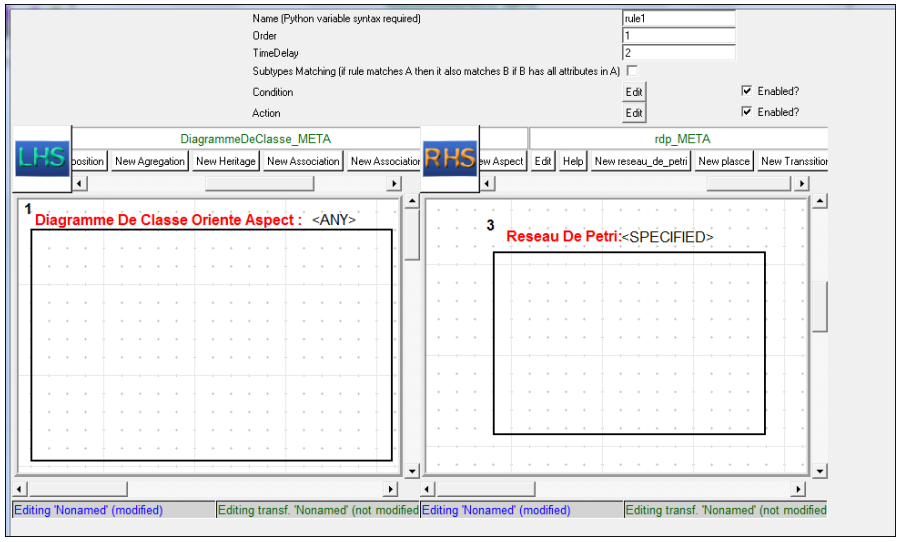

FIGURE 5.6 – La 1 ère règle «diagramme de classes oriente aspect» .

La 2 éme règle : «classe»

- **L'ordre d'exécution : 2**

Cette règle permet de transformer « une classe » de diagramme de classes orienté aspect de la partie gauche LHS en « une place » de réseau de pétri dans la partie droite RHS. Cette classe possède comme attribut «Class_name» et la place possède comme attribut « name ».

La structure de cette règle est résumée dans la figure 5.7.

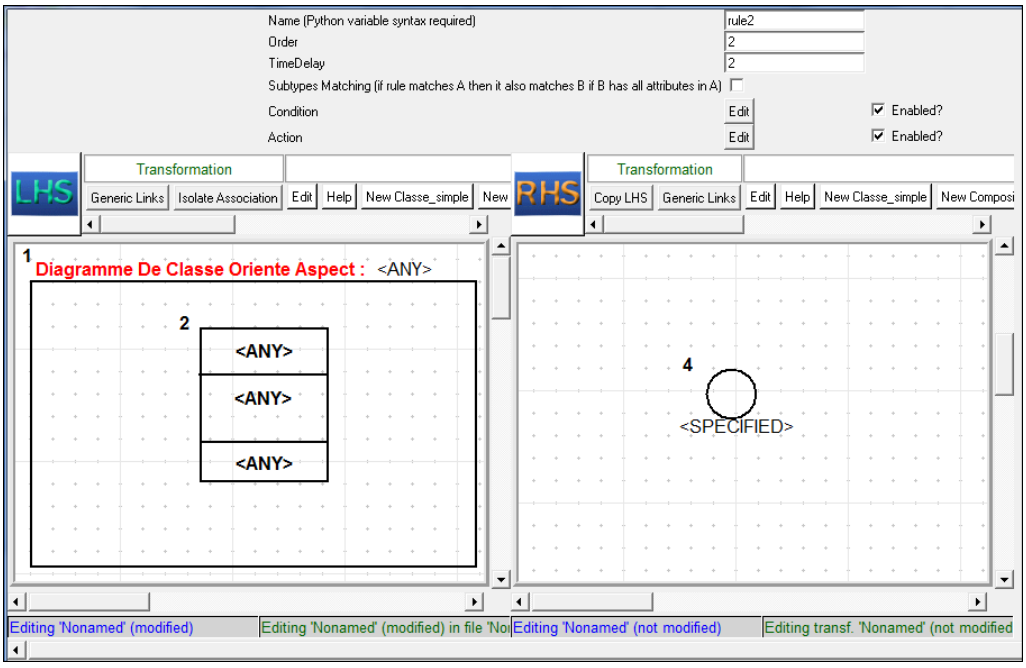

FIGURE 5.7 – La 2 éme règle «classe» .

La 3 éme règle : «aspect»

- **L'ordre d'exécution : 2**

Cette règle permet de transformer « un aspect » de diagramme de classes orienté aspect de la partie gauche LHS en « une place » de réseau de pétri dans la partie droite RHS. Cette classe possède comme attribut «Class_name» et la place possède comme attribut « name ».

La structure de cette règle est résumée dans la figure 5.8.

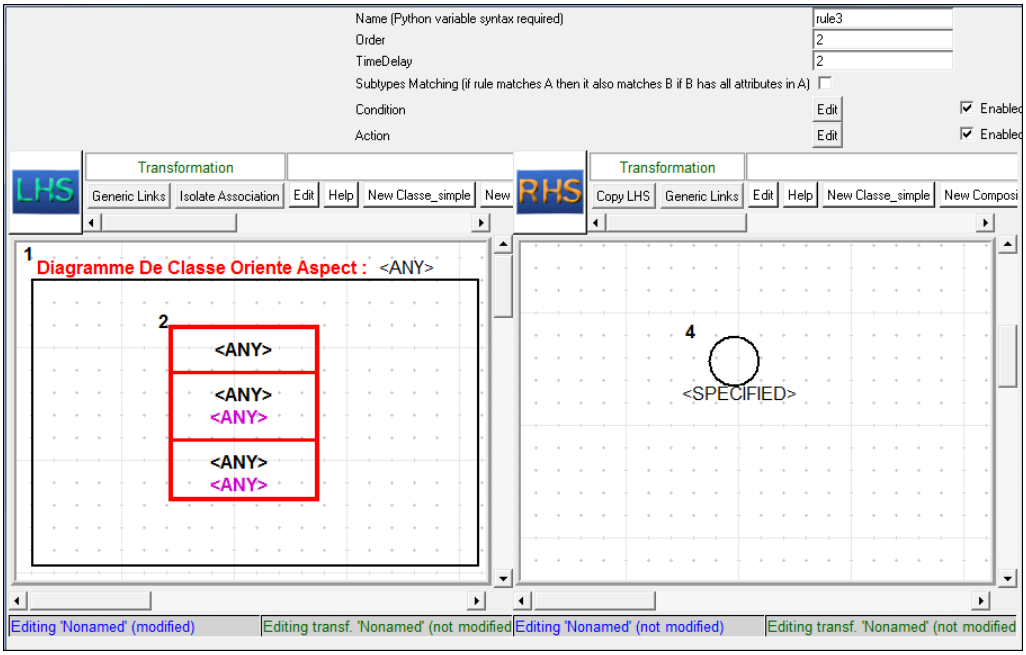

FIGURE 5.8 – La 3 éme règle «aspect».

Catégorie 02 : cette catégorie permet de transformer les Liaisons de diagramme de classes orienté aspect vers des transitions dans le réseau de pétri, cette catégorie contient **28 règles**, chaque règle contient son numéro et son ordre.

La 4 éme règle : « association entre deux classe»

- **L'ordre d'exécution : 3**

Cette règle permet de transformer « une association» qui relie deux classe d'un diagramme de classe orienté aspect de la partie gauche LHS en « une transition » relie deux place dans le réseau de pétri dans la partie droite RHS. Cette association possède comme attribut «Name» « clagouche» « cladroit » et la transition possède comme attribut « jeton » « name ».

La structure de cette règle est résumée dans la figure 5.9 .

