République Algérienne Démocratique et Populaire
Ministère de L'enseignement Supérieur et de La Recherche Scientifique

Université de Constantine

Faculté des Sciences de l'Ingénieur

N° d'ordre:
N° de série:

Mémoire de Magistère dans le cadre de l'école doctoral pôle Est
Option : Génie logiciel

Thème

"De M-UML vers les réseaux de Petri « Nested Nets » : Une approche basée transformation de graphes"

Réalisé par : HETTAB ABDELKAMEL **Encadré par**: Chaoui Allaoua

Soutenue devant le jury composé de :

Mr. Kholladi Mohemed Khireddine **Président**

Mr. Chaoui Allaoua **Rapporteur**

Mr. Saidouni Djamel Eddine **Examinateur**

Mr.Chikhi Salim **Examinateur**

Novembre 2009

Dédicaces

A nos très chères personnes au monde, qui sans eux nous

Ne serons arrivées là, ma très chère mères, source de

tendresse et de courage, et

mon très chère pères source de patience,

leurs souhaites bonne santé et belle vie.

A ma très chère femme

Et tous les membres de sa famille.

A mes frères et ma sœur,

A tous les membres de ma famille

A tous mes amis

A tous nos collègues de la promotion 2007 surtout Boubacr et

Naofle.

A tous ceux qu'on aime et qui nous sont chères.

On dédie ce modeste travail.

Hettab Abdelkamel

Remerciement

Nous remercions ***DIEU*** *le tout puissant qui m'a donné la force, la volonté et le courage pour accomplir ce modeste travail.*

Je tiens à formuler mon gratitude et mon profonde reconnaissance.

A l'égard de mon promoteur M. ***Allaoua Chaoui*** *qui m'a toujours accueilli avec bienveillance qui n'a ménagé ni son temps ni ses efforts pour me guider.*

Mes remerciements aux membres de jury M. ***Kholladi Mohemed Khireddine,*** *M.* ***Saidouni Djamel Eddine*** *et M.* ***Chikhi Salim*** *qui ont accepté de juger notre travail.*

Mes remerciements aux M. ***Elhillali Kerkouche,*** *M.* ***Mouhamed Rida Bahri*** *et mon collège M.* ***Boubacr Achichi*** *qui sont me aidés pour accomplir ce travail.*

Enfin nous exprimons notre profonde reconnaissance à tous les enseignants de l'université de Constantine et surtout mes enseignants de l'école doctoral pôle Est.

Abstract

The mobile agent applications are more and more take place in informatics world in last years, and the definition of a mobile agent is an agent capable of moving from platform to another during his life cycle. This particularity of mobile applications is due to birth of a numerous approaches and methods for modelessing them, among those approaches we interested by a semi formal approach in precisely mobile UML.

In way mobile UML is an extension of UML language for modeling the mobile agent application, it use the concepts of UML standard at add a new concepts of mobility. And an anther way ***Nested Petri Nets*** is an extension of ***Petri Nets*** and it is used for verifying mobile systems.

In this work we present a transformation from Mobile Stateshart Diagram of mobile UML (M-UML) to Nested Nets formalism on based of graph transformation approach and rules for transforming between two deferent formalisms. This transformation is implementing by **ATOM3** tool, transformation rules, and the two metamodels (mobile statshart diagram and Nested Nets).

Finally this work is given good results.

Keywords: mobile agent, mobile UML, Nested Petri Nets, graph transformation, Atom3.

Résumé

Les applications d'agent mobile sont plus en plus prennent sa place dans le monde informatique sur les dernières années, et dans sa définition un agent mobile est agent informatique capable de se déplace d'une plateforme à une autre durant son cycle de vie. Cette particularité des applications mobile due à la naissance de nombreux approches et méthodes pour les modéliser, parmi ces approches on s'intéresse par l'approche semi formelle et plus précisément UML mobile.

D'un côté UML mobile est une extension du langage UML pour modéliser les applications d'agent mobile, elle utilise les concepts standards d'UML en ajoutant des concepts propres à la mobilité, et d'une autre côté les réseaux de Pétri sont utilisés pour des raisons de vérification et il existe une extension des RDPs adaptable avec les systèmes adaptatives et mobiles ce qu'on appelle Nested Nets.

Dans ce travail nous avons présenté une transformation de diagramme d'état transition mobile de M_UML vers les RDPs Nested Nets en se basant sur le paradigme de la transformation de graphe, notre approche consiste à proposer une grammaire de graphe et des règles pour la transformation entre deux formalismes différents. La transformation est implémentée à l'aide de l'outil **AToM3**, en plus des règles de transformation et les deux métas modèles (diagramme d'état transition mobile et les RDPs Nested Nets).

Finalement nous avons argumenté notre proposition avec deux exemples réels et on a abouti des bons résultats.

Mots clés: agent mobile, UML mobile, Nested Nets, transformation de modèle, transformation de graphe, AToM3.

ملخص

إن تطبيقات الأجنت موبايل (Agent mobile) أخذت لها مكانة هامة في عالم الإعلام الآلي و المعلوماتية، وبالاعتماد على تعريفها (الأجنت موبايل هو أجنت قادر على التنقل من بلات فورم (Plateforme) إلى بلات فورم أخرى خلال دورة حياته) نجد أنها متميزة في مصطلحاتها فقد ظهرت إلى الوجود عدة طرائق من أجل دراسة و نمذجة هذه التطبيقات، و سنقوم في هذا العمل بالتركيز على احدى هذه الطرائق و هي الطريقة تصف الرياضية و بالأخص لغة النمذجة أي آم آل موبايل (UML mobile).

من جهة أي آم آل موبايل(UML mobile) هي امتداد للغة البرمجة أي آم آل (UML) من أجل نمذجة تطبيقات الأجنت موبايل (Agent mobile) ، فهي تستعمل مصطلحات أي آم آل (UML) و تضيف إليها المصطلحات الخاصة بتطبيقات الأجنت موبايل (Agent mobile)، و من جهة أخرى تستعمل شبكات بيتري (réseaux de Pétri) من أجل فحص و مراجعة النماذج و يوجد امتداد لشبكات بيتري (réseaux de Pétri) من أجل نمذجة تطبيقات الأجنت موبايل(Agent mobile) يسمى ناستد نات(Nested Nets).

في عملنا هذا سنقوم بتقديم قواعد بيانية (grammaire de graphe) تسمح لنا بعملية تحويل الرسم البياني الحالة انتقال للأي آم آل موبايل (diagramme d'état transition mobile) إلى لشبكات بيتري ناستد نات(Nested Nets).

في النهاية لقد قمنا بتجربة عملنا هذا على مجموعة من الأمثلة وقد أعطت نتائج طيبة.

كلمات مفتاحيه: الأجنت موبايل (Agent mobile) ، أي آم آل موبايل (UML mobile) ، ناستد نات (Nested Nets) ، تحويل الرسم البياني.

Table des matières

CHAPITR 2 : la modélisation UML

CHAPITRE 3 La transformation de modèle

CHAPITRE 4 Les réseaux de Petri

CHAPITRE 5 Notre approche de transformation de modèle

Liste des figures

CHAPITRE 1 Les agents mobiles

CHAPITRE 2 la modélisation UML

CHAPITRE 3 La transformation de modèle

CHAPITRE 4 Les réseaux de Petri

CHAPITRE 5 Notre approche de transformation de modèle

Liste des tableaux

Introduction générale

Introduction général

La programmation par agents mobiles est un paradigme de programmation des applications réparties, susceptible de compléter ou de se substituer à d'autres paradigmes plus classiques tel le passage de messages, l'appel de procédure à distance, l'invocation d'objet à distance, l'évaluation à distance. Elle est d'un grand intérêt pour la mise en œuvre d'applications dont les performances varient en fonction de la disponibilité et de la qualité des services et des ressources, ainsi que du volume des données déplacées. Le concept d'agent mobile facilite en effet la mise en œuvre d'applications dynamiquement adaptables, et il offre un cadre générique pour le développement des applications réparties sur des réseaux de grande taille qui recouvrent des domaines administratifs multiples.

Les besoins croissants des applications d'agents mobiles sur les dernières années ont conduit à de nombreuse approches et méthodes pour les modéliser tels que les extensions du langage UML. Parmi ces extensions en citant UML mobile comme un très bon outil pour modéliser ces applications mobiles. En conséquence UML mobile comme UML est une cadre méthodologique, c'est une norme, un langage de modélisation objet, un support de communication,…etc. Malgré UML comme UML mobile est tout cela, la vérification de tels modèles est une tâche assez délicate, à cause de la sémantique semi-formelle d'UML.

Dans le cadre de l'approche MDA (Model Driven Architecture) la transformation de modèle joue un rôle fondamental ; Le processus de conception peut alors être vu comme un ensemble de transformations de modèles partiellement ordonné, chaque transformation prenant des modèles en entrée et produisant des modèles en sortie, jusqu'à obtention d'artéfacts exécutables.

Pour bénéficier des avantages d'UML mobile dans la modélisation des systèmes mobile sans omettre la tâche de vérification on a essayé de faire une transformation des diagrammes d'UML mobile vers les réseaux de Petri ***Nested Nets*** (une extension des RdPs ordinaire pour modéliser les systèmes adaptative et mobile) en se basant sur le paradigme de transformation de graphe.

Notre mémoire est décomposé en cinq chapitres:

Dans le premier chapitre on a introduire la notion d'agent mobile et ses application en commençant par une introduction sur les agents dans le cas général et les dif

férents type d'agent, suivie par une brève représentation de paradigme d'agent mobile, et enfin une représentation des quelque plateformes pour implémenter les applications d'agent mobiles.

Dans le deuxième chapitre la modélisation UML est abordée et les extensions du langage UML pour modéliser les applications d'agent mobile sont présentées. Dans ce chapitre, nous avons présentés une introduction sur la notion de modélisation, suivie par une brève introduction au langage UML, ensuite, la notation UML et enfin une représentation précise des extensions d'UML pour modéliser les applications d'agent mobile.

Le troisième chapitre présente le paradigme de transformation de modèle en focalisant sur la transformation de graphe comme une approche de celle-ci. Dans ce chapitre, on va présenter le concept de transformation de modèles qui joue un rôle fondamental dans l'approche MDA en commençant par une représentation des concepts de transformation de modèle dans le cadre général, ensuite, on présente une énumération des différents types de transformations, suivie par une classification des différents approches, et enfin on présente, aussi, un cadre spécifique des transformations de modèles basé sur les transformations de graphe.

Dans le quatrième chapitre la modélisation par Réseaux de Petri (RdPs) est rappelée, les propriétés des RdPs sont ensuite discutées. Les extensions des RdPs sont mises en perspective. Le chapitre se termine par une discussion succincte sur la modélisation des systèmes adaptatifs et mobiles par les RdPs.

Dans le cinquième chapitre on a présenté notre proposition d'une grammaire de graphe pour transformer les diagrammes d'état transition mobile vers les réseaux de Petri ***Nested Nets*** en utilisant l'outil **Atom3**. Dans ce chapitre on a commençais par une proposition d'un métamodèle pour le diagramme d'état transition mobile, suivie par une autre proposition d'un métamodèle pour le formalisme de RdP Nested Nets, et enfin nous avons proposés une grammaire de graphe pour transformer le formalisme source vers le formalisme cible.

Chapitre 01: Les agents mobiles

1.1 Introduction

La programmation par agents mobiles est un paradigme de programmation des applications réparties, susceptible de compléter ou de se substituer à d'autres paradigmes plus classiques tel le passage de messages, l'appel de procédure à distance, l'invocation d'objet à distance, l'évaluation à distance. Elle est d'un grand intérêt pour la mise en œuvre d'applications dont les performances varient en fonction de la disponibilité et de la qualité des services et des ressources, ainsi que du volume des données déplacées. Le concept d'agent mobile facilite en effet la mise en œuvre d'applications dynamiquement adaptables, et il offre un cadre générique pour le développement des applications réparties sur des réseaux de grande taille qui recouvrent des domaines administratifs multiples.

Dans ce chapitre, on présente une introduction dans le paradigme d'agent mobile en commençant par une introduction sur les agents dans le cas général et les différents type d'agent, suivie par une brève représentation de paradigme d'agent mobile, et enfin une représentation des quelque plateformes pour implémenter les applications d'agent mobiles.

1.2 Les agents informatique

1.2.1 Définition

Il n'existe pas encore un consensus sur la définition d'un agent. En plus de la relative jeunesse du domaine, une des raisons est que diverses communautés revendiquent ce terme avec des problématiques parfois au départ assez différentes (par exemple en ce qui concerne les agents mobiles) même si ces différentes problématiques sont complémentaires et conduites à se rencontrer à terme.

Nous présentons Quelques définitions alternatives ont été données :

[BL-MO]: « An agent is a program that assists people and acts on their behalf, agents function by allowing people to delegate work to them».

[Ferber 95] :

On appelle agent une entité physique ou virtuelle

- qui est capable d'agir dans un environnement,
- qui peut communiquer directement avec d'autres agents,

- qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
- qui possède des ressources propres,
- qui est capable de percevoir (mais de manière limitée) son environnement,
- qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
- qui possède des compétences et offre des services,
- qui peut éventuellement se reproduire,
- dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

[Wool] : « Un agent est un système informatique situé dans un environnement et capable d'actions autonomes afin d'atteindre ses objectifs de conception ».

[PERRET] : « Un agent est une entité active autonome agissant par délégation pour le compte d'un client ».

Cette définition assez générale, du concept d'agent met l'accent sur :

A. L'agent est une entité qui agit par *délégation*, l'agent doit respecter la stratégie du client vis à vis des choix qu'il est amené à faire, afin que le client soit responsable des tâches effectuées par l'agent ;

B. L'agent est une entité *autonome* qui dispose de son propre environnement, c'est à dire que l'agent n'est pas lié constamment au client qui l'utilise, afin que le client ne dépende pas des contraintes vécues par l'agent.