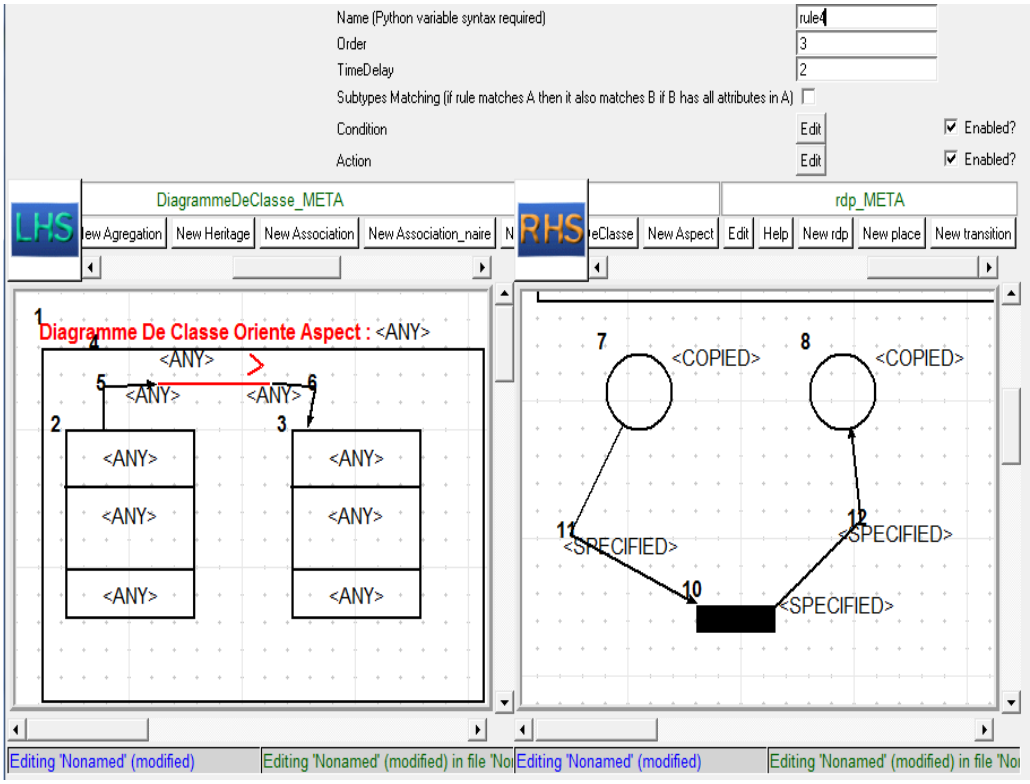

FIGURE 5.9 – La 4 éme règle association entre deux classes.

La 5 éme règle : « association entre deux aspect»

- **L'ordre d'exécution : 3**

Cette règle permet de transformer « une association» qui relie deux aspect d'un diagramme de classes orienté aspect de la partie gauche LHS en « une transition » relie deux place dans le réseau de pétri dans la partie droite RHS. Cette association possède comme attribut «Name» «clagouche» « cladroit » et la transition possède comme attribut « jeton » « name ».

La structure de cette règle est résumée dans la figure 5.10.

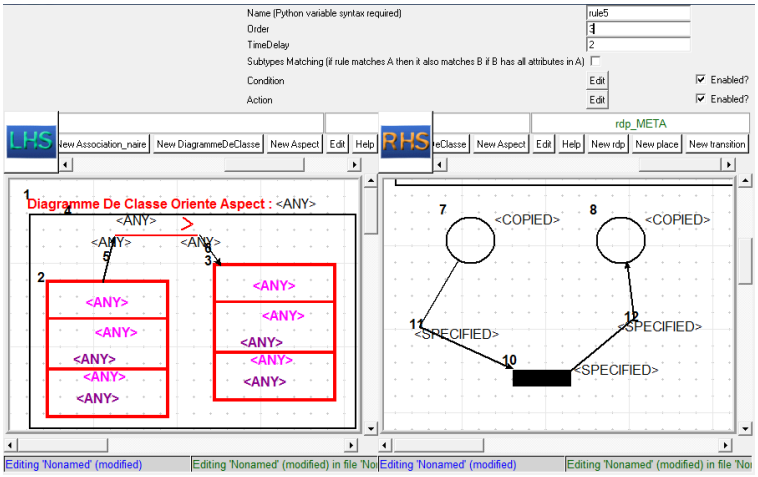

FIGURE 5.10 – La 5 éme règle association entre deux aspects.

La 6 éme règle : « association entre classe et aspect»

- **L'ordre d'exécution : 3**

Cette règle permet de transformer «une association» qui relie classe et aspect d'un diagramme de classe orienté aspect de la partie gauche LHS en « une transition » relie deux place dans le réseau de pétri dans la partie droite RHS. Cette association possède comme attribut «Name» «clagouche» « cladroit » et la transition possède comme attribut « jeton » « name ».

La structure de cette règle est résumée dans la figure 5.11 .

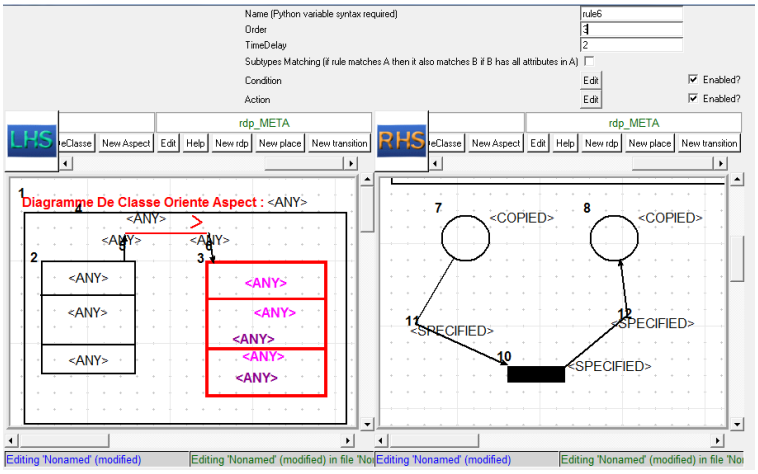

FIGURE 5.11 – La 6 éme règle association entre classe et aspect.

La 8 éme règle : « héritage entre les classes»

- **L'ordre d'exécution : 4**

Cette règle permet de transformer « l'héritage» qui relie les classes d'un diagramme de classes orienté aspect de la partie gauche LHS en « une transition » relie deux place dans le réseau de pétri dans la partie droite RHS.

La structure de cette règle est résumée dans la figure 5.12.

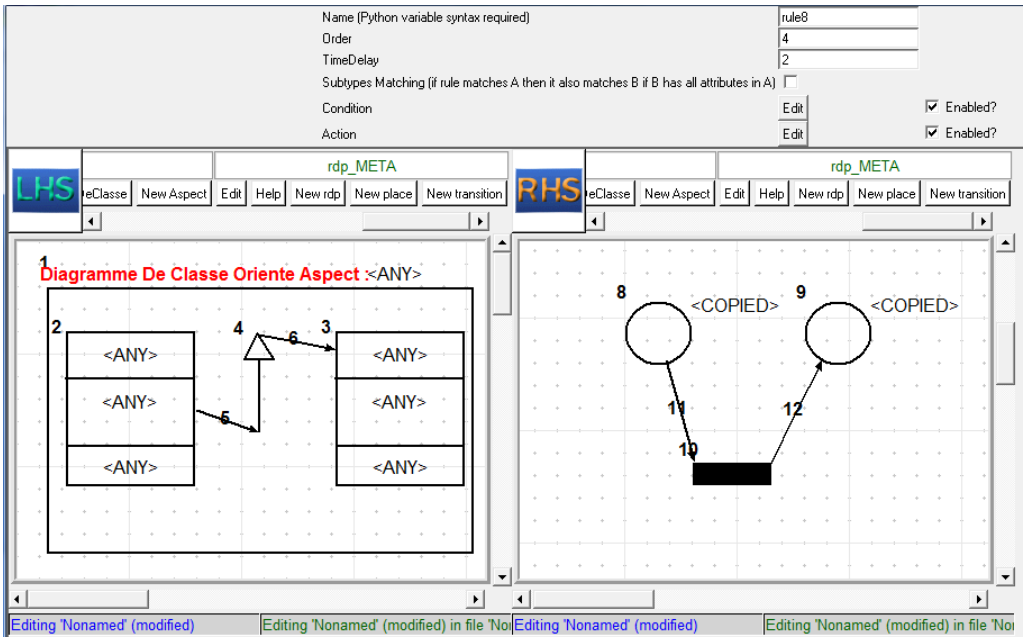

FIGURE 5.12 – La 8 éme règle héritage entre les classes

La 12 éme règle : « composition entre les classes»

- **L'ordre d'exécution : 5**

Cette règle permet de transformer «la composition» qui relie les classes d'un diagramme de classes orienté aspect de la partie gauche LHS en « une transition » relie deux place dans le réseau de pétri dans la partie droite RHS. Cette composition possède comme attribut «Name_comp» et la transition possède comme attribut « jeton » « name ».