1.2.2 Capacités d'un agent

A. **Autonomie :** Un agent autonome est un agent qui agit dans son environnement et qui ne se contente pas d'être dirigé par des commandes venant d'un autre agent ou de l'opérateur. Un tel agent doit posséder des effecteurs et être situé dans son environnement. L'autonomie d'un agent ne porte pas seulement sur son comportement mais aussi sur ses ressources internes qui sont elles aus-

si autonomes **[DRIEU]**. L'agent agit de façon indépendante et évolue sans aucune orientation extérieure. Un exemple courant d'agent autonome est le ***« démon ».***

B. **Adaptation** : Un agent adaptatif est capable d'apprendre en fonction de son expérience passée et de son évolution. Les architectures d'agents produisant des unités fonctionnelles dédiées à des tâches précises, les techniques d'apprentissage classique de l'intelligence artificielle s'appliquent particulièrement à l'apprentissage d'un agent en particulier, il peut réagir différemment selon l'agent avec lequel il interagit. Cette caractéristique nécessite des capacités d'apprentissage, par exemple pour élaborer des cartes cognitives ou pour modifier ses préférences comportementales en fonction de son expérience. L'adaptation peut également être collective : en utilisant des mécanismes d'évolution, des populations d'agents peuvent apprendre à s'adapter à leur environnement.

C. **Communication** : La possibilité d'échanges de messages entre agents. Plusieurs langages existent, dont par exemple BlackBoard, KQML, KIF, COOL.

D. **Coopération :** Un agent coopératif est un agent capable de raisonner sur ses propres compétences et sur les compétences des autres agents en vue de résoudre un problème qu'aucun agent n'est capable de le résoudre individuellement.

E. **Mobilité :** C'est la capacité d'un agent de migrer d'une machine vers à une autre, et ce pour exécuter des tâches pour le compte de son client.

F. **Apprentissage :** Un agent possède cette propriété s'il est capable d'utiliser ses connaissances afin de modifier son comportement (augmentation de l'espace disque, détection d'un seuil, etc.).

G. **Pro-Activité :** Elle caractérise la capacité d'un agent à anticiper des changements plutôt qu'à réagir au changement. Par exemple, il s'agit d'appliquer une règle d'administration particulière dès qu'une variable de l'environnement atteint un certain seuil, c'est-à-dire agir avant que le seuil ne soit atteint.

H. **Réactivité :** Elle caractérise la capacité d'un agent à effectuer une nouvelle tâche lorsqu'un événement survient.

1.2.3 Les différents types d'agents

Le terme agent recouvre beaucoup de sens différents suivant les communautés et les domaines d'applications et donc les perspectives et techniques mises en avant.

La classification des différentes architectures d'agents, basée sur des critères pertinents et sur un mode naturaliste (*taxinomie*) est un mécanisme nécessaire à la compréhension des architectures existantes. Parmi les classifications existantes il existe une taxinomie **[DRIEU], [Morge]** des agents logiciels repose sur un besoin de nommer précisément les architectures existantes en fonction de leur aspect opérationnel ou architectural. Dans cette taxinomie les agents sont repartis dans les classes suivantes: Les agents réactifs, Les agents intentionnels, Les agents autonomes, Les agents adaptatifs, Les agents mobiles, Les agents flexibles et Les agents sociaux.

Il existe d'autre classification basée sur « domaines » ou « points de vue » sur les agents. Elle corresponde pour certains à différents points de vue historiques qui auront forgé une certaine vision des agents, et pour d'autres à des points de vue applicatifs qui auront amené certaines problématiques plus ou moins spécifiques (par exemple, la mobilité d'agents). Cette classification ne prétend donc pas être une typologie complète, mais se veut plutôt un tour d'horizon **[Briot-D]**.

A. **Agent cognitif :** L'intelligence artificielle (IA) classique s'est concentrée très tôt sur l'expression, sur des bases logiques, du comportement délibératif d'un agent rationnel en fonction de ses croyances et buts. Ceci a donné lieu plus tard aux premières architectures modernes d'agents dits cognitifs.

B. **Agent logiciel :** Les premiers et les plus simples sont les démons Unix (processus informatiques autonomes capables de se réveiller à certaines heures ou en fonction de certaines conditions). Les virus informatiques en sont des versions déjà plus sophistiquées (notamment douées de la capacité de reproduction) et malfaisantes. Notons qu'il est difficile de donner une définition actuelle (consensuelle) du terme agent logiciel. Cela sous-entend souvent aussi une encapsulation en agent (" agentification") de programmes ou de bases de données plus traditionnels, de manière à favoriser leur coopération. En tout cas, ce terme s'oppose au terme agent physique, tel un robot.

C. **Agent assistant :** Pour dépasser les limitations des interfaces homme machine à manipulation directe (rigidité, complexité, etc.), les agents assistants (aussi appelés agents intelligents), apportent une adaptation au profil de l'utilisateur et une capacité à anticiper leurs besoins (automatiser certaines tâches, rappeler certaines informations utiles, etc.). Ceci peut se transposer dans le domaine du collectif. Prenons l'exemple d'un rendez-vous entre plusieurs personnes, mis en place par la coopération de leurs agents assistants respectifs pouvant résider sur des agendas électroniques (PDAs) ou ordinateurs.

D. **Agent robotique :** On peut considérer l'architecture logicielle de contrôle d'un robot comme un agent. Cette architecture peut d'ailleurs être testée et entraînée en simulation avant d'être mise en oeuvre dans un vrai robot. Cependant la réalité physique d'un robot apporte des problèmes spécifiques (imprécision de la perception et de l'action, aspects temps réel, évolution du monde) qui rendent particulièrement difficile, mais aussi particulièrement riche, la conception de tels agents robotiques. On peut aussi envisager des coopérations entre robots, par exemple dans le cadre de la compétition de robots footballeurs (RoboCup), présentée par ses organisateurs comme un nouveau problème étalon de l'intelligence artificielle

E. **Agent mobile :** La première plate-forme d'agents mobiles est Telescript, née dans la première moitié des années 90. Elle a depuis donné lieu à de nombreux descendants, extensions du langage Java (Aglets, Voyager, etc.).

1.3 Les agents mobiles

La mobilité est une propriété non obligatoire des agents. En effet, tous les agents ne sont pas mobiles. Un agent peut communiquer avec son environnement en utilisant des moyens conventionnels comme les RPC (Remote Procedure Calling). Ce type d'agents est dit stationnaire ou statique.

Un agent statique est un agent qui s'exécute seulement dans le système où il commence son exécution. S'il a besoin d'information non disponible dans le système ou a besoin d'interagir avec un agent dans un système différent, il utilise un mécanisme de communication tel que RPC.

Par contre, un agent mobile n'est pas lié au système dans lequel il débute son exécution. L'agent mobile est capable de se déplacer d'un hôte à un autre dans le réseau. Il peut transporter son état et son code d'un environnement vers un autre dans le réseau où il poursuit son exécution.

1.3.1 Définition d'un agent mobile

Un agent mobile est un agent capable de se déplacer dans son environnement, qui peut être physique (réel ou simulé) ou structurel (niveaux d'exécution par exemple). Un agent mobile dispose donc de dispositifs assurant sa mobilité **[Briot-D].**

1.3.2 La création et la mort d'un agent mobile

Un agent est créé dans une place. La création peut être initiée soit par un autre agent résidant dans la même place ou par un agent ou un système non agent en dehors de la place. Le créateur doit s'authentifier dans la place et établir les autorisations et les références que va posséder l'agent. Le créateur peut aussi définir des arguments d'initialisation pour l'agent crée.

La classe de définition nécessaire pour instancier l'agent peut résider dans l'hôte local ou sur une machine distante ou fournie par le créateur. La création se fait en trois étapes :

1.3.2.1 L'instanciation et l'attribution de l'identificateur

Le code de la classe agent est chargé puis exécuté. L'objet agent est instancié. La place assigne un identificateur unique à l'agent.

1.3.2.2 L'initialisation

L'agent peut s'initialiser en utilisant les arguments d'initialisation fournis par son créateur. Une fois l'initialisation achevée, l'agent est complètement installé dans la place.

1.3.2.3 L'exécution autonome

Après l'installation, l'agent peut commencer son exécution. Il est maintenant capable de s'exécuter indépendamment des autres agents dans la même place.

Dans la plupart du temps, l'agent est détruit lorsqu'il quitte sa place. Ce processus peut être initié par l'agent lui même, ou par un autre agent ou système non-agent résidant ou non dans la même place. L'agent peut aussi être renvoyé de sa place par le système pour l'une des raisons suivantes :

- La fin de sa" durée de vie ",
- L'agent n'est pas utilisé ou référencé,
- Violation des règles de sécurité,
- Le système est arrêté.

Le processus de destruction est amorcé en deux étapes :

- La préparation : Une chance est donnée à l'agent de finir sa tâche courante avant d'être renvoyé.
- Suspension de l'exécution : La place suspend l'exécution de l'agent.

1.3.3 Migration d'un agent mobile

Le processus de transfert peut être initié par l'agent lui même, par un autre agent résidant dans la même place ou par un agent ou système non agent résidant dans une autre place. L'agent est ensuite transféré de sa place courante, pour être reçu par la place (destination) spécifiée. **(Voir Figure 1.1)**

La place d'origine et la place destination gèrent le processus de transfert. Quand la place d'origine contacte la place de destination, celle-ci peut soit réaliser le transfert, soit envoyer un message d'échec à la place d'origine. Si la place d'origine ne peut pas contacter la place destination, elle doit retourner un message d'échec à l'agent.

A. L'envoi de l'agent :

Quand l'agent mobile se prépare pour son voyage, il doit être capable d'identifier sa destination. Si la place est non spécifiée, l'agent s'exécutera dans une place par défaut, sélectionnée par le système d'agent de destination. Une fois la localisation de la destination est établie, l'agent mobile informe le système d'agent local qu'il veut se transférer au système d'agent de destination. Ce message est retransmis via un API interne entre l'agent et le système d'agent.

Quand le système d'agent reçoit la requête de transfert, il doit entreprendre les actions suivantes :

- Suspendre l'agent : L'agent est d'abord prévenu à propos de son transfert, et un temps lui est alloué pour terminer sa tâche courante. Une fois ce temps écoulé, son thread d'exécution est arrêté.

- Sérialiser l'agent : L'agent (composé de son état et de la classe agent) est sérialisé par un moteur. La sérialisation est le processus de création d'une représentation persistante de l'objet agent qui peut être transportée à travers le réseau. La sérialisation de l'agent peut inclure son état d'exécution.
- Encoder l'agent sérialisé : Le moteur encode l'agent sérialisé pour le protocole de transport choisi.
- Transférer l'agent : Le moteur établit une connexion réseau avec l'hôte spécifié et entame le processus de transfert.

B. Réception de l'agent :

Avant que le transfert puisse avoir lieu, l'origine de l'agent doit s'authentifier auprès du moteur de réception. Lorsque l'authentification est réussie, le processus de transfert peut alors commencer :

- Réception de l'agent.
- Décoder l'agent.
- Désérialiser l'agent : La représentation persistante de l'agent est désérialisée, la classe agent est instancié et l'état de l'agent transféré est rétabli.
- Reprendre l'exécution de l'agent : L'agent "recrée" est notifié dans sa place destination. Un nouveau thread d'exécution lui est assigné.

C. Le transfert de la classe d'agent :

L'agent ne peut pas reprendre son exécution sans que ses classes ne soient présentes. Il y a plusieurs manières de mettre ces classes à la disposition du moteur de destination :

- La classe existe à destination *:*

Si la classe est déjà disponible dans l'hôte éloigné -soit dans la mémoire cache ou dans un fichier système local, alors il n'est pas nécessaire de la transférer. L'agent transféré n'a besoin que des informations requises pour identifier la classe, tel que le nom complet de la classe. Il peut aussi contenir des informations supplémentaires qui décrivent la localisation de la classe et sa définition.

- La classe existe à l'origine :

Si la classe est localisée à l'origine, comme il est souvent le cas, elle peut être facilement transportée avec l'état de l'agent au moteur destination. Cependant, dans la

figure 1.1, on observe que les classes peuvent être facilement transférées plusieurs fois ce qui peut engendrer une augmentation du trafic dans le réseau et un gaspillage de la bande passante.

– Le code à la demande :

Dans ce cas, la classe est disponible à partir d'un serveur et le moteur de destination peut la retirer en appliquant le principe du code à la demande. Cependant, notons que le moteur de destination doit, dans ce cas, réaliser une connexion réseau supplémentaire pour retirer la classe. Après l'instanciation de l'agent, celui-ci peut créer d'autres objets. Il est évident que les classes de ces objets sont nécessaires pour leur instanciation et pour la continuation de l'exécution. Si l'une de ces classes, n'est pas disponible dans le moteur de destination, elle doit être transférée soit à partir de l'origine où à partir du moteur qui a lancé l'agent.

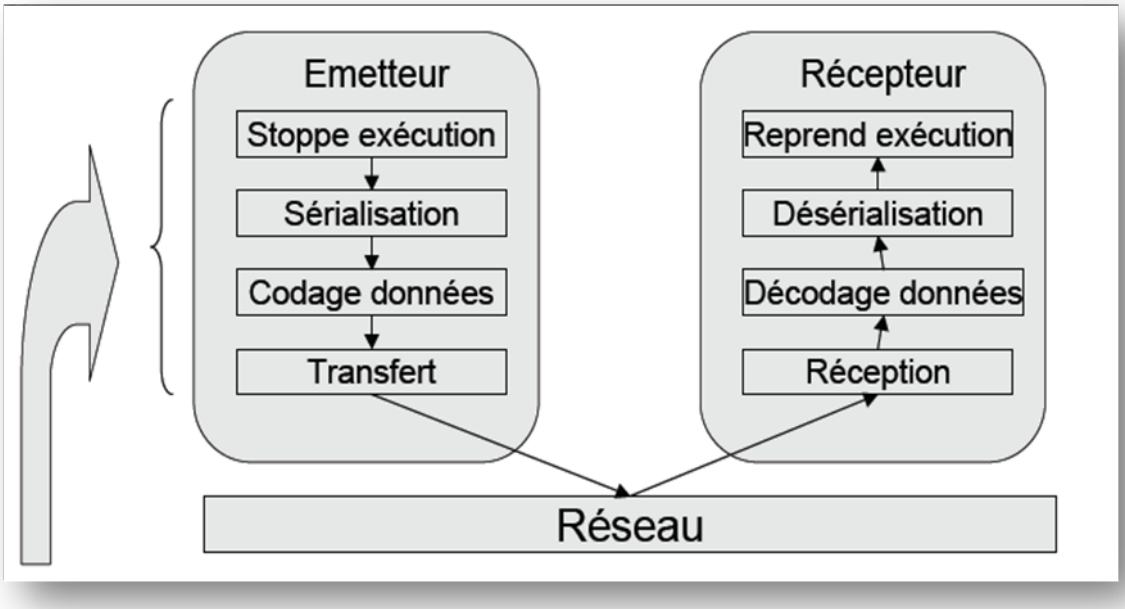

Figure 1.1 Le transfert de l'agent mobile

1.3.4 La communication entre les agents mobiles

Les agents peuvent communiquer avec d'autres agents résidents dans la même place ou avec des agents résidents dans d'autres places. Un agent peut évoquer une méthode d'un autre agent comme il peut lui envoyer des messages s'il est autorisé à le faire. La communication inter-agents peut suivre trois schémas différents :

A. **Now-type messaging** : C'est le type de messagerie le plus utilisée. C'est un type synchrone. Il bloque l'exécution de l'émetteur du message jusqu'à ce que le receveur aura complètement téléchargé le message et aura envoyé sa réponse. (**voir Figure 1.2)**

B. **Future-type messaging** : C'est un type de messagerie asynchrone non bloquant. L'expéditeur retient une variable, qui peut être utilisée pour obtenir le résultat. Ce type de messagerie est utile en particulier quand plusieurs agents communiquent ensemble. (**Voir Figure 1.2)**

C. **One-way-type messaging** : C'est un type asynchrone qui ne bloque pas l'exécution courante. L'expéditeur ne va pas retenir une variable pour ce message et le receveur ne va jamais répondre. Ce type est utile quand deux agents engagent une conversation où l'agent expéditeur n'a pas besoin de la réponse de l'agent récepteur. Ce type de messagerie est appelée **fire-and-forget.** (**Voir Figure 1.2**).

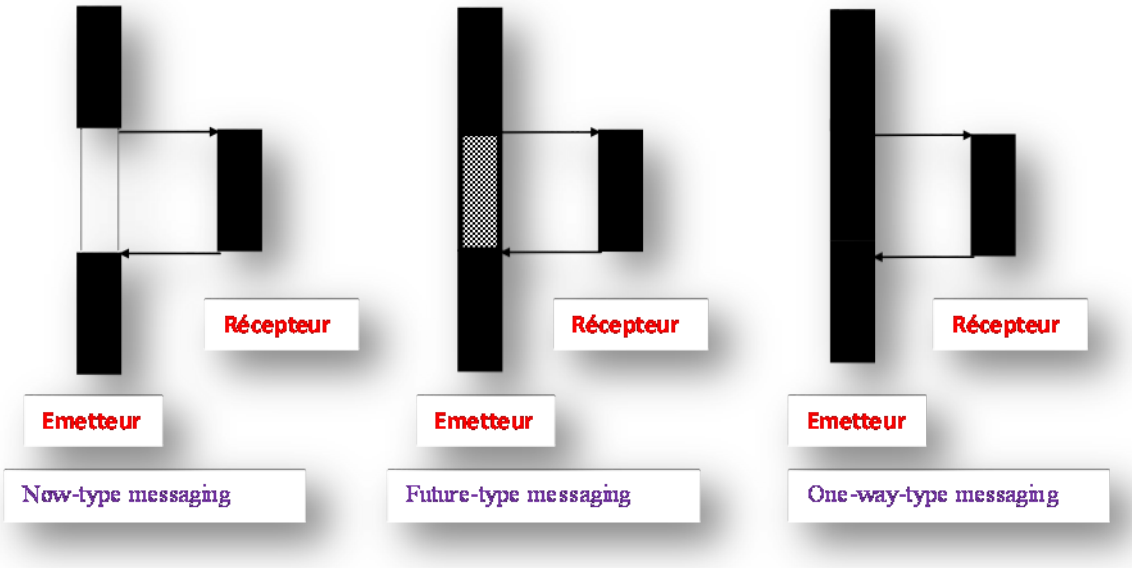

Figure 1.2 La communication entre les agents mobiles

1.3.5 Pourquoi les Agents Mobiles

Les principaux raisons sont :

– **Réduction de la charge Réseau :** Il est généralement plus avantageux, en terme de performances, de faire voyager du code plutôt que des données.

- **Déplacer le Code vers les données:** Les serveurs contenant des données procurent un ensemble fixe d'opérations. Un agent peut étendre cet ensemble pour les besoins particuliers d'un traitement.
- **Fiabilité:** La vie d'un programme classique est liée à la machine ou il s'exécute. Un agent mobile peut se déplacer pour éviter une erreur matérielle ou logicielle ou tout simplement un arrêt de la machine.
- **L'adaptation dynamique du système aux problèmes** : Les agents peuvent se répartir eux-mêmes sur des machines du réseau, afin de mieux tenir compte de l'état du système au moment où ils doivent exécuter leurs tâches.
- **La gestion de la robustesse et de la tolérance aux pannes** : Si une machine s'arrête, l'agent qui s'y trouve peut changer de machine pour terminer sa tâche en cours.
- **L'encapsulation de protocoles** : Dans les systèmes distribués les protocoles définissent comment les messages et les données sont échangées. Pour modifier un protocole, il faut changer le code sur toutes les machines du système. Les agents encapsulent les protocoles. Changer de protocole revient à interagir avec un nouvel agent.
- **L'autonomie des composants et l'exécution synchrone/asynchrone** : Les terminaux mobiles (PDA, ordinateurs portables, téléphones, ...) ne sont pas toujours connectés au réseau. Les systèmes qui nécessitent des connexions permanentes ne sont pas adaptés dans ce cas. Avec des agents, les terminaux mobiles peuvent se connecter pour vérifier s'ils ont des messages. Un agent peut être envoyé sur le terminal mobile au moment de la connexion et travailler sur ce dernier après avoir interrompu la connexion. Le même agent peut attendre la prochaine connexion pour envoyer des informations sur le réseau signifiant par exemple qu'une tâche a été terminée.

1.3.6 Domaine d'application

A. Compression des documents :

La récupération des documents de toutes sortes est une des utilisations principales du Web. Par exemple si un client veut obtenir une copie d'un document alors il le demande au prêt du serveur et récupère un flux de données correspondant au contenu du document. Afin d'optimiser le transfert de données le client pourra utiliser un outil

de compression des données. Pour cela il sera nécessaire d'étendre le serveur pour y ajouter cette fonction de compression. Différents clients peuvent utiliser différents outils de compression, nécessitant des extensions particulières pour chaque client

B. Commerce électronique :

Les principes d'autonomie et de délégation sont largement employés dans le commerce électronique. Les agents sont donc naturellement adaptés à ce type d'application. L'agent agit pour le compte d'une personne et applique différentes stratégies pour remplir les services escomptés. La mobilité de l'agent lui permet de se déplacer de site en site pour dialoguer et négocier avec les services présents afin de réaliser la meilleure transaction possible.

C. Recherche et filtrage d'information :

C'est le plus grand champ d'application et d'expérimentation des agents. Les premiers systèmes commerciaux utilisant des agents étaient des agents "moniteurs". C'étaient des programmes qui alertaient l'utilisateur quand une information intéressante apparaît. Alexa est une barre d'outils d'aide gratuite à la navigation. Elle procure des informations statistiques et des liens sur chaque site visité. Elle aide également au magasinage en ligne en vérifiant l'identité d'un site de commerce à partir des contacts donnés sur la page.

Les agents, et spécialement les agents mobiles, sont adaptés pour agir comme des "bots", des logiciels qui naviguent continuellement sur la toile pour trouver de nouvelles informations.

D. Services de télécommunication :

La fonctionnalité de mobilité prend toute son importance dans le domaine des télécommunications. Les agents mobiles peuvent y être utilisés pour couvrir toutes les couches des protocoles de communication, de la maintenance de réseau, jusqu'aux applications mobiles, suivant l'usager dans ses déplacements.

Un "Personal Communicator Agent" (PCA) est un agent mobile chargé de délivrer un message au destinataire, quelle que soit son appareil (téléphone, ordinateur, portable ou téléphone sans fil). Le PCA d'un utilisateur doit pouvoir recevoir les messages et les conduire sans interruption dans des réseaux hétérogènes à l'utilisateur. Par exemple, si le seul moyen de délivrer un message urgent à un utilisateur est via un téléphone sans fil, l'agent personnel doit convertir le message textuel en mes-

sage vocal. La mobilité permet à ces agents de suivre l'utilisateur même quand il change de machine.

E. Robotique :

Les agents peuvent aussi servir à aider le comportement de robots physiques. À l'aide de capteurs, les robots basent leurs comportements sur des agents qui traduisent l'information obtenue pour définir les actions à suivre (tourner, avancer etc..).Exemple : Astérix est un exemple. Il permet un dépistage de haut niveau en utilisant une opinion basée sur un comportement réactionnaire.

F. **Surveillance :**

Ces agents veillent sur des sites ou des thèmes définis par l'usager. Ils peuvent être utilisés pour être automatiquement alertés des dernières actualités dans un domaine donné. Les agents mobiles sont ici adéquats pour voyager dans le réseau et effectuer des tâches périodiques.

G. **Administration des réseaux :**

Le domaine de l'administration de réseau est également l'objet de nombreuses recherches, beaucoup pensent que le futur des réseaux réside dans une plus grande intelligence, pour plus d'adaptabilité et de mobilité; et que cette évolution passe par les agents mobiles.

L'administration de réseau est par nature asynchrone et distribuée. De plus, il est souvent important d'avoir une vue locale du système pour pouvoir déterminer les causes et les conséquences d'un problème. L'administrateur doit alors se déplacer vers des machines lointaines qui nécessitent des tâches de maintenance ou de mise à jour. L'installation et la maintenance des logiciels deviennent difficiles avec l'augmentation du nombre de machines.

1.3.7 Avantages et inconvénients

Les agents mobiles ont rapidement suscité un intérêt tout particulier dans les domaines de recherche portant sur les applications réparties. Très rapidement, cette nouvelle méthode de programmation a été évaluée afin de voir ce qu'elle pouvait apporter comme caractéristiques propres et ce qu'elle permettait d'améliorer par rapport aux méthodes de programmation plus classiques **[C.Cubat]**.

1.3.7.1 La performance

Les premières améliorations apportées par les agents mobiles portent sur le gain de performances dues à une meilleure utilisation des ressources physiques mises à disposition. L'amélioration va intervenir à différents niveaux permettant d'optimiser la tâche globale des agents.

A. **Diminution de l'utilisation du réseau**

Le déplacement des agents mobiles permet de réduire significativement, voir supprimer, les communications distantes entre les clients et les serveurs. En privilégiant les interactions locales, l'utilisation du réseau va se limiter principalement au transfert des agents. Cette situation présente trois principaux avantages:

La diminution de la consommation de bande passante. En effet, plusieurs études montrent qu'en comparaison de l'envoi à distance de requête (procédures et méthodes), la mise en place des agents mobiles permet d'obtenir une réduction significative de la charge réseau en termes de nombre total de données transférées. Cette diminution est constatée dans différents types d'applications nécessitant d'intenses échanges d'informations entre le client et le serveur. Nous pouvons donner à titre d'exemple, la collecte d'informations dans des bases de données réparties, l'exploration d'internet ou encore la gestion de réseaux.

Le deuxième avantage est la diminution des temps de latence. Dans le contexte des réseaux à large échelle, la mise en place d'applications reparties, nécessitant de fréquentes interactions entre client et serveur, se heurte aux temps de latence propres aux communications réseaux. Il arrive fréquemment que le temps d'attente de la réponse d'une requête soit plus long que le temps de traitement nécessaire à la réalisation du service. En rapprochant client et serveur dans un même sous-réseau, voire sur un même site, on les place dans un environnement où les temps de réponse des interactions sont limités, ce qui permet de réduire d'autant les temps de latence.

Le dernier avantage à souligner vient des brèves périodes de communication. En réduisant le plus possible les communications distantes aux seuls transferts d'agents mobiles, on diminue considérablement les périodes de connexion entre deux sites. Cette diminution de la fenêtre d'utilisation des communications réseaux permet de moins se soucier des ruptures de liens physiques qui peuvent intervenir fréquemment dans les environnements sans fil.

B. Des calculs indépendants

Dans le modèle classique d'évaluation à distance, le client et le serveur doivent rester connectés tant que le service est en cours d'exécution. En plus des problèmes liés aux fluctuations des performances réseaux, certains services, nécessitant de longues phases de traitement, ne supportent pas facilement une rupture de connexion avec le client. Dans ce cas, ils doivent souvent redémarrer entièrement leurs calculs. Mais, le maintien du lien de communication peut s'avérer difficile dans des réseaux à large échelle ou sans fil. Avec les agents mobiles, un client peut déléguer les interactions avec le service sans maintenir une connexion de bout en bout. Avec cette possibilité d'un calcul indépendant, le client peut demander un service, se déplacer (ou simplement terminer une session) puis venir récupérer les résultats plus tard. Ce mode de fonctionnement est particulièrement intéressant lors du lancement à distance de simulations numériques.

C. Optimisation du traitement

L'optimisation des phases de traitement se produit à deux niveaux. Premièrement, comme nous l'avons déjà dit, en localisant les ressources et le savoir faire sur un même site. On supprime les phases de dialogue entre le client et le serveur qui sont perturbées par des temps de latence dus aux communications réseaux. Ensuite, le déplacement du savoir faire va permettre de déléguer les calculs sur des machines serveurs, un supercalculateur par exemple, qui sont généralement plus puissantes qu'une machine cliente. Cela est particulièrement vrai dans l'informatique nomade où la miniaturisation s'accompagne d'une perte de puissance significative.

D. Tolérance aux fautes physiques

En se déplaçant avec leur code et données propres, les agents mobiles peuvent s'adapter facilement aux erreurs systèmes. Ces erreurs peuvent être d'ordre purement physique, disparition d'un nœud par exemple, ou d'ordre plus fonctionnel, arrêt d'un service par exemple. Si on prend le cas d'un site perdant une partie de ses fonctionnalités, un service tombant en panne, l'agent pourra alors choisir de se déplacer vers un autre site contenant la fonctionnalité désirée. Ceci permet une meilleure tolérance aux fautes que le modèle statique classique.