La structure de cette règle est résumée dans la figure 5.13.

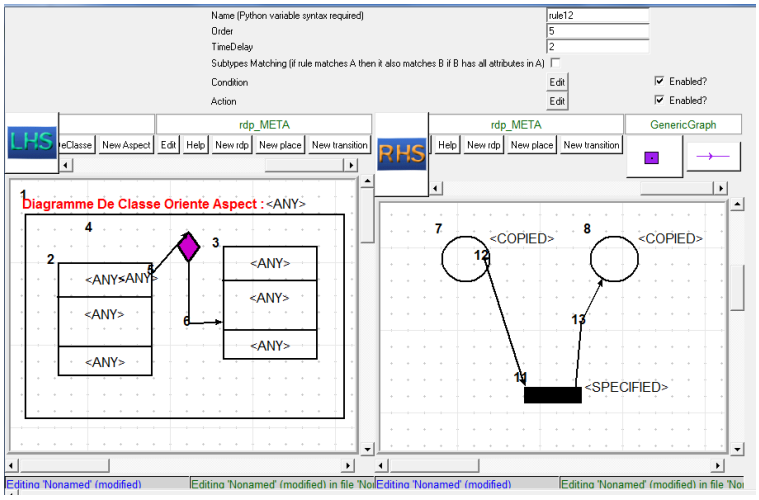

FIGURE 5.13 – La 12 éme règle composition entre les classes .

La 17 éme règle : « agrégation entre les aspects»

•L'ordre d'exécution : 5

Cette règle permet de transformer « l'agrégation » qui relie les aspects d'un diagramme de classes orienté aspect de la partie gauche LHS en « une transition » relie deux place dans le réseau de pétri dans la partie droite RHS. Cette agrégation possède comme attribut «Namagre» et la transition possède comme attribut « jeton » « name ».

La structure de cette règle est résumée dans la figure 5.14.

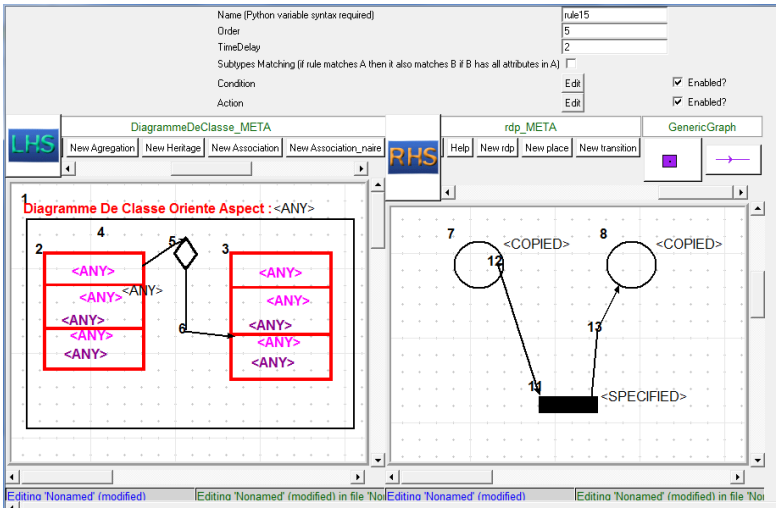

FIGURE 5.14 – La 17 éme règle agrégation entre les aspects .

La 21 éme règle : « Association n_aire entre trois aspect»

- **L'ordre d'exécution : 6**

Cette règle permet de transformer «association n_aire » qui relie les aspects d'un diagramme de classes orienté aspect de la partie gauche LHS en « une transition » relie deux place dans le réseau de pétri dans la partie droite RHS. Cette association n_aire possède comme attribut «Name» «clagouche» «cladroit» «clabas» et la transition possède comme attribut « jeton » «name ».

La structure de cette règle est résumée dans la figure 5.15

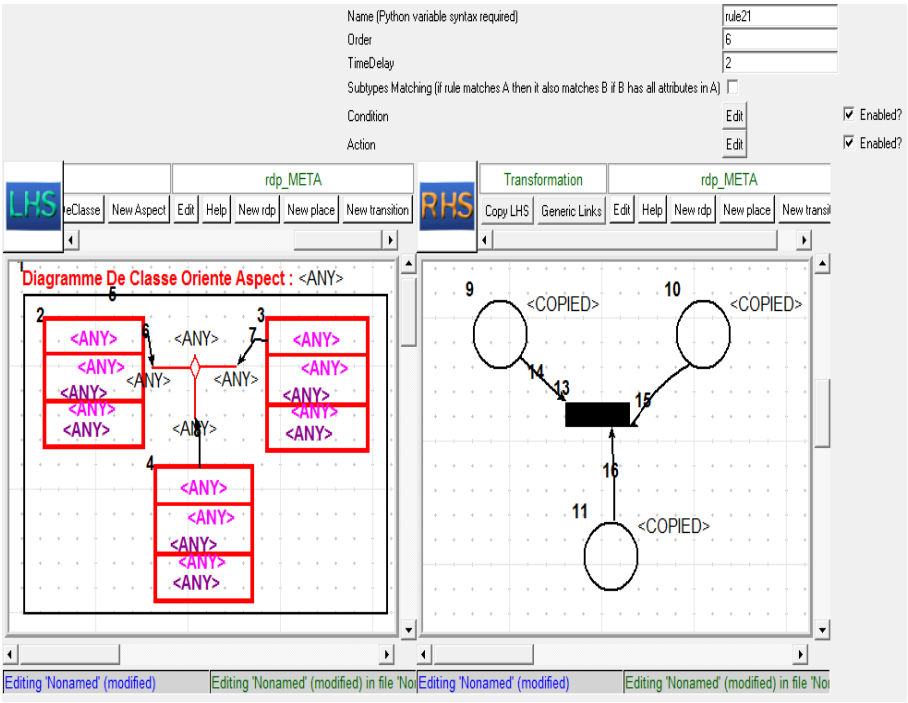

FIGURE 5.15 – La 21 éme règle association n_aire .

La 27 éme règle : « Association n-aire entre la classe et l'aspect»

- **L'ordre d'exécution : 6**

Cette règle permet de transformer «association n-aire » qui relie les aspects et les classe d'un diagramme de classes orienté aspect de la partie gauche LHS en « une transition » relie deux place dans le réseau de pétri dans la partie droite RHS. Cette association n-aire possède comme attribut «Name» «clagouche» «cladroit» «clabas» et la transition possède comme attribut « jeton » «name ».

La structure de cette règle est résumée dans la figure 5.16.

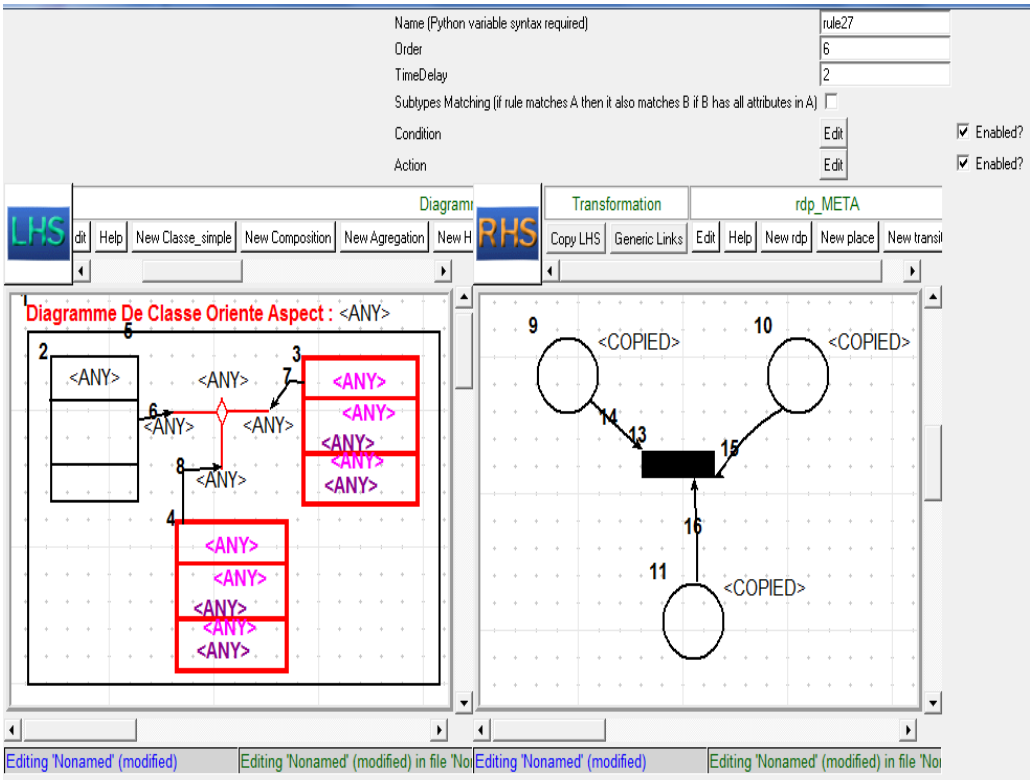

FIGURE 5.16 – La 27 éme règle association n-aire .