Cependant, notons que dans le cas d'un agent unique, s'il vient à disparaître pour une erreur interne à son savoir faire ou pour une erreur système, il est généralement très difficile de détecter sa disparition. 'Siegfried Rouvrais'. énumère plusieurs

mécanismes basés sur la réplication, transparents aux agents, permettant de résoudre en partie ce problème.

1.3.7.2 La conception

Du point de vue de la conception, les agents mobiles représentent à la fois une méthode permettant de mieux caractériser certaines applications mais ils apportent aussi une complexité accrue par rapport au classique client/serveur bien mieux maîtrisé. Pour commencer, en règle générale, les concepteurs préfèrent avoir une méthode permettant de décrire facilement un comportement réel. Avec la méthode classique, il est très contraignant de décrire des algorithmes d'exploration (de réseau) ou bien encore de caractériser les déplacements des utilisateurs nomades.

Avec les agents mobiles, les concepteurs disposent d'une méthode qui permet de décrire naturellement ce genre de comportement. Ainsi, on peut facilement mettre en place un déploiement et/ou une maintenance d'application sur un réseau ou encore suivre les utilisateurs dans leurs déplacements.

Ensuite, les agents possèdent une capacité de traitement spécifique leur permettant de s'adapter à leur environnement. Généralement, dans le modèle client/serveur, le service est caractérisé préalablement par une interface stricte et/ou avec un protocole d'utilisation bien défini. Ainsi, si le client n'a pas connaissance de la description du service, il ne pourra pas l'utiliser. Par contre, les agents pourront eux s'adapter aux caractéristiques des services afin d'exprimer leur requête. Par exemple, si un service demande une communication sécurisée, l'agent pourra récupérer un module de sécurité, mettre à jour sa pile de communication et dialoguer avec le service. On peut aussi tout à fait imaginer que l'agent pourra s'adapter aux conditions du réseau en se séparant d'une partie de ses fonctionnalités pour s'adapter aux terminaux pauvres en ressources.

La capacité de raisonnement va permettre de concevoir des agents qui seront autonomes, adaptant leurs déplacements en fonction de l'environnement et pouvant moduler leurs fonctionnalités en cours d'exécution. Cependant ces propriétés s'accompagnent d'une conception bien plus difficile que celle du classique client/serveur. En effet, dans la majorité des applications distribuées, on essaie le plus possible de cacher la répartition en s'appuyant sur un ensemble de services système qui permettent de concevoir une application comme si tous ses éléments étaient locaux. C'est le cas de CORBA par exemple. Pour pleinement tirer partie des agents mobiles, la distribution

doit être explicite et gérée par les agents eux-mêmes, compliquant d'autant la tâche des concepteurs. De plus, si on se réfère à la conception objet où une application est définie comme un ensemble d'éléments et de relations, il est difficile de bien définir la tâche de chaque agent (élément) et surtout de savoir comment et où les interactions (relations) vont s'opérer. Enfin, la complexité même des services rendus par les applications impose souvent aux concepteurs d'utiliser une méthode parfaitement maîtrisée plutôt que de recourir à un mécanisme dont ils n'ont pas vraiment l'habitude.

1.3.7.3 Le développement

Le domaine des agents mobiles étant encore relativement jeune, ils se heurtent à de fortes contraintes lors des phases de développement. La première d'entre elles est qu'il existe à l'heure actuelle un trop grand nombre d'intergiciels pour agents mobiles qui possèdent chacun leurs propres défauts et qualités **[C.Cubat]**. Il suffit pour s'en convaincre de regarder le site de la « Mobile List » **[C.Cubat]** qui référence toutes les plates-formes connues. Avec cette offre pléthorique, il est difficile de parler de standardisation et aucune ne semble encore s'imposer. En sachant que les agents doivent s'adapter aux conditions de l'environnement, les développeurs sont confrontés à un éventail de possibilités bien trop large.

Le manque de standardisation se retrouve aussi dans les interactions entre agents, ainsi il n'existe pas de langage partagé par toutes les plates-formes. Ceci représente un handicap sérieux, car les agents ont besoin d'exprimer les caractéristiques des services qu'ils recherchent et surtout d'obtenir des réponses précises sur leur localisation ainsi que sur la manière de les utiliser. De plus, avec tous les langages existants, les développeurs sont, encore une fois, face à un trop large éventail de possibilités. Pour résoudre ce problème, il faudra se tourner vers le domaine des systèmes multi-agents qui cherche à mettre une place des langages précis en s'appuyant sur un ensemble d'ontologies et de protocoles caractérisant les interactions inter-agents.

Le second problème important lors des phases de développement est la complexité de mise au point des programmes qui constitue une partie critique du processus de développement. La plupart des environnements de programmation possèdent des outils permettant de suivre les différents éléments d'une application durant son exécution afin d'en trouver les erreurs. Cependant, ce genre d'outil s'utilise parfaitement avec des éléments statiques mais est difficilement applicable dès le moment où les éléments se déplacent. Ceci est particulièrement vrai dans les architectures hybrides

et/ou à grande échelle où il est quasiment impossible de surveiller l'intégralité d'un système. Ainsi, il peut s'avérer impossible de récupérer l'état d'erreur d'un élément se trouvant sur un site déconnecté. Sans outil de débogage, il devient très difficile de différencier les erreurs de conception et de développement. Notons que des efforts pour proposer un premier outil de débogage sont réalisés par la plate-forme JADE.

Le troisième problème important, en relation avec le débogage, est la difficulté de mettre en place une vérification des applications à base d'agents mobiles. La technique la plus utilisée encore de nos jours, bien qu'elle ne soit pas totalement satisfaisante, est sans aucun doute le test. Dans cette méthode, on va mettre à l'épreuve une application en lui injectant des données dans ses différents points d'entrée dans le but de simuler le comportement utilisateur et de recouvrir une plage de situations la plus large possible. Si le test s'applique assez naturellement dans des programmes classiques, i.e. non répartis, il représente un domaine de recherche à part entière dès qu'il s'agit des applications réparties. Dans ce cadre, il faut être capable de coordonner les différentes injections sur les points d'entrée répartis afin de garantir que le comportement simulé est bien celui qui était visé. Cette méthode est déjà loin d'être simple dans le cas d'une application répartie statique, on peut alors facilement imaginer que cela devient encore plus compliqué si les points d'entrées (les agents) deviennent mobiles. Pour vérifier les applications construites sur des agents mobiles, il faut utiliser une méthode permettant de garantir le comportement et les interactions des agents avant leur déploiement.

L'analyse statique de programme permet de déterminer statiquement, sans exécuter un programme, des propriétés qui seront satisfaites lors de l'exécution. Cette méthode, qui permet de garantir un certain niveau de correction du programme analysé, semble toute indiquée pour permettre de se passer des phases de test et, ce, même dans le cas des applications réparties. Notons tout de même que l'analyse statique à fait ses preuves dans le cadre de la programmation classique mais qu'elle constitue toujours un domaine de recherche pour les applications réparties.

1.3.7.4 La sécurité : le talon d'Achille des agents mobiles

D'un point de vue logiciel, la sécurité consiste à empêcher des accès, et/ou des modifications, non-autorisés aux éléments d'un système informatique. Dans ce cadre nous faisons la distinction entre les demandeurs (utilisateur, processus, périphérique ...) et les éléments demandés (processeur, fichier, mémoire ...). Lorsque l'on souhaite

avoir un système sûr, on met en place une politique de sécurité qui doit garantir la confidentialité, i.e. les données des éléments ne sont pas divulguées aux demandeurs non-autorisés, et l'intégrité, i.e. les éléments ne sont pas modifiés par des demandeurs non-autorisés. Nous omettons volontairement la disponibilité qui doit garantir, aux demandeurs autorisés, l'accessibilité aux éléments et qui est plus du ressort de la tolérance aux fautes.

Pour mettre en place la politique de sécurité visée, on utilise généralement des mécanismes d'authentification (mot de passe, certificat ...), de cryptographie (SSH, SSL ...) et de contrôle d'accès (droit utilisateur, pare-feu). L'authentification à pour but d'identifier précisément le demandeur, la cryptographie doit assurer la confidentialité des données échangées et le contrôle d'accès vérifie l'adéquation entre demandeur et éléments demandés. Dans le schéma classique des applications réparties, on regroupe les éléments critiques sur des machines sécurisées et on se focalise sur les canaux de communication externes en utilisant les mécanismes d'authentification et de cryptographie. Pour la mise en place de la mobilité de code, les politiques de sécurité se basent généralement sur une relation étroite entre le site stockant le programme et celui qui l'exécute. Implicitement, le possesseur du code est le même que celui de l'environnement qui va l'exécuter. Pour ce qui est des agents mobiles, d'un point de vue de la sécurité, nous sommes face à un problème qui n'est pas encore intégralement résolu. C'est d'ailleurs le principal argument avancé pour expliquer la faible utilisation de ce paradigme. En effet, les agents mobiles représentent un nouveau champ d'investigation pour le domaine de recherche en sécurité, d'une part dans la protection des sites vis-à-vis des agents malveillants et d'autre part dans la protection des agents vis-à-vis des sites malveillants.

A. Protection des sites

La protection des sites contre des attaques menées pas des agents malveillants est un problème maîtrisé. En effet, plusieurs solutions permettent de se prémunir d'éventuelles attaques et voici quelques méthodes :

- Bac à Sable : La technique du bac à sable consiste à exécuter un agent à l'intérieur d'un environnement restreint, en interdisant l'accès au système de fichiers par exemple. Ainsi, un site peut exécuter un agent douteux dans le bac à sable sans trop se soucier des problèmes de sécurité. Cette approche peut facilement se mettre en place en utilisant des interpréteurs de code dont leurs pos-

sibilités sont limitées. Pour illustrer cette technique nous pouvons citer les applets Java exécutés à l'intérieur d'un navigateur Web.

- Signature de code : La signature de code intervient lors de la création d'un agent, son créateur le signant numériquement afin qu'il puisse s'identifier durant ses déplacements. Cette technique permet d'obtenir une authentification de haut niveau pour les sites. Les applets Java adoptent désormais ce mode de fonctionnement. Ainsi si une applet est signée, elle est considérée comme un code de confiance et peut accéder à toutes les fonctionnalités de Java. Elle sera placée dans un bac à sable dans le cas contraire.
- Contrôle d'accès: Pour améliorer les deux précédentes techniques, on met en place une politique de contrôle d'accès plus complexe. On peut la voir comme un raffinement d'une politique de bac à sable générale vers une politique spécifique à chaque application ou classe d'agents. En fonction des agents, le site pourra autoriser l'accès à un ensemble précis de fonctionnalités. Le contrôle d'accès permet de mixer les deux premières techniques en offrant aux agents signés plus de fonctionnalité qu'un simple bac à sable sans pour autant accéder à toutes les fonctionnalités.
- Vérification du code : La vérification de code permet d'obtenir une garantie sur la sémantique d'un code à travers l'analyse de sa structure, ou de son comportement pour un agent, en fonction d'une politique de sécurité donnée. Les bacs à sable font déjà des vérifications rudimentaires en cours d'exécution, principalement pour garantir le typage des opérandes des instructions, mais cela est fortement coûteux. Une autre approche est de vérifier automatiquement le code, avant son lancement, en s'appuyant sur une preuve de conformité. Pour cela on peut utiliser l'approche PCC (Proof Carrying Code). Lors de la mise en route de l'agent, son créateur fournit un ensemble de preuves intégrées qui est transporté par l'agent. Ces preuves garantissant le comportement de l'agent en fonction de critères de sécurité des sites à visiter. Lorsque l'agent commence une nouvelle visite, le site récupère la preuve lui correspondant et vérifie si elle correspond à sa politique de sécurité. Le site choisit alors d'exécuter ou non l'agent. Cette technique se base pour l'instant sur des propriétés de typage et la preuve est fournie par le compilateur.

B. Protection des agents

À l'opposé de la protection des sites, la protection des agents contre des sites malveillants ne dispose pas de solution éprouvée et reste encore un champ de recherche ouvert.

Pour comprendre ce que risque un agent lors de son exécution sur un site malveillant, les éléments transportés sont tous cibles d'attaque :

- Le Code : Ensemble des instructions composant la tâche de l'agent.
- Les données statiques : Données ne changeant pas durant les déplacements (la signature par exemple)
- Les données collectées : Ensemble des résultats obtenus au cours des déplacements réalisés par l'agent depuis son lancement.
- L'état courant : Ensemble de données servant à l'exécution courante de l'agent. La sécurité des agents mobiles consiste alors à garantir les critères de confidentialité et d'intégrité de l'ensemble de ces éléments. Du point de vue des données, il est évident qu'un agent ne souhaite pas divulguer des informations critiques à n'importe quel site. Par exemple, un site malveillant pourrait récupérer la signature d'un code et l'utiliser pour créer un nouvel agent afin de s'introduire dans des environnements auxquels il n'a normalement pas accès. Pour le code, un agent transporte un savoir-faire propre à son concepteur qui pourrait tomber aux mains de ses concurrents.

  Les attaques peuvent être classées en trois catégories : l'inspection, la modification et le rejeu. L'inspection consiste à examiner le contenu de l'agent, ou le flot d'exécution afin de récupérer des informations critiques transportées par l'agent. La modification se réalise en remplaçant certains éléments de l'agent dans le but de conduire une attaque. En remplaçant le code, l'agent effectuera des opérations malveillantes sur les futurs sites à visiter. Le rejeu s'obtient en clonant l'agent puis en exécutant le clone dans plusieurs configurations pour retrouver le savoir de l'agent. Différentes techniques sont en cours d'étude pour garantir aux agents qu'ils peuvent accéder à des nœuds dans lesquels ils peuvent avoir toute confiance ou pour détecter les agents corrompus. Cependant, aucune d'elle n'apporte un niveau de sécurité suffisant comme celui proposé pour protéger les sites.

En conclusion, le domaine de sécurité lié à l'utilisation des agents mobiles, c'est qu'il s'agit encore d'un champ d'investigation complet. Bien que la protection des sites soit à présent maîtrisée, la protection des agents n'est, pour l'instant, toujours pas satisfaisante. Les différents efforts menés par la communauté de la sécurité nous poussent à croire que nous obtiendrons le niveau de sécurité requis.

1.4 Le Système d'Agent

1.4.1 Définition

Un système d'agent (aussi appelé serveur d'agent) est un environnement qui est capable de créer, interpréter, exécuter, transférer et arrêter un agent. De la même manière qu'un agent, un système d'agent est associé à une autorité qui identifie la personne ou l'organisation pour laquelle il agit. Un système d'agent est identifié par son nom et son adresse. Une machine hôte peut contenir plusieurs systèmes d'agent. Cinq concepts jouent un rôle important dans le système d'agent :

A. La place

Un agent mobile se déplace d'un environnement d'exécution à un autre. Cet environnement est appelé " place ". Une place est un contexte au sein d'un système d'agent, dans lequel un agent s'exécute. Ce contexte peut fournir un ensemble de services uniformes sur lesquels l'agent peut compter indépendamment de sa localisation spécifique.

La place de départ et la place de destination peuvent être situées au sein d'un même système d'agent ou sur des systèmes d'agents différents. Un système d'agent peut contenir une ou plusieurs places (**voir figure 1.3**).

B. **Région :**

Une région est un ensemble de systèmes d'agent qui appartiennent à la même autorité mais qui ne sont pas nécessairement de même type. Le concept de région permet à une autorité d'être représentée par plusieurs systèmes d'agent (**voir figure 1.3**).

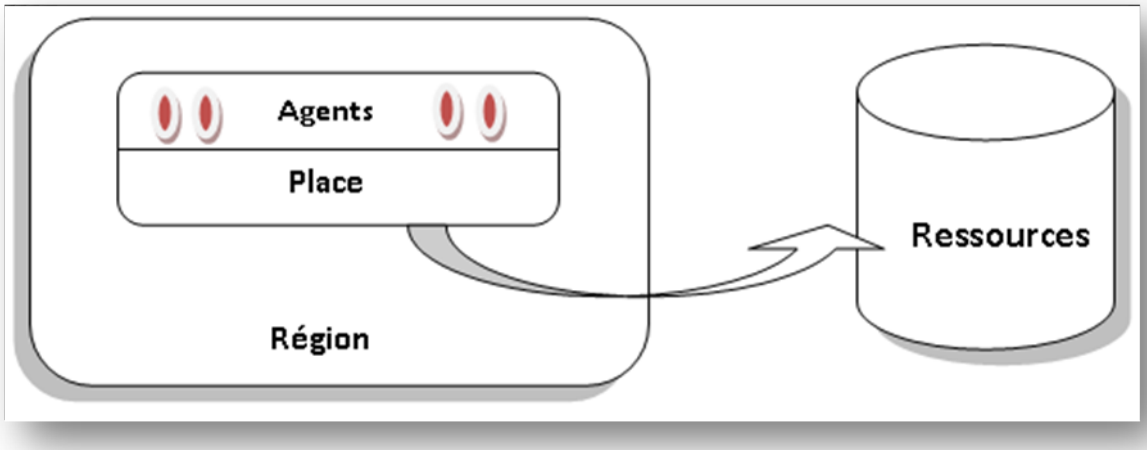

Figure 1.3 place et région

C. Les ressources :

Le système d'agent et la place fournissent un accès contrôlé aux ressources locales et aux services (base de données, processeurs, mémoire, disques).

D. Le type d'un système d'agent :

Le type d'un système d'agent permet de définir le profil d'un agent. Par exemple, si le type d'un système d'agent est " AGLET ", alors le système d'agent est implémenté par IBM et supporte Java comme langage de programmation.

E. Localisation :

La localisation est un concept important pour les agents mobiles. On définit la localisation d'un agent comme étant la combinaison de la place dans laquelle il s'exécute et l'adresse réseau du système d'agent où réside la place. Concrètement, elle est définie par l'adresse IP et le port d'écoute du système d'agent avec le nom de la place comme attribut.

1.4.2 Présentation des plateformes agents mobiles

Cette partie du chapitre décrit brièvement les fameuses plateformes **[B.L-M.O]** :

1.4.2.1 Concordia

Concordia qui est l'offre d'agents mobiles de Mitsubishi Electric Information Technology Center America est un environnement de développement, d'exécution et de gestion d'agents mobiles. Pour ce qui est des systèmes requis pour son fonctionnement, Concordia est compatible avec Windows95/98/NT et Solaris; il requiert, aussi,

Le JDK (Java Development Kit). Par ailleurs, Concordia est compatible avec tous les réseaux qui utilisent le protocole TCP/IP.

En fait, le système Concordia est composé d'un ensemble de classes Java permettant l'exécution de serveurs, le développement d'agents et leur activation. Chaque nœud du système représente un serveur Concordia qui s'exécute dans le cadre de la machine virtuelle Java.

1.4.2.2 Odyssey

Le successeur de Telescript, développé également par General Magic, s'appelle Odyssey. Différent du premier, ce dernier est basé sur Java. Il est largement inspiré par Telescript, mais n'a pas exactement la même fonctionnalité. Par exemple, Odyssey ne supporte pas le transfert de l'état d'exécution. Le système n'a toujours pas dépassé les versions *bêta*, et ne joue visiblement pas le rôle important que jouait Telescript dans la stratégie de General Magic. Répétons, toutefois, que Danny Lange a joint l'équipe.

1.4.2.3 Voyager

Voyager est un middleware développé par ObjectSpace en 1997. Voyager implémente en Java une plate-forme pour systèmes distribués. Cette plateforme inclut un ORB (Object Request Brocker) supportant les objets mobiles et les agents autonomes.

La seule condition pour pouvoir installer la plate-forme Voyager est l'existence de la version 1.1 ou la version 1.2 ou la version 1.3 du JDK et du JRE. Les agents Voyager sont autonomes et traités comme des objets ayant un cycle de vie de 24 heures.

Un agent Voyager est une classe qui hérite de la classe Agent de l'API Voyager et qui a la capacité d'être instanciée à distance. Lorsqu'un objet distant est construit, un GUID (Globally Unique IDentifier) de 16 octets lui est automatiquement assigné pour l'identifier de façon unique. Voyager fournit un service d'annuaire permettant de localiser et de se connecter ultérieurement à un tel objet. D'autant plus que Voyager est capable d'exploiter des composants JavaBeans.

La communication dans Voyager s'effectue par envoi de messages entre les agents.

Ces messages peuvent être de l'un des types suivants:

- Messages synchrones: c'est le type de message par défaut.

- Messages à sens unique: ce sont des messages envoyés une seule fois et qui n'ont pas de valeur de retour.
- Messages Asynchrones (futures messages) : la transmission non bloquante ou asynchrone de messages permet aux agents de poursuivre leur exécution jusqu'à l'arrivée de la réponse au message asynchrone envoyé. En d'autres termes, ce type de message permet de ne pas interrompre l'exécution de l'appelant (l'émetteur du message) en attendant la réponse à sa requête.
- Messages multicast: permettant d'envoyer un message à plusieurs objets Voyager à la fois.

L'architecture de la plate-forme Voyager est distribuée.

1.4.2.4 AgentTCL

AgentTCL (Une version complète 2.0 datée du 11 mars 98 est disponible) est l'un des premiers projets d'agents mobiles développés par l'université Dartmouth. AgentTCL a été basé sur le système Tcl (Tool Command Language) et plus tard a été étendu pour supporter Java, Scheme et C/C++.

Agent Tcl est constitué de deux parties :

– Un serveur chargé du management, du suivi des agents et de la sécurité. En effet, le serveur AgentTCL reçoit puis authentifie et démarre les agents, il stocke les messages, fournit un service de désignation local et maintient une table d'information sur les agents présents sur le site.

- Un interpréteur TCL (une version modifiée de l'intérpréteur TCL 7.5) qui fournit un mécanisme de sauvegarde et de restauration de l'état de l'excécution d'un script TCL. Il propose des commandes pour la migration, la communication et la création d'un agent.

AgentTCL définit un agent comme un programme identifié qui peut se déplacer de machine en machine dans un réseau hétérogène en choisissant le lieu et le moment de ses déplacements.

Les agents se déplacent d'une machine à l'autre en utilisant la commande *Jump*. Le système capture l'état d'exécution d'un agent puis l'envoi au serveur de la machine cible qui restaure l'agent. Chaque fois qu'un agent est envoyé vers un serveur (création ou migration) il est crypté et signé digitalement pour éviter son altération du-

rant le transfert. Le processus de migration de l'agent n'obéit à aucune règle particulière sauf celles relatives à la sécurité de l'agent.

Il est à ajouter que le système AgentTCL fournit un service de messages classiques proposant la communication asynchrone avec les commandes *agent_send* et a*gent_receive*.

Il propose également l'établissement de rendez-vous synchrone entre deux agents avec les commandes *agent_meet* et *agent_accept* permettant de réaliser ensuite des communications directes sur une connexion identifiée (TCP).

1.4.2.5 TACOMA

Le projet TACOMA (Tromsô And COrnell Moving Agents) fournir un support dans les systèmes d'exploitation pour la programmation d'agents mobiles. Le modèle ne fait pas de distinction entre un client, un serveur et un agent. Il propose simplement la notion d'agents et des mécanismes de migration et de communication. Un agent TACOMA est un processus exécutant un script Tcl qui est capable de se déplacer d'une manière autonome d'une machine à une autre. Un agent peut manipuler les données locales et transporter des données lors de ses déplacements. TACOMA introduit pour cela la notion de *Folder* qui correspond à une unité de donnée manipulée par un agent. Chaque Folder est identifié par un nom et contient une valeur. TACOMA propose un mécanisme de communication fondé sur le concept de rendez-vous par la commande *meet*. Une communication est toujours synchrone. Elle est réalisée par l'intermédiaire d'un *Briefcase* échangé entre les deux agents. Un *briefcase* est un objet qui contient un ensemble de *Folders.*

La communication avec un agent distant est réalisée avec la même abstraction de Rendez-vous par l'intermédiaire d'un agent système de transport nommé *rexec*. Pour cela, il faut construire un *Briefcase* spécifique qui doit contenir les *Folders HOST* pour désigner le site cible et *CONTACT* pour spécifier l'agent distant.

La migration dans TACOMA est réalisée par un rendez-vous avec un agent distant spécial nommé *ag_tcl* dont la seule fonction est d'instancier un nouvel agent à partir d'un *Folder CODE* qui contient le code Tcl de réactivation de l'agent.

1.4.2.6 JADE

JADE (Java Agent DEvelopment Framework), apanage du CSELT (Centre Studi E Laboratori Telecomunicazioni) de l'université de Parma. Cet environnement de

développement d'applications à base d'agents, JADE, est implémenté entièrement en Java.

JADE est conforme aux normes et spécifications de FIPA (Foundation For Intelligent Physical Agents), elle hérite alors son architecture, ses protocoles, ses services et ses mécanismes de transport et de communication.

JADE est compatible avec la plupart des configurations matérielles (P.C., Macos) et logiciels (Windows, Unix). Le seul système nécessaire pour son fonctionnement est la version 1.2 de Java (JRE ou JDK)

L'agent JADE peut être implémenté en Java ou en Jess (Java Expert System Shell). Ce dernier est un système expert qui permet d'implémenter des architectures d'agents à base de règles.

Les agents JADE communiquent à travers le langage ACL (Agent Communication Language) de FIPA, dans le cas où ces agents sont hétérogènes, ils coopèrent par négociation. En général, la négociation est utilisée comme un mécanisme pour coordonner les actions d'un groupe d'agents lors de la résolution de problèmes **[jade]**.

1.4.2.7 MAP

MAP (Mobile Assistant Programming) a été publié en 1995 pour l'accès nomade au réseau d'information. En effet, le système MAP présente un modèle d'agents mobiles pour la programmation et l'exécution efficace d'actions par délégation dans un réseau de grande envergure pour l'accès à l'information depuis des stations nomades.

Le système MAP utilise le support du *World-Wide Web* et le langage *Scheme* pour l'implantation du modèle d'agent mobile, le premier prototype du système MAP a été réalisé en 1996.

MAP permet la programmation d'agents mobiles appelés assistants qui peuvent se déplacer d'un site à un autre, instancier des clones et envoyer des résultats. L'exécution des assistants est persistante et les opérations de déplacement, de clonage et d'envoi de résultats ont une sémantique d'actions atomiques.

Le système MAP utilise le WWW comme support de communication à travers le protocole HTTP et le mécanisme d'extension CGI (Common Gateway Interface: utilisé pour traiter des bases de données via le web) comme support d'exécution distribué.

Chaque nœud qui participe au réseau des assistants comporte un serveur HTTP auquel on ajoute un programme CGI nommé MapServer. La communication entre les nœuds est assurée par la fonction interne *Post* qui permet de transférer automatiquement des données avec authentification. Cette fonction est implantée en utilisant la méthode *POST* du protocole HTTP. La cible de ce postage est le programme CGI MapServer qui doit réaliser l'authentification et prendre en compte les ruptures de communication.

1.4.2.8 JATLite

JATLite (Java Agent Template, Lite) est une palte-forme agents mobiles implémentée en Java. Elle est compatible avec Windows 95, Windows NT, Soloris, UNIX, et elle nécessite le Jdk1.1 ainsi que des navigateurs Internet (Internet Explorer, Netscape).

Les agents mobiles JATLite communiquent à travers Internet via le langage KQML et les standards Internet TCP/IP, SMTP et FTP. Leurs communications peuvent être synchrone ou asynchrone.

L'architecture de la plate-forme JATLite est organisée en une hiérarchie de couches spécifiques, ainsi l'utilisateur peut sélectionner la couche appropriée à partir de laquelle il commence la construction du système.

1.4.2.9 Grasshopper

La plate-forme Grasshopper développée par IKV++ (Innovation Know-how Vision++) en 1997 est un environnement d'implémentation et d'exécution pour agents mobiles.