La 29 éme règle : « agrégation réflexive entre la classe et l'aspect»

- **L'ordre d'exécution : 7**

Cette règle permet de transformer « l'agrégation réflexive » qui relie aspects lui-même d'un diagramme de classes orienté aspect de la partie gauche LHS en « une transition » relie place lui-même dans le réseau de pétri dans la partie droite RHS. Cette agrégation réflexive possède comme attribut «Namagre» et la transition possède comme attribut « jeton » «name ».

La structure de cette règle est résumée dans la figure 5.17.

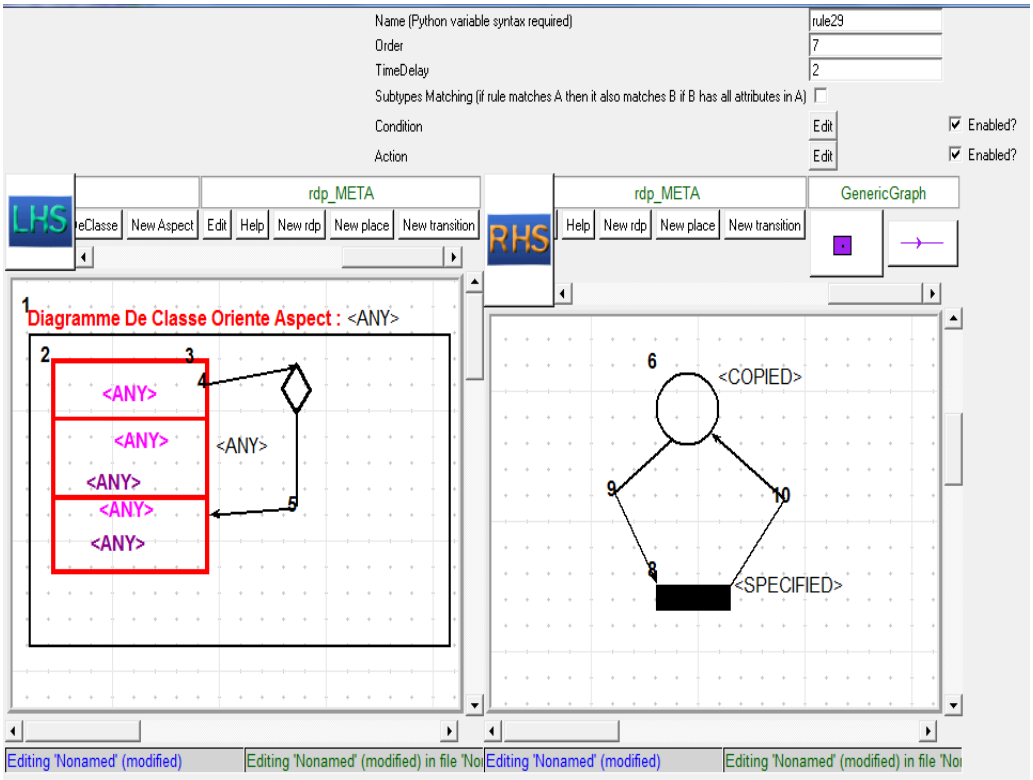

FIGURE 5.17 – La 29 éme règle agrégation réflexive.

Catégorie 03 :cette dernière catégorie permet de supprimer les composants du diagramme de classes orienté aspect qui restent dans le « ***Input*** » de l'outil AToM^3 lorsque la transformation est terminée, bien sûr pour garder seulement le « ***Output*** » (notre résultat est le RdP), cette catégorie contient 8 règles, et chaque règle contient son numéro et son ordre

La 32 éme règle : «supprimer les classes »

- **L'ordre d'exécution : 8**

Cette règle permet de supprimer « les associations » qui existent dans le diagramme de classes orienté aspect.

La structure de cette règle est résumée dans la figure 5.18.

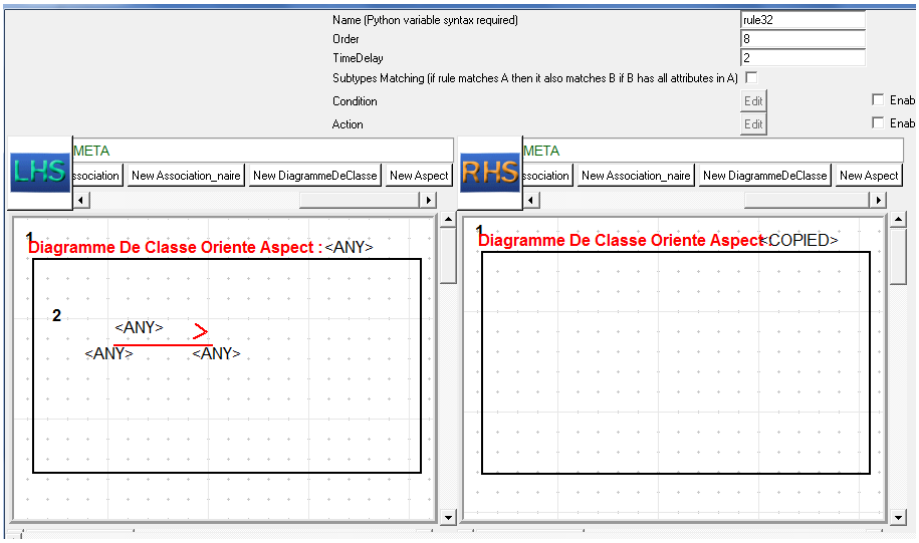

FIGURE 5.18 – La 32 éme règle supprimer les associations .

La 33 éme règle : «supprimer les composition »

- **L'ordre d'exécution : 8**

Cette règle permet de supprimer « les composition » qui existent dans le diagramme de classes orienté aspect.

La structure de cette règle est résumée dans la figure 5.19.

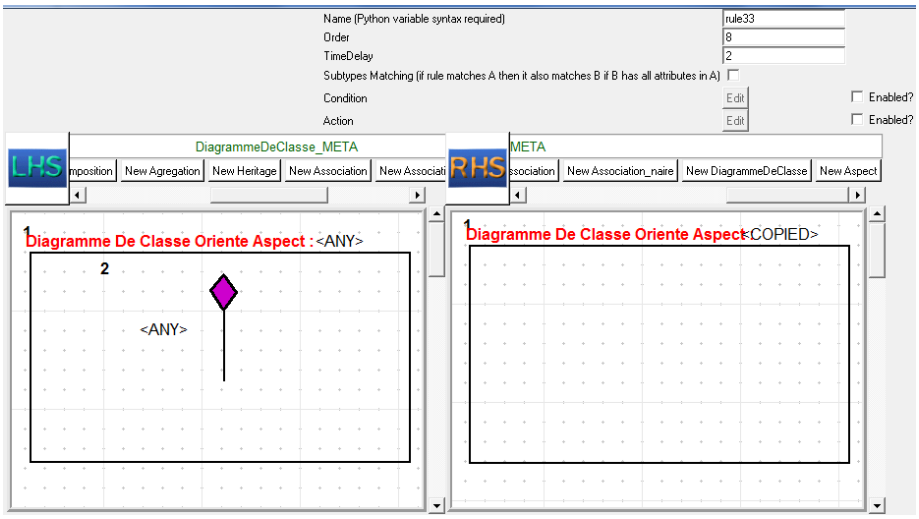

FIGURE 5.19 – La 33 éme règle supprimer les compositions .