Ce produit développé en Java a été testé sur SUN Solaris 2.5 et 2.6, et sur Windows (NT et 9x). Pour son fonctionnement, Grasshopper requiert au minimum la version Java 1.1.4Visibroker 3.1. Grasshopper a la particularité d'être la première plateforme compatible avec le protocole standard MASIF (Mobile Agent System Interoperability Facility) de l'OMG (Object Managment Group) et les standards de FIPA (Foundation For Intelligent Physical Agents). Grasshopper supporte plusieurs protocoles de transports et de communication des agents tels que TCP/IP, JavaRMI, CORBAIIOP, MAFIIOP, TCP/IPSSL, RMISSL).

1.4.2.10 Aglet

Le système d'IBM Aglet Workbench propose un environnement de programmation d'agents mobiles dans le langage Java qui est issu du modèle d'agents itinérants de 'Chess'.

Aglet est compatible avec la configuration LINUX, X86/Solaris et aussi avec celle de Windows.

Un Aglet (AGent appLET) est un objet mobile qui dispose de son propre thread de contrôle, auquel on peut envoyer des événements et qui a la possibilité de communiquer par l'intermédiaire de messages.

1.5 Conclusion

Dans ce chapitre on a essayé de présenter une introduction dans le paradigme d'agent mobile, en commençant par une introduction sur les agents dans un cadre général, ensuite par une brève représentation de paradigme d'agent mobile et finalement une brève présentation des quelque plateformes d'implémentation des applications d'agent mobiles.

En conséquence, Les applications d'agent mobile sont prouvé leur apparence car les motivations suivantes: La première motivation est généralement la minimisation des communications distantes (il est en général moins coûteux de migrer le code de traitement que les données à traiter, qui peuvent être beaucoup plus volumineuses). Une autre motivation est le cas de l'informatique nomade (l'agent mobile peut continuer ses traitements sur des serveurs pendant la déconnexion de la machine cliente). Les problèmes principaux actuels sont liés à la sécurité et à l'interopérabilité des agents mobiles sur les différentes plates-formes d'accueil.

Le domaine de recherche sur les agents mobiles est relativement récent, les travaux de recherche sur ce domaine est très nombreux et sont apparut récemment en essayant de dégager les concepts importants.

Le chapitre suivant va introduire les approches et les méthodes de modélisation des applications d'agent mobile et notamment la notion d'UML mobile.

Chapitre 02: La modélisation UML

2.1 Introduction

Un modèle est un vecteur privilégié pour communiquer entre le client, commandant un système logiciel, et le fournisseur, établissant et fournissant un système logiciel, c'est un langage commun, précis, qui est connu par tous les membres de l'équipe.

Les méthodes structurées ont été proposées en tant que méthodes de développement de systèmes en premier lieu. Ces méthodes offrent les diagrammes entité-relation pour modéliser l'aspect statique (données) d'un système et les diagrammes de flot de données ou les techniques de décomposition fonctionnelle pour la modélisation de l'aspect dynamique (comportement) d'un système. La taille des applications ne cessant de croître et la programmation structurée a également rencontré ses limites, faisant alors place à la programmation orientée objet (Simula 67 en 1967, Smalltalk en 1976, C++ en 1982, Java en 1995, …). La technologie objet est donc la conséquence ultime de la modularisation dictée par la maîtrise de la conception et de la maintenance d'applications toujours plus complexes. Cette nouvelle technique de programmation a nécessité la conception de nouvelles méthodes de modélisation.

Dans les années 90, une guerre des méthodes de développement orienté-objet a été déclenchée, où une multitude de méthodes a été proposée. La fin de cette guerre a donné naissance à un nouveau standard, un langage de modélisation orienté-objet qui est l'UML (Unified Modeling Langage). Aujourd'hui, le langage de modélisation unifié UML a été largement accepté par l'industrie et s'est établi comme le langage commun pour l'analyse et la conception en génie logiciel orienté objet.

Les applications d'agent mobile sont plus en plus prennent sa place dans le monde informatique sur les dernières années et pour la particularité des ces méthodes des nombreux approches et méthodes sont apparu pour les modéliser, d'après **[Adl]** il existe trois approches de modélisations d'agent mobile : l'approche pattern **[Arido]**, l'approche formelle **[Picco]** et l'approche semi formelle **[Lokos]**. Dans ce travail nous avons concerné par l'approche semi formelle et plus précisément le langage UML et ses extensions.

Dans ce chapitre, on présente le langage de modélisation UML et son extension UML mobile en commençant par l'introduction de la notion de modélisation, suivie par une brève introduction au langage UML, ensuite, la notation UML et enfin une représentation précise des extensions d'UML pour modéliser les applications d'agent mobile.

2.2 La modélisation

2.2.1 Qu'est ce qu'un modèle?

La modélisation est le processus de produire un modèle [Anu97]. Selon *Marvin L. Minsky* [Min68], pour un observateur B, un objet A * est un modèle d'un objet A dans la mesure où B peut employer A * pour répondre aux questions qui l'intéressent concernant A.

Un modèle est une abstraction d'un système construite dans un but précis. On dit alors que le modèle représente le système. [Mil] 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.

2.2.2 Pourquoi modéliser?

Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence. [Aud07] Un modèle est un langage commun, précis, qui est connu par tous les membres de l'équipe et il est donc, à ce titre, un vecteur privilégié pour communiquer. Cette communication est essentielle pour aboutir à une compréhension commune aux différentes parties prenantes (notamment entre la maîtrise d'ouvrage et la maîtrise d'œuvre informatique) et précise d'un problème donné.

Dans le domaine de l'ingénierie du logiciel, le modèle permet de mieux répartir les tâches et d'automatiser certaines d'entre elles. C'est également un facteur de réduction des coûts et des délais. Par exemple, les plateformes de modélisation savent maintenant exploiter les modèles pour faire de la génération de code (au moins au niveau du squelette) voire des allers-retours entre le code et le modèle sans perte d'information. Le modèle est enfin indispensable pour assurer un bon niveau de qualité et une maintenance efficace. En effet, une fois mise en production, l'application va devoir être maintenue, probablement par une autre équipe et, qui plus est, pas nécessairement de la même société que celle ayant créée l'application.

Le choix du modèle a donc une influence capitale sur les solutions obtenues. Les systèmes non-triviaux sont mieux modélisés par un ensemble de modèles indépendants. Selon les modèles employés, la démarche de modélisation n'est pas la même.

2.2.3 Différentes Formes de Modélisation

On peut classer la modélisation comme étant formelle, semi-formelle ou informelle. La **table 2.1 [José]** présente une définition des catégories de langages ainsi que des exemples de langages ou de méthodes qui les utilisent.

<table>
<tr><th colspan="4">Catégories de Langages</th></tr>
<tr><td colspan="2">Langage Informel</td><td rowspan="2">Langage Semi-Formel</td><td rowspan="2">Langage Formel</td></tr>
<tr><td>Simple</td><td>Standardisé</td></tr>
<tr><td>Langage qui n’a pas un ensemble complet de règles pou restreindre une construction</td><td>Langage avec une structure, un format et des règles pour la composition d’une construction.</td><td>Langage qui a une syntaxe définie pour spécifier les conditions sur lesquelles les constructions sont permises.</td><td>Langage qui possède une syntaxe et une sémantique définies rigoureusement. Il existe un modèle théorique qui peut être utilisé pour valider une construction.</td></tr>
<tr><th colspan="4">Exemples de Langages ou Méthodes</th></tr>
<tr><td>Langage Naturel.</td><td>Texte Structuré en Langage Naturel.</td><td>Diagramme Entité-Relation, Diagramme à Objets.</td><td>Réseaux de Petri, Machines à états finis, VDM, Z.</td></tr>
</table>

Table 2.1 - Classification et Utilisation de Langages ou de Méthodes **[José]**

2.2.3.1 Modélisation Informelle

Le processus de modélisation informelle selon un langage informel peut avoir son emploi justifié par plusieurs raisons **[José]**: (1) par sa facilité de compréhension, elle permet des consensus entre des personnes qui spécifient et celles qui commandent un logiciel ; (2) elle représente une manière familière de communication entre personnes. D’un autre côté, l’utilisation d’un langage informel rend la modélisation imprécisé et parfois ambigüe. De plus, comme le raisonnement humain est le facteur principal pour l’analyse et la vérification de la spécification, il peut conduire à des erreurs de compréhension, d’interprétation et de vérification. Il est possible d’utiliser une *Modélisations Informelle Standardisée* pour restreindre ces problèmes ; 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.

2.2.3.2 Modélisation Semi-Formelle

Le processus de modélisation semi-formelle est basée sur un langage textuel ou graphique pour lequel une syntaxe précise est définie ainsi qu'une sémantique ; cette sémantique est assez faible mais permet néanmoins une certaine dose de contrôle et l'automatisation de quelques tâches **[José]**. La plupart des propositions semi-formelles s'appuient fortement sur un langage graphique. Cela peut se justifier par l'expressivité que peut avoir un modèle graphique bien développé ; le langage textuel est utilisé normalement comme appui aux modèles graphiques. La modélisation semi-formelle en utilisant un langage graphique peut produire des modèles de compréhension facile et qui peuvent être très ponctuels ; cela justifie largement son utilisation. Cependant le manque de sémantique complète est un fort handicap pour ce genre de modélisation ; le problème existant pour les langages informels, de manque de précision par rapport à la compréhension de la modélisation ainsi que le problème d'ambigüité persiste pour les langages semi-formels. Pour essayer de résoudre ce problème de manque de précision et d'ambigüité, l'utilisation de contraintes dans les modèles graphiques a été introduite. Un exemple d'utilisation de contraintes textuelles sur un modèle graphique concerne le travail de J. J. Odell **[Odel]**. Dans ce travail, des contraintes structurelles sont appliquées à une modélisation basée sur des modèles orientés objets. Une autre proposition est celle de S. Cook et J. Daniels **[Cook]**. Dans ce travail, sur un modèle graphique basé sur la méthode OMT **[Rum]** et les Diagrammes d'Etats, le langage formel Z **[Spy]** est utilisé pour représenter des contraintes.

2.2.3.3 Modélisation Formelle

Une méthode formelle est un processus de développement rigoureux basé sur des notations formelles avec une sémantique précise ainsi que sur des vérifications formelles. Le principal avantage des spécifications formelles est leur capacité à exprimer une signification précise, en permettant de cette manière des vérifications de la cohérence et de la complétude d'un système. Nous pouvons résumer **[José]** brièvement les analyses des méthodes formelles réalisées par A. Hall **[Hall]** et par J. P. Bowen et M. C. Hinchey **[Hin]** qui concernent plus particulièrement les mythes inhérents aux langages formels.

A. Hall **[Hall]** fait ressortir que les méthodes formelles aident à trouver des erreurs et à diminuer leur incidence à travers une spécification complète qui peut être appliquée à n'importe quel type de système, logiciel ou matériel ; elles ne remplacent cependant pas des méthodes existantes et doivent être utilisées d'une manière

conjointe avec celles-ci. Si une connaissance mathématique solide est nécessaire pour réaliser des preuves, elle ne l'est pas impérativement pour spécifier. L'utilisation de méthodes formelles peut provoquer une diminution des coûts et elle n'est pas à l' origine des retards dans les projets.

J. P. Bowen et M. C. Hinchey **[Hin]** montrent qu'avec une traduction appropriée, les méthodes formelles peuvent aider dans la compréhension d'un système par un utilisateur y compris dans le cas de gros systèmes où elles sont aussi applicables. En plus des outils de preuves déjà existants, les études pour la création d'autres outils qui couvrent toute la spécification, montrent l'intérêt de la communauté pour ces méthodes. L'utilisation des méthodes formelles, plus que souhaitable, commence à être imposée par des sociétés dans certains cas, même si elles ne sont pas utilisées dans tous les types de modélisation.

2.2.4 La modélisation orientée objet

Le concept fondamental dans la modélisation et la conception orientées objet est l'objet, qui combine à la fois une structure de données et un comportement, cet objet est organisés autour de concepts du monde réel.

Les modèles orientés objet permettent de comprendre des problèmes, de communiquer avec des experts du domaine d'application, de modéliser le métier d'une entreprise, de réparer la documentation et de concevoir des programmes et des bases de données. [BR05]

2.2.4.1 qu'est ce l'orienté objet

Le terme **orienté objet** (**OO**) signifie qu'on organise un logiciel formé d'une collection d'objets indépendants qui incorporent à la fois une structure de données et un comportement [BR05]. Les structure de données et le comportement sont faiblement associent dan cette approche comme les autres approches de programmation. D'une manière générale les caractéristiques requises par une approche orientée objet sont: l'*identité*, la *classification*, l'*héritage* et le *polymorphisme*.

2.2.4.2 Pourquoi l'approche objet

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 [MG00], ainsi que la simplicité du modèle qui fait appel à cinq concepts fondateurs (les objets, les messages, les classes, la généralisation et le polymorphisme) sont des avantages de l'approche objet.

L'approche orientée objet est largement adoptée, tout simplement parce qu'elle a démontré son efficacité lors de la construction de systèmes dans une diversité de domaine métier et qu'elle englobe toutes les dimensions et tout les degrés de complexité.

2.3 Introduction à UML

UML (**Unified Modeling Language**) est un langage pour visualiser, spécifier, construire et documenter tous les aspects et artefacts d'un système logiciel [OMG03a]. Dans la pratique, UML concerne tout système construit en utilisant l'approche orienté objet (OO).

Comme son nom l'indique, UML est le résultat de la fusion d'un ensemble d'autres méthodes de modélisations orientées objets. Il est devenu un standard de facto. Un des ingrédients de son succès est le fait d'être un langage générique. L'ensemble des concepts et vues composant la sémantique d'UML peut s'adapter à tous les domaines et problèmes de conception.

L'évolution du langage UML est contrôlée par l'**OMG** (**O**bject **M**anagement **G**roup) [OMG], L'OMG est une association américaine à but non-lucratif créée en 1989 dont l'objectif est de standardiser et promouvoir le modèle objet sous toutes ses formes. L'OMG est notamment à la base des spécifications UML, MOF, CORBA et IDL. L'OMG est aussi à l'origine de la recommandation MDA.

2.3.1 UML est un langage de modélisation

Toutefois, il ne faut pas oublier que la notation UML est un langage de modélisation objet et non pas une méthode **(**Une description normative des étapes de la modélisation**)** objet. La notation UML est conçue pour servir de langage de modélisation objet indépendamment de la méthode suivie, il ne constitue qu'une partie d'une méthode de développement logiciel. Ses auteurs ont en effet estimé qu'il n'était pas opportun de définir une méthode en raison de la diversité des cas particuliers. Ils ont préféré se borner à définir un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système.

Un langage comprend un vocabulaire et les règles qui permettent de combiner les mots de ce vocabulaire afin de communiquer. Un langage de modélisation est un langage dont le vocabulaire et les règles sont centrées sur la représentation conceptuelle et physique d'un système [BRJ01].

UML est un langage de modélisation objet [MG00]. En tant que tel, il facilite l'expression et la communication de modèles en fournissant un ensemble de symboles (la notation) et de règles qui régissent l'assemblage de ces symboles (la syntaxe et la sémantique).

2.3.2 L'historique d'UML

Les méthodes utilisées dans les années 1980 pour organiser la programmation impérative (notamment Merise) étaient fondées sur la modélisation séparée des données et des traitements. Lorsque la programmation par objets prend de l'importance au début des années 1990, la nécessité d'une méthode qui lui soit adaptée devient évidente. Plus de cinquante méthodes apparaissent entre 1990 et 1995 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE, etc.) mais aucune ne parvient à s'imposer. En 1994, le consensus se fait autour de trois méthodes:

- OMT de James Rumbaugh (*General Electric*) fournit une représentation graphique des aspects statique, dynamique et fonctionnel d'un système ;
- OOD de Grady Booch, définie pour le *Department of Defense*, introduit le concept de paquetage (*package*) ;
- OOSE d'Ivar Jacobson (Ericsson) fonde l'analyse sur la description des besoins des utilisateurs (cas d'utilisation, ou *use cases*).

Chaque méthode avait ses avantages et ses partisans. Le nombre de méthodes en compétition s'était réduit, mais le risque d'un éclatement subsistait : la profession pouvait se diviser entre ces trois méthodes, créant autant de continents intellectuels qui auraient du mal à communiquer.

Événement considérable et presque miraculeux, les trois gourous qui régnaient chacun sur l'une des trois méthodes se mirent d'accord pour définir une méthode commune qui fédérerait leurs apports respectifs (on les surnomme depuis « the Amigos »). UML (*Unified Modeling Language*) est né de cet effort de convergence. L'adjectif *unified* est là pour marquer qu'UML unifie, et donc remplace.

En fait, et comme son nom l'indique, UML n'a pas l'ambition d'être exactement une méthode : c'est un langage.

L'unification a progressé par étapes. En 1995, Booch et Rumbaugh (et quelques autres) se sont mis d'accord pour construire une méthode unifiée, *Unified Method 0.8* ; en 1996, Jacobson les a rejoints pour produire UML 0.9 (notez le remplacement du

mot *méthode* par le mot *langage*, plus modeste). Les acteurs les plus importants dans le monde du logiciel s'associent alors à l'effort (IBM, Microsoft, Oracle, DEC, HP, Rational, Unisys etc.) et UML 1.0 est soumis à l'OMG [OMG]. L'OMG adopte en novembre 1997 UML 1.1 comme langage de modélisation des systèmes d'information à objets. La version d'UML en cours en 2008 est UML 2.1.1 et les travaux d'amélioration se poursuivent.

UML est donc non seulement un outil intéressant mais une norme qui s'impose en technologie à objets et à laquelle se sont rangés tous les grands acteurs du domaine, acteurs qui ont d'ailleurs contribué à son élaboration.

2.3.4 Les déférentes vues d'un système

Les vues définissent le système, ce sont des formulations du problème selon un certain point de vue et elles peuvent se chevaucher pour compléter une description. Il existe cinq types de vues pour le quel leur somme représente le modèle en entier.

La **figure 2.1** présente différentes vues, indépendantes et complémentaires, permettant de définir l'architecture d'un système [BRJ01], chaque vue est une projection, selon un aspect particulier, dans l'organisation et la structure du système.

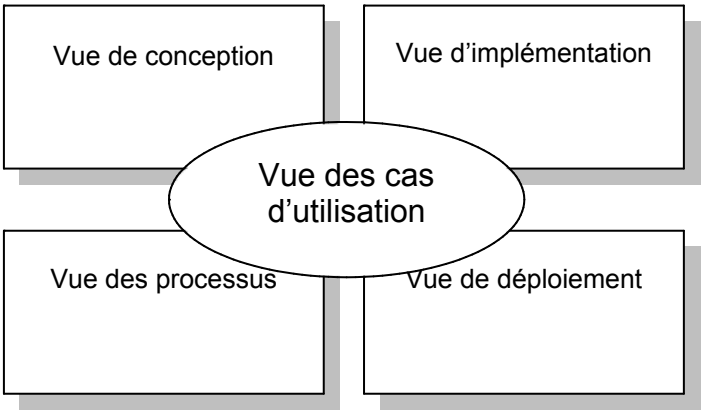

Figure2.1 : modélisation de l'architecture d'un système. **[BRJ01]**

Les différentes vues sont :

Vue des cas d'utilisation (qui, quoi) : Description du système vu par les acteurs du système.

Vue logique (de conception) : elle modélise les éléments et les mécanismes principaux du système. Les éléments UML impliqués sont les classes, les interfaces, etc.

Vue d'implémentation : elle liste les différentes ressources des projets, fichiers binaires, bibliothèques, bases de données, etc. Elle établie le lien entre les composants et elle permet d'établir des dépendances et de ranger les composants en modules.

Vue de déploiement : Dans le cas de système distribué, elle définie les composants présents sur chaque nœud du système. C'est la vue spatiale du projet.

Vue des processus : c'est la vue temporelle et technique, qui manipule les notions de tâches concurrentes, contrôle, synchronisation, processus, threads, etc.

2.3.5 La méthodologie UML

Pour tirer avantage d'UML et pour modéliser un système selon les déférentes vues mentionnés précédemment plusieurs processus prennent ses places dans le monde de la modélisation comme Catalysis, OPEN et le processus RUP (**Rational Unified Process**).

RUP **[Kim]** est un processus d'ingénierie de logiciel, il est crié par les mêmes gents qui sont crié UML. Il est très largement utilisé dans le développement de différents types de logiciels y compris les logiciels temps réel et embarqués. Il présente une approche d'affectation de tâches et de responsabilités dans une organisation de développement. Son but est d'assurer la production de logiciels de haute qualité satisfaisant les besoins des utilisateurs dans des délais et avec des coûts prédictibles.

Piloté par les cas d'utilisation, RUP est un processus de développement de logiciel orienté objet qui reprend les principes du Processus unifié. Il présente des lignes directrices, schémas (templates), et exemples à tous les aspects et phases de développement. RUP et les autres processus similaires sont des outils d'ingénierie de logiciel combinant différentes phases, techniques, et pratiques avec d'autres composants de développement (tels que documents, modèles, manuels, codes, etc.) dans un framework unifié.

Il prend en compte l'ensemble des intervenants: client, utilisateur, gestionnaire, contrôleur de qualité, etc. Pour comprendre cette démarche, il est donc nécessaire de sortir du cadre strict du développement, au sens technique du terme. Se mettre à la place du client est un moyen d'y parvenir. Ainsi, les phases apparaissent naturellement comme des étapes du projet et non plus comme des activités techniques (analyse, conception,...).

La figure 2.2 montre que RUP gère le processus de développement selon deux axes. L'axe vertical représente les principaux enchaînements d'activités, qui regroupent les activités selon leur nature. Cette dimension rend compte l'aspect statique du

processus qui s'exprime en termes de composants, de processus, d'activités, d'enchaînements, d'artefacts et de travailleurs. L'axe horizontal représente le temps et montre le déroulement du cycle de vie du processus; cette dimension prend en compte l'aspect dynamique du processus qui s'exprime en terme de cycles, de phases, d'itérations et de jalons.

RUP définit quatre phases de développement. Chaque phase est organisée en itérations qui doivent satisfaire des critères définis avant de passer à la phase suivante. La première phase est l'amorçage (inception) où le développeur délimite le projet et les frontières des ses cas métiers. Après quoi, dans la phase d'élaboration les développeurs analyse les besoins du projet en détail et définissent ses fondements architecturaux. Lors de la phase de construction, les développeurs créent la conception de l'application et le code source. Finalement, à la phase de transition, les développeurs délivrent le système aux utilisateurs (ou clients). RUP permet de produire un prototype fonctionnel à l'accomplissement de chaque itération.

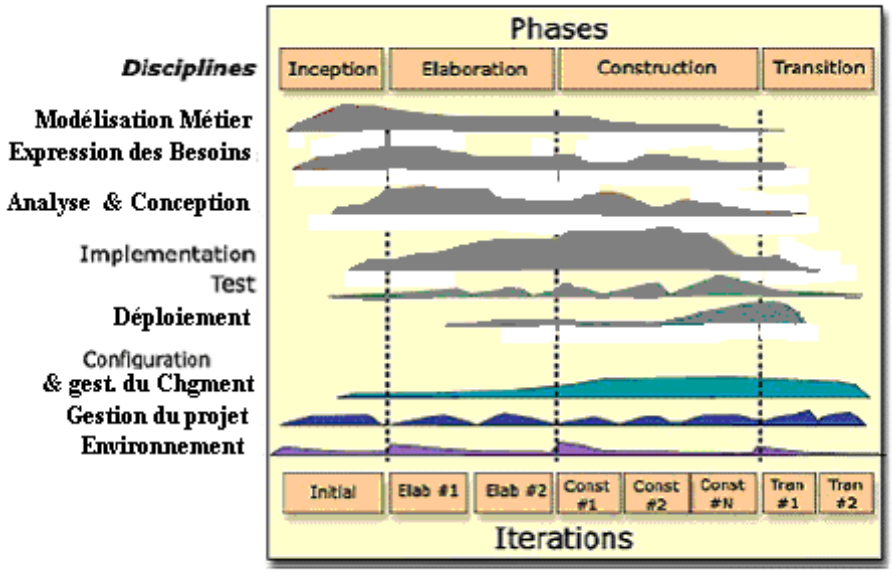

Figure 2.2: Schéma fonctionnel du RUP

Dans RUP, il y a quatre principaux éléments de modélisation: les rôles, les activités, les artefacts et les workflows. Les rôles représentent les comportements et les responsabilités des membres de l'équipe de développement (i.e. analystes système, architectes, concepteurs). Les activités sont les travaux effectués par les rôles (i.e. détermination des cas d'utilisation et acteurs, revue de la conception et exécution des tests de performance). Les artefacts sont les éléments produits ou utilisés par les rôles afin

d'effectuer les activités (i.e. document de l'architecture logicielle, le modèle de conception).

Les workflows (ou disciplines) décrivent la séquence des activités qui produisent un certain résultat et montrent les interactions entre les rôles. Il y a neuf workflows dans RUP: *Modélisation Métier*, *Expression des besoins*, *Analyse et conception*, *Implémentation*, *Test*, *Déploiement, Gestion de la configuration et des modifications*, *Gestion du projet*, et *Environnement*.

2.3.6 La notation UML

Les **éléments**, les **relations** et les **diagrammes** sont des briques de base du langage UML, ils sont considérés comme des éléments fondateurs des modèles de ce langage. **[BRJ01]**

2.3.6.1 Les éléments d'UML

Les éléments sont des abstractions essentielles à un modèle et servent à le concevoir correctement, Il existe quatre types d'éléments dans UML :

2.3.6.1.1 Les éléments structurels

Ils sont représentés par des noms dans les modèles UML, les éléments structurels représentent des éléments conceptuels ou physiques et ils sont comme des parties les plus statiques d'un modèle UML. On peut citer sept types d'éléments structurels:

- **Les classes**: une classe représente un ensemble d'éléments qui partagent les mêmes attributs, les mêmes opérations, les mêmes relations et les mêmes sémantiques. Une classe implémente une ou plusieurs interfaces. (**figure 2.3 (a)**)

- **Les interfaces**: l'interface est utilisée pour décrire le comportement apparent d'une classe ou d'un composant. (**figure 2.3 (b)**)

- **Les collaborations**: une collaboration possède une dimension structurelle et comportementale, donc elle représente l'implémentation des parties qui structurent un système. Une collaboration présente une unité supérieure à la somme de toutes ses parties et elle constitue une société de rôle et de divers éléments qui travaillent ensemble pour fournir un comportement coopératif. (**figure 2.3 (c)**)

- **Les cas d'utilisations**: Un cas d'utilisation capture le comportement d'un système, d'un sous-système, d'une classe ou d'un composant tel qu'un utilisateur extérieur le voit. Il scinde la fonctionnalité du système en unités cohérentes,

les cas d'utilisation, ayant un sens pour les acteurs. Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système, ils sont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique. (**figure 2.3 (d)**)

- **Les classes actives**: Une classe est passive par défaut, elle sauvegarde les données et offre des services aux autres. Une classe active initie et contrôle le flux d'activités. Une classe active est une classe dont le comportement de ses objets représente une concurrence avec d'autres éléments. (**figure 2.3 (e)**)

- **Les composants**: Un composant doit fournir un service bien précis. Les fonctionnalités qu'il encapsule doivent être cohérentes entre elles et génériques (par opposition à spécialisées) puisque sa vocation est d'être réutilisable. Un composant comporte une ou plusieurs interfaces requises ou offertes. Son comportement interne, généralement réalisé par un ensemble de classes, est totalement masqué : seules ses interfaces sont visibles. La seule contrainte pour pouvoir substituer un composant par un autre est de respecter les interfaces requises et offertes. (**figure 2.3 (f)**)

- **Les nœuds**: Dans la phase d'exécution les nœuds représentent les ressources matérielles qui composent le système. ils représentent une source de calcule et dispose généralement, au moins, d'un peut de mémoire et souvent d'une capacité de traitement. (**figure 2.3 (g)**)

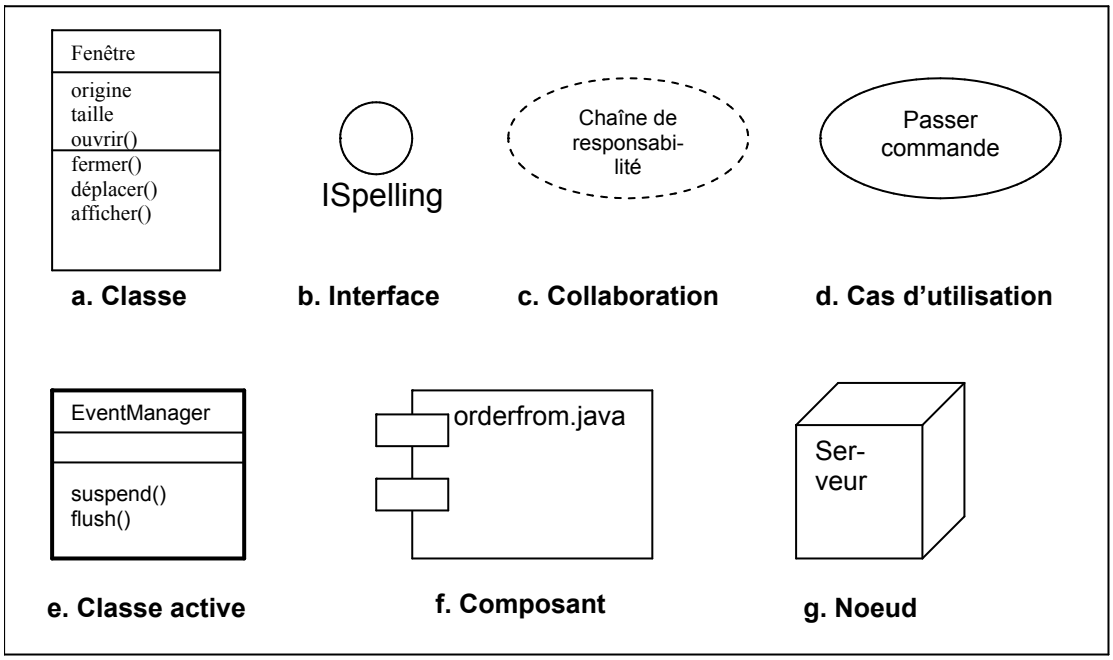

Figure 2.3 : les éléments structurels d'UML.