La 37 éme règle : «supprimer les classes »

- **L'ordre d'exécution : 9**

Cette règle permet de supprimer «les classes» qui existe dans le diagramme de classes orienté aspect.

La structure de cette règle est résumée dans la figure 5.20.

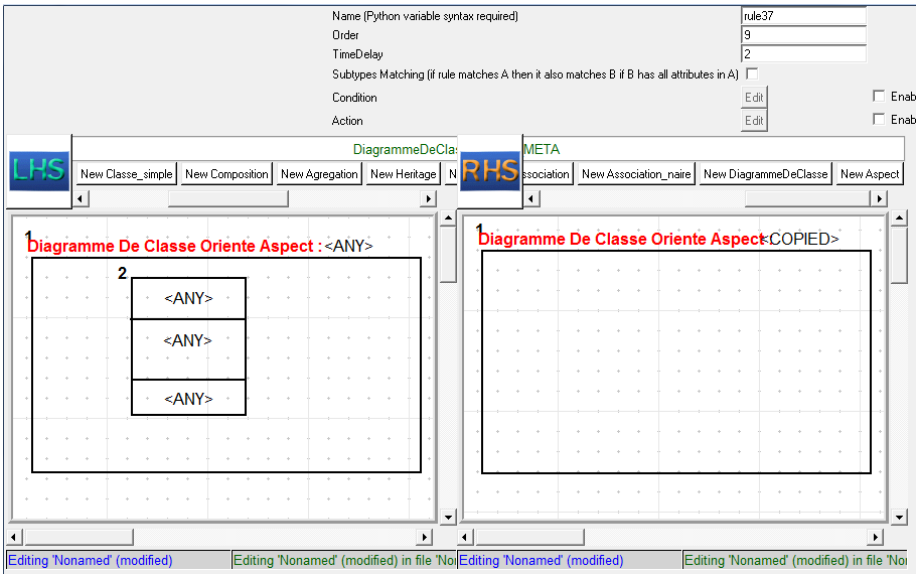

FIGURE 5.20 – La 37 éme règle supprimer les classes.

La 38 éme règle : «supprimer les aspect »

- **L'ordre d'exécution : 9**

Cette règle permet de supprimer «les aspect» qui existe dans le diagramme de classes orienté aspect.

La structure de cette règle est résumée dans la figure 5.21.

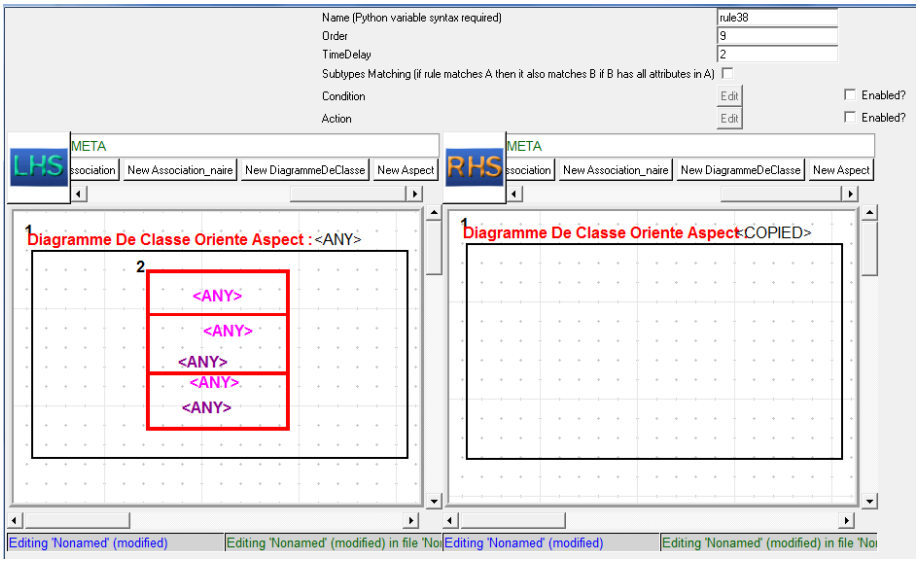

FIGURE 5.21 – La 38 éme règle supprimer les aspects.

La 39 éme règle : «supprimer le diagramme de classes orienté aspect»

- **L'ordre d'exécution : 10**

Cette règle permet de supprimer le diagramme de classes orienté aspect.

La structure de cette règle est résumée dans la figure 5.22.

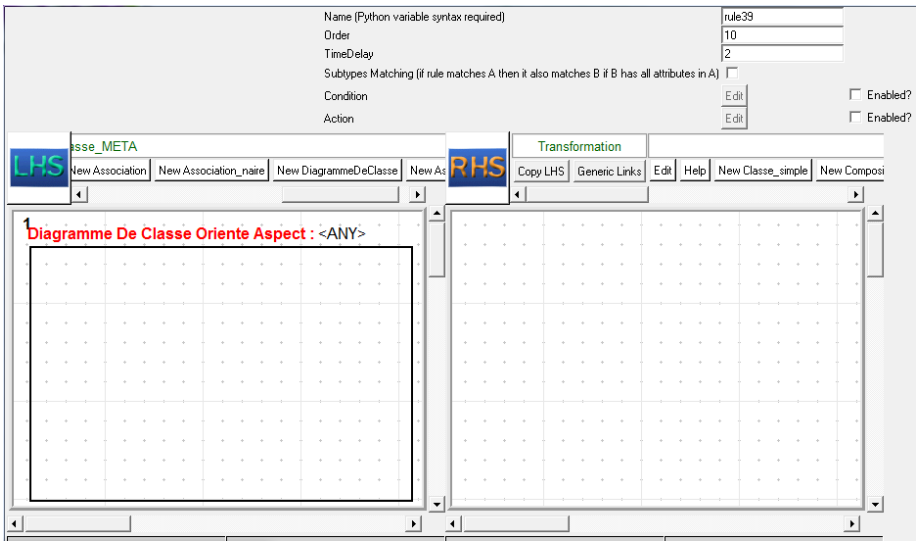

FIGURE 5.22 – La 39 éme règle supprimer diagramme de classe oriente aspect .

5.2 Études de cas

Pour illustrer notre approche de transformation présentée dans ce chapitre et aussi pour pouvoir concrétiser l'utilité de la grammaire définie, nous reprenons des étude de cas

Nous avons appliqué notre approche sur deux études de cas

- Le système de «la gestion des services bancaires».
- Le système de «la gestion des stages au niveau du centre universitaire de mila».

5.2.1 Présentation des études de cas

A -L'explication de système la gestion des services bancaires Les besoins d'utilisation de ce système :

- Consulter compte ; Demande l'affichage de mouvement ; Commander chaque ;
- Effectuer virement bancaire ; Consulter annuaire agence. . . . . etc.
- Le client possède une ou plisseur comptes dans la banque.
- La banque compose des agences.
- Les comptes appartenir dans l'agence.
- Il existe deux types d'administrateur :
  - Administrateur banque pour gère les banques.
  - Super administrateur pour gère les agences.

1) Diagramme de classes orienté aspect :

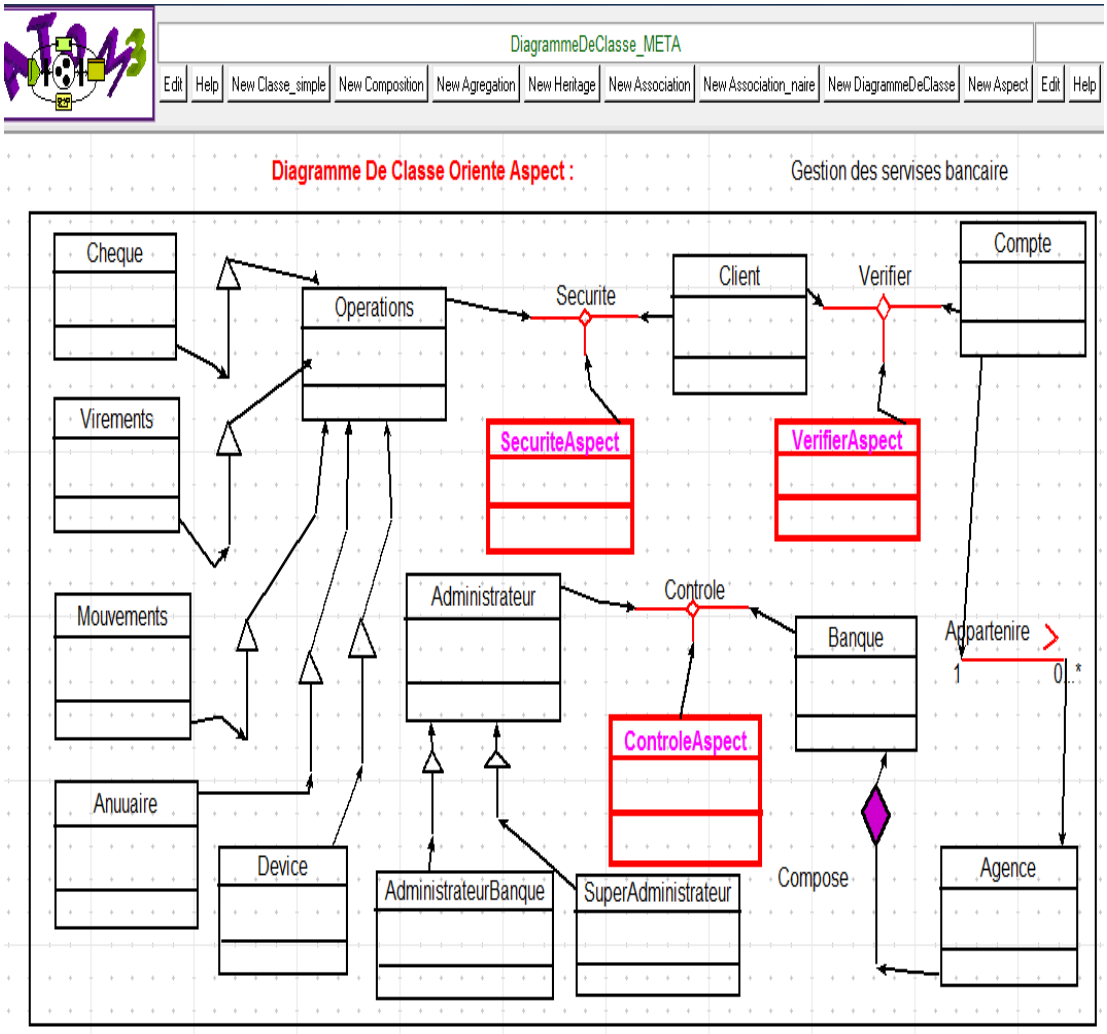

FIGURE 5.23 – Diagramme de classes orienté aspect «système de gestion des services bancaire».

2) Réseaux de pétri :

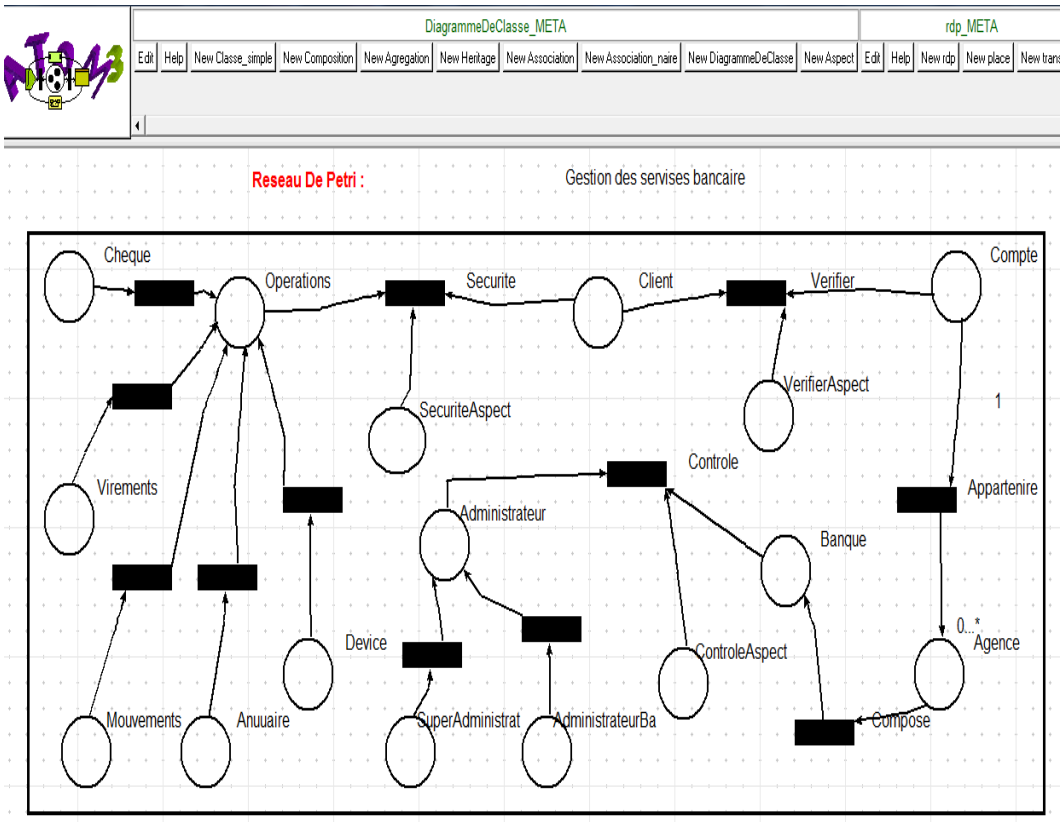

FIGURE 5.24 – Réseaux de pétri pour «la gestion de systéme des services bancaire» .

B - Explication du système de «la gestion des stages au niveau du centre universitaire de mila»

- Demandeur de stage : la personne qui utilise l'application pour effectuer en ligne, les opérations de consultation des annonces, présentation et suivie des candidatures grâce à son espace personnel. La personne possède un dossier pour le stage.

– Conseil scientifique : cet acteur est chargé de réaliser les taches suivantes : établissement des critères scientifiques pour la sélection des candidatures, étude des candidatures afin de les noter, classement des candidatures, et étude des recours.

– Administrateur : c'est l'acteur responsable de la gestion administrative des stages. Nous pouvons distinguer deux types d'administrateurs, un administrateur central qui gère les stages au niveau du centre universitaire, et des administrateurs locaux,

dont chacun s'occupe de la gestion des stages au niveau d'un institut.

-Le demandeur de stage soit un doctorant ou enseignent.

-L'utilisateur de système :demandeur de stage,administrateur et Conseil scientifique.

1) Diagramme de classes orienté aspect :

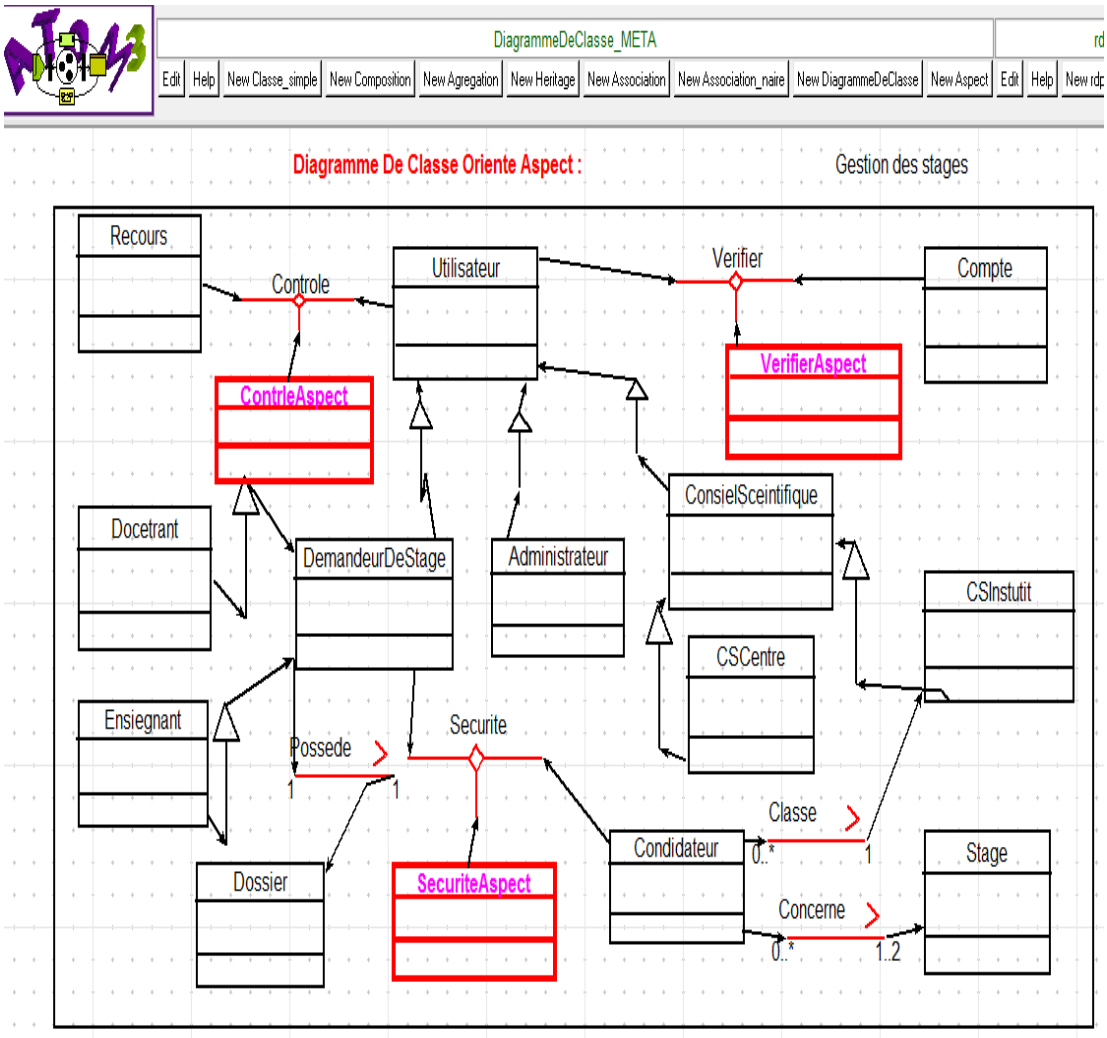

FIGURE 5.25 – Diagramme de classes orienté aspect «système de gestion des stages» .

2) Réseaux de pétri :

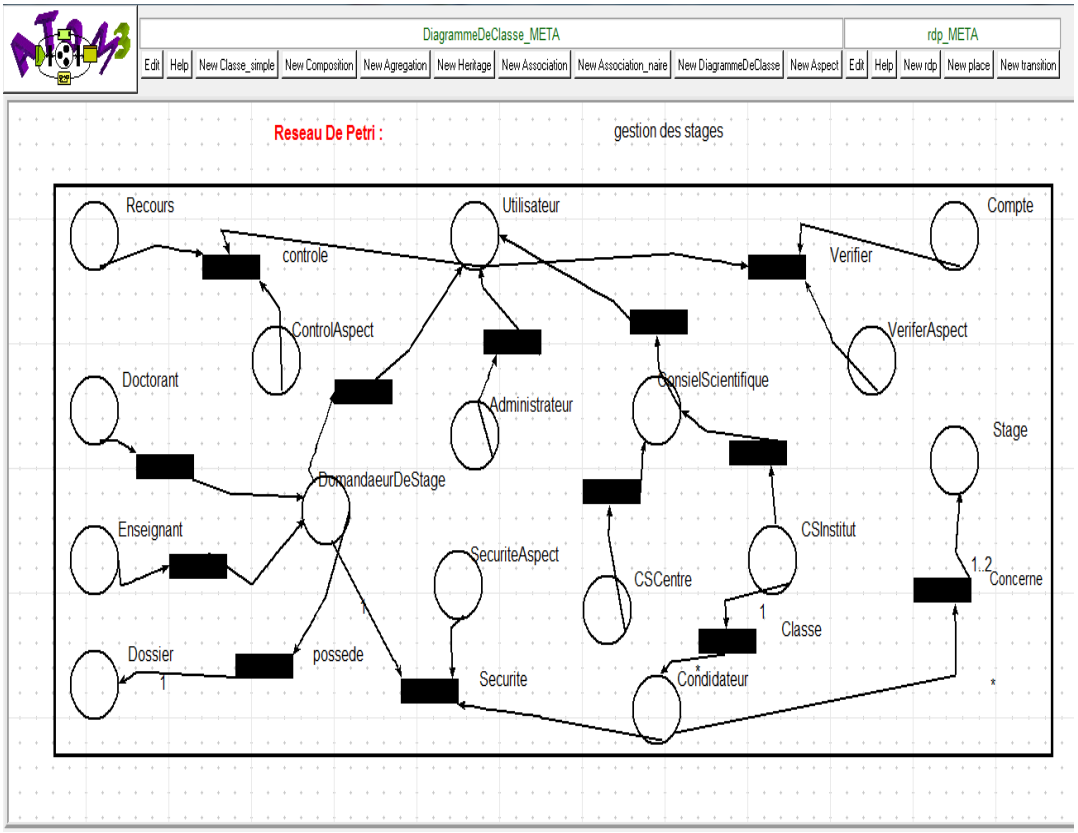

FIGURE 5.26 – Réseaux de pétri pour «la gestion des stages» .

Conclusion

Dans ce chapitre, nous avons proposé une approche automatique basée sur la transformation de graphes pour générer un réseau de pétri à partir d'un diagramme de classes orienté aspect. Cette approche est basée sur la méta-modélisation.

Donc, nous allons décrire et concevoir notre approche en définissant le méta-modèle du diagramme de classes orienté aspect, et aussi le méta-modèle du réseau de pétri ensuite nous avons présenté une grammaire de graphes « des différentes règles » permettant la réalisation de la transformation. Cette transformation est effectuée à l'aide de l'outil de modélisation et de méta-modélisation, multi-formalisme « AToM^3 ». Et en fin, nous allons illustrer un exemple de transformation bien discuté.

Conclusion et Perspectives

Le travail présenté dans ce mémoire s'inscrit dans le domaine de l'ingénierie dirigée par les modèles. Il se base essentiellement sur l'utilisation combinée de méta-modélisation et de transformation de modèle. Plus précisément, la transformation de graphes est utilisée comme outil de transformation de modèles. Le résultat de notre travail est une approche pour transformer les diagrammes de classes orientés aspect vers les réseaux de pétri. L'approche proposée est basée sur la transformation de graphes.

On a choisi l'approche de transformation de graphes parmi toutes les approches de transformation de modèle, car elle est très utilisée dans la littérature et des nombreux outils sont développés pour rendre ces transformations automatiques et lisibles ; parmi ces outils on a choisi l'outil ATom^3 comme un bon outil de modélisation, multi formalismes, méta modélisation et de transformation de graphes. La transformation de graphes avec ATom^3 est une opération simple et efficace grâce à leur sémantique. Il forme une base pour l'intégration d'autres nouveaux outils.

Pour atteindre nos objectifs, nous avons eu l'opportunité de comprendre et d'utiliser plusieurs concepts (RdP, MDA, IDM, grammaire de graphes… etc.).
Notre travail est réalisé dans deux étapes :

- La première étape consiste à proposer deux méta-modèles. Un pour les diagrammes de classes orientés aspect, et l'autre pour les réseaux de pétri, afin de générer

un outil visuel permettant la manipulation de ces modèles.

- La deuxième étape sert à proposer une grammaire de graphes permettant la génération automatique de modèle cible.

Comme perspectives, nous comptons dans un premier lieu, de continuer la transformation des classes, des aspects et des relations entre les classes et les aspects, afin d'arriver à une transformation complète des diagrammes de classes orientés aspect. Ensuite, et dans le but de valider notre approche, nous devrons vérifier le modèle obtenus qui est le réseau de pétri.

Le sujet une fois traité, ne s'est pas refermé, il peut être amélioré et perfectionné, dans de différentes vues possibles.

Finalement, afin d'arriver à développer une approche totalement automatique, incluant tous les diagrammes UML 2.0, nous proposerons de continuer la transformation des autres diagrammes UML 2.0 (diagramme de séquence orienté aspect, diagramme de cas d'utilisation orienté aspect, diagramme..., orientée aspect, etc. ...) vers les réseaux de pétri en utilisant toujours la transformation de graphes et l'outil AToM^3 .

Références Bibliographiques

[Abdellah et Tiga , 2014] Abdellah ,Houssam-eddine.et Tigha, Wahid .(2014).*Une Approche de Transformation des Diagrammes De Communication Orientés Aspects vers les Réseaux de Pétri Colorés Basée sur la Transformation de Graphes*. Mémoire de master. Université de Constantine2 , Constantine.pp :14-15.

[Alonso, 2009] Alonso,Mathilde. (2009).*Conception de l'interaction dans un eiah pour la modélisation orientée object*. Thèse de doctorat.Université du Maine.pp :20-25.

[Amroune, 2014] Amroune , Moammed.(2014).*Vers une Approche Orientée Aspect d'Ingénierie des Besoins dans les Organisations multi Entreprises*. Thèse de doctorat. Université de Toulouse, Toulouse.pp :15-20.

[Aouag, 2014] Aouag,Mouna. (2014).*Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes*. Thèse de doctorat, Université de Mentouri, Constantine.pp :8-43.

[Aouag et al, 2013] Aouag, Mouna.et Allaoua ,Chaoui.(2013). *From UML Class Diagrams to Aspect-Oriented Class Diagrams Using Graph Transformation*. International Conference on Control,Decision and Information Technologies. CoDIT'13 978-1-4673-5549-0/13/$31.00 ©2013 IEEE.

[Audibert, 2008] Audibert, Laurent. (2008).*UML 2*.Institut Universitaire de Technologie de Villetaneuse, Villetaneuse. pp 45-62.

[Bahri, 2011] Bahri, Mohamed Rédha .(2011).*Une approche intégrée Mobile-UML/Réseaux de Petri pour l'Analyse des systèmes distribués à base d'agents mobiles*. Thèse de doctorat, Université de Mentouri, Constantine.pp : 55-71.

[Belounnar, 2011] Belounnar ,*Saliha.Une approche formelle pour l'adaptabilité d'un agent*.Mémoire de Magister . Université Mohamed Khider , Biskra.pp :31.

[Benmoussa et Houidi, 2015] Benmoussa, Fatima Zohra.et Houidi,Farida. *Une approche basé sur la transformation du graphe sous ATom3 pour l'intégration des deux outils de réseau de petri*.Mémoire de master. Université Echahide Hamma Lakhdar,EL-OUED.pp :4-6.

[Bouarioua, 2013] Bouarioua ,Mouna.(2013). *Une approche basée transformation de graphes pour la génération de modèles de réseaux de Petri analysables à partir de diagrammes UML.* Thèse de doctorat .Université de Mentouri, Constantine.pp :71-81.

[Boubendir, 2011] Boubendir, Amel. (2011). *Un cadre générique pour la détection et la résoulution des intéraction entre les aspects*. Thèse de doctorat, Université de Mentouri, Constantine.pp :35-36.

[Blanc, 2005] Blanc, Xavier. (2005).*Mda en action : Ingénierie logicielle guidée par les modèles*. Paris : Eyrolles.pp :3-5.

[Dehimi, 2014] Dehimi,Nardjess Tissilia.(2014). *Un Cadre Formel pour La Modélisation et L'analyse Des Agents Mobiles*. Thèse de Doctorat. Université de Mentouri ,Constantine. pp :50-70.

[Elmansouri, 2009] Elmansouri, Raida. (2009). *Modélisation et Vérification des processus métiers dans les entreprises virtuelles : Une approche basée sur la transformation de graphes*. Thèse de doctorat, Université de Mentouri,Constantine.pp :55-82.

[Gabay et Gabay, 2008] Gabay, Joseph.et Gabay, David. (2008).*UML 2 analyses et conception*. Paris :Dunod. pp : 32-33.

[Hachichi, 2013] Hachichi, Hiba.(2013).*Test formel des systèmes temps réel : Approche de transformation de graphes*. Thèse de doctorat. Université de Mentouri, Constantine.pp :27-28.

[Haiouni, 2010] Haiouni,Houda.(2010). *Approche mixte de modélisation par Réseaux de Petri et SMA*. Mémoire de magister. Université de Mentouri, Constantine.pp :51-52

[Hettab, 2009] Hettab, Abdelkamel. (2009).*De M-UML vers les réseaux de Petri « Nested Nets » : Une approche basée transformation de graphes*.Thèse de doctorat.Université de Mentouri, Constantine.pp :59-82.

[Kerkouche, 2011] Kerkouche, Elhillali. (2011).*Modélisation Multi-Paradigme : Une Approche Basée sur la Transformation de Graphes*. Thèse de doctorat. Université de Mentouri, Constantine.pp :7-57.

[Khalfaoui, 2014] Khalfaoui,Khaled.(2014).*Une Approche de Spécification des Changements des Besoins Basée Transformation de Graphes*. Thèse de doctorat.Université Mohamed Khider,Biskra.pp :31.

[Kholadi, 2009] Kholadi, Mohamed Naoufel.(2009).*Une Approche de transformation de la notation BPMN vers BPEL basée sur la transformation de graphe*. Thèse de doctorat, Université de Mentouri, Constantine.pp :44-51.

[Klein, 2006] Klein , Jacques.(2006).*Aspects comportementaux et tissage* .Thèse de doctorat.Universitéde rennes 1 ,Rennes. pp :19-29.

[Laouar, 2013] Laouar,Adnane. (2013).*Une approche de transformation de diagrammes d'activités orienté aspects vers les réseaux de pétri colorés basée sur la transformation de graphes*. Mémoire de master.Université de Constantine2,

Constantine. pp :5-35.

[Menasra et cherafa, 2013]Menasra ,Mohamed .et Cherafa ,Said.(2013).*Une approche de transformation des diagrammes de classes UML vers la méthode B sur la transformation de graphes*.mémoire de master. Université de Mentouri, Constantine.pp :4-10.

[Muller, 2006] Muller, Pierre-Alain.(2006).*De la modélisation objet des logiciels à la metamodélisation des langages informatiques*. Thèse de doctorat.Universitéde rennes 1 ,Rennes. pp :20.

[Otmane Rachedi, 2015] Otmane rachedi ,Soumeya.(2015). *Apports des Approches de Séparation Avancée des Préoccupations : Une Etude Comparative Fondée sur les Modèles de Conception*. Thèse de doctorat. Université de Badji mokhtar ,annaba. pp :16-20.

[Rahab, 2014] Rahab ,Hichem.(2014).*Une approche à base d'agents adaptatifs pour la résolution des systèmes complexes*. Mémoire de magister .Université de Hadj Lakhdar,Batna.pp :47-48.

[Roques , 2006] Roques, Pscale. (2006).*UML 2 par la pratique*. France : Eyrolles. pp : 4-9.

[Roques , 2008] Roques ,Pscale.(2008).*UML 2 par la pratique*. France : Eyrolles.pp : 76-80.

[Roques et Vallée, 2007] Roques, Pscale .et Vallée, Franck.(2007).*UML 2 en action de l'analyse des besoins à la conception*. France : Eyrolles. pp : 25-29.

[Saggadi, 2007] Saggadi,Samira.(2007).*Optimisation des temps d'attente des système flexibles de production basée sur les réseaux de pétri* . Mémoire de magis-

ter. Université M'hamed bougara,Boumerdése .pp :24-25.

[Wautelet et al, 2013]Wautelet, Yves .et Louvigny, Laurent .et Manuel, Kolp. (2013). *Modélisation oriente-objet d'aspect opérationnel de bases de donnes siderurgique*. IAG – ISYS (Unité de Systèmes d'Information), Université Catholique de Louvain,1 Place des Doyens, 1348 Louvain-la-Neuve, Belgique .

[Web] https ://www.lucidchart.com/pages/fr/tutoriel-sur-les-diagrammes-de-classes. 25-03-2018 .

[Zahoui, 2014] Zahoui, Fatima Zohra. (2014).*Devloppemant d'une chaine d'outils en fonction du nouveau standard fondationnel uml(FUML)*.Mémoire de magister. Université Badji Mokhtar, Annaba.pp :18-20 .