2.3.6.1.2 Les éléments comportementaux

Ils représentent le comportement des modèles UML dans l'espace et dans le temps.

- **Les interactions**: une interaction représente un comportement d'un ensemble de *messages* (**Figure 2.4 (a)**) échangés au sein d'un groupe d'éléments, dans un contexte particulier, pour atteindre un but bien défini.

- **Les automates à états finis:** un automate à états finis représente le comportement interne d'un élément. Ils présentent les séquences possibles d'états (**figure 2.4 (b)**) et d'actions qu'un élément peut traiter au cours de son cycle de vie en réaction à des événements.

Figure 2.4 : Les éléments comportementaux

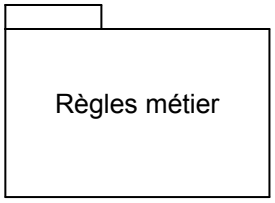

Figure 2.5 : Paquetage

2.3.6.1.3 Les éléments de regroupement

Il existe un seul type d'éléments de regroupement : les *paquetages*

Un **paquetage** (**figure 2.5**) est un regroupement d'éléments de modèle et de diagrammes. Il permet ainsi d'organiser des éléments de modélisation en groupes. Il peut contenir tout type d'élément de modèle : des classes, des cas d'utilisation, des interfaces, des diagrammes, … et même des paquetages imbriqués (décomposition hiérarchique).

2.3.6.1.4 Les éléments d'annotation

Il existe un type fondamental d'éléments d'annotation appelé *note*.

Une **note** (**figure 2.6**) contient une information textuelle comme un commentaire, un corps de méthode ou une contrainte.

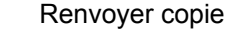

Figure 2.6 : Note

2.3.6.2 Les relations dans UML

Les relations constituent les liens entre les éléments Il existe quatre types de relations dans UML :

- **Les dépendances:** une dépendance est une relation unidirectionnelle exprimant une dépendance sémantique entre des éléments du modèle. Elle

indique que la modification de la cible peut impliquer une modification de la source (**figure 2.7 (a)**).

- **Les associations:** 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 (**figure 2.7 (b)**).

- **Les généralisations:** la généralisation décrit une relation entre un élément générale (élément de base ou élément parent) et un élément spécialisé (sous-élément). L'élément spécialisé est intégralement cohérent avec l'élément de base, mais comporte des informations supplémentaires (attributs, opérations, associations) (**figure 2.7 (c)**).

- **Les réalisations:** une réalisation est une relation sémantique entre classificateurs, selon laquelle un classificateur spécifie un contrat dont l'exécution est garantie par un autre classificateur (**figure 2.7 (d)**).

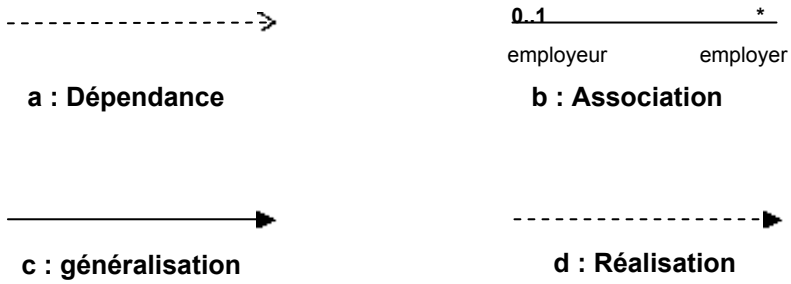

Figure 2.7 : Les relations d'UML.

2.3.6.3 Les diagrammes

Un **diagramme** est la représentation graphique d'un ensemble d'éléments qui constituent un système. La plupart du temps, il se présente sous forme d'un graphe connexe où les sommets correspondent aux éléments et les arcs aux relations. Les diagrammes servent à visualiser un système sous différentes perspectives et sont donc des projections dans un système. Pour les systèmes complexes, un diagramme ne représente qu'une vue partielle des éléments qui composent ces systèmes.

UML 2.0 comporte ainsi treize types de diagrammes représentant autant de *vues* distinctes pour représenter des concepts particuliers du système d'information. Ils se répartissent en deux grands groupes :

Diagrammes structurels ou diagrammes statiques (*UML Structure*)

- diagramme de classes (*Class diagram*)
- diagramme d'objets (*Object diagram*)
- diagramme de composants (*Component diagram*)
- diagramme de déploiement (*Deployment diagram*)
- diagramme de paquetages (*Package diagram*)
- diagramme de structures composites (*Composite structure diagram*)

Diagrammes comportementaux ou diagrammes dynamiques (*UML Behavior*)

- diagramme de cas d'utilisation (*Use case diagram*)
- diagramme d'activités (*Activity diagram*)
- diagramme d'états-transitions (*State machine diagram*)
- **Diagrammes d'interaction (*Interaction diagram*)**
  - diagramme de séquence (*Sequence diagram*)
  - diagramme de communication (*Communication diagram*)
  - diagramme global d'interaction (*Interaction overview diagram*)
  - diagramme de temps (*Timing diagram*)

Ces diagrammes, d'une utilité variable selon les cas, ne sont pas nécessairement tous produits à l'occasion d'une modélisation.

2.3.6.3.1 Diagramme de cas d'utilisation

Décrit la façon dont le système sera utilisé. Il montre les relations entre les acteurs et les cas d'utilisation du système. Les acteurs sont des entités interagissant avec

le système (en général, ils sont des utilisateurs ou des dispositifs extérieurs), et un cas d'utilisation est une manière spécifique d'utiliser le système.

2.3.6.3.2 Diagramme de classes

Le diagramme de classes est généralement considéré comme le plus important dans un développement orienté objet. Il représente l'architecture conceptuelle du système : il décrit les classes que le système utilise, ainsi que leurs liens, que ceux-ci représentent un emboîtage conceptuel (héritage) ou une relation organique (agrégation).

2.3.6.3.3 Diagramme d'objets

Le diagramme d'objets permet d'éclairer un diagramme de classes en l'illustrant par des exemples. Il est, par exemple, utilisé pour vérifier l'adéquation d'un diagramme de classes à différents cas possibles.

2.3.6.3.4 Diagramme d'états-transitions

Le diagramme d'états-transitions représente la façon dont évoluent (*i.e.* cycle de vie) les objets appartenant à une même classe. La modélisation du cycle de vie est essentielle pour représenter et mettre en forme la dynamique du système.

2.3.6.3.5 Diagramme d'activités

Le diagramme d'activités n'est autre que la transcription dans UML de la représentation du processus telle qu'elle a été élaborée lors du travail qui a préparé la modélisation : il montre l'enchaînement des activités qui concourent au processus.

2.3.6.3.6 Diagramme de séquence

Le diagramme de séquence représente la succession chronologique des opérations réalisées par un acteur. Il indique les objets que l'acteur va manipuler et les opérations qui font passer d'un objet à l'autre. On peut représenter les mêmes opérations par un diagramme de communication (section prochaine). En fait, le diagramme de séquence et le diagramme de communication sont deux vues différentes mais logiquement équivalentes (on peut construire l'une à partir de l'autre) d'une même chronologie. Ce sont des diagrammes d'interaction.

2.3.6.3.7 Diagramme de communication

Présente l'interaction organisée autour des objets et de leurs liaisons. À la différence du diagramme de séquence, il ne montre pas le temps comme une dimension séparée. Il se focalise sur l'organisation structurelle des objets qui émettent et reçoivent les messages.

2.3.6.3.8 Diagramme de composants

Représente les composants qui composent une application un système, ou une entreprise. Il montre aussi, les composants, leurs liens, et leurs interfaces publiques.

2.3.6.3.9 Diagramme de déploiement

Représente l'architecture d'exécution du système. Il comprend les nœuds, les environnements d'exécution aussi bien hardware que software, le middleware des connectant.

2.3.6.3.10 Diagramme de paquetages

Montre comment les éléments du modèle sont organisés en packages ainsi que les dépendances entre les packages.

2.3.6.3.11 Diagramme de structures composites

Décrit la structure interne d'un classifieur (une classe, un composant, ou un cas d'utilisation), comprenant les points d'interaction du classifieur avec d'autres parties du système.

2.3.6.3.12 Diagramme de global d'interaction

Une variante du diagramme d'activité qui synthétise le flux de contrôle dans un système ou un processus. Chaque nœud peut représenter un autre diagramme d'interaction.

2.3.6.3.13 Diagramme de temps

Décrit le changement dans l'état ou la condition d'une instance durant une ligne de temps. Typiquement, il montre le changement dans l'état d'un objet durant le temps en réponse aux événements externes.

2.4 UML mobile

La modélisation des applications d'agent mobile prend en considération les concepts suivants: l'itinéraire, location, la migration, cloning a distance et la sécurité.

La littérature est défini trois catégorie des approches pour modéliser les applications d'agent mobile **[Adl]:** l'approche pattern, l'approche formelle et l'approche semi formelle.

L'approche semi formelle elle-même est décomposée en deux classes **[Adl]**.

La première classe est une extension de la méthodologie orientée agent (agent-oriented methodologies), parmi les travaux concernés par cette classe d'approche la méthodologie **MaSE [MaSE]** et **m-Gaia [m-Gaia]**, cette dernière est une extension de la méthodologie **Gaia [Gaia]**.

La deuxième classe des approches semi formelle vise de proposer des extensions du langage UML pour modéliser les concepts propre à la mobilité.

Dans ce travail nous avons intéressé par l'approche semi formelle et plus particulièrement UML mobile. Dans le paragraphe suivant nous avons présenté quelques travaux dans ce sens.

- Le profile MAM-UML (Mobile-Agent Modeling with UML) **[Edg01], [Edg02]** comporte quatre point de vue (l'aspect Organisationnel, le cycle de vie, l'interaction et la mobilité) pour modéliser les applications d'agent mobile.

Le point de vue ***organisationnel*** décrit les entités *(agents, systèmes, ressources, etc.)* et les leur relations structurelles, il adresse les diagrammes de clase stéréotypés pour modéliser le domaine d'application, les deux types d'agents (stationnaire, mobile), les rôles et les ressources. Les diagrammes de package stéréotypés sont utilisés pour modéliser les systèmes d'agent mobile, les places d'exécution et les régions. De plus il existe des relations stéréotypés pour relier entre les deux types de classeur (les classes et les packages). Dans cette vue il existe des autres stéréotypes («*Identifier*», *«Location*», *«Authority», «home*», etc.) pour identifier les concepts de la mobilité.

Le point de vue ***cycle de vie*** décrit l'itinéraire qui suit l'agent mobile, les états et les activités qui se font par l'agent mobile durant son cycle de vie, il existe trois modèles pour modéliser les concepts précédant. ***Itinerary Model*** est utilisé pour modéliser le chemin qui suit l'agent mobile durant son cycle de vie en utilisant l'un des deux types de diagramme d'interaction (diagramme de collaboration et diagramme de séquence), de plus en utilisant des messages stéréotypés pour les spécifier la mobilité. ***Execution States Model*** spécifie chaque type d'agent mobile comme une machine a état, et dans ce sens le diagramme d'état transition stéréotypé par des stéréotypes de mobilité est utilisé pour modéliser le changement d'état d'un agent mobile et les événements et les actions qui influencent ce dernier durant son cycle de vie. **Activity Model** est utilisé pour modéliser les déférentes activités d'un agent mobile durant son cycle de vie, donc le diagramme d'activité d'UML est le plus recommandé dans ce cas en ajoutant des stéréotypes pour montrer la mobilité.

Le point de vue ***d'interaction*** est une complémentaire du point de vue ***cycle de vie*** dans le contexte de la vision dynamique des applications d'agent mobile, il décrit quel, pourquoi, quand et comment communiquer les agents, les rôles et les systèmes d'agent mobiles en spécifiant les interactions entre

ces entités, leur protocoles et le mécanisme de coordination. Le diagramme d'interaction d'UML (plus précisément le diagramme de collaboration) et les stéréotypes de mobilité sont utilisés pour remplir les exigences de ce point de vue.

Le point de vue ***Mobile*** inclus les modèles spécifiant le code et l'exécution des agents mobile, ce point de vue comporte deux modèles (***Data Space Management Model*** et ***Model of Code and Execution State Mobility*****).** Le *Data Space Management Model* est utilisé pour spécifier le type de ressources manipulé par les agents et le type of **bindings** maintenir par ces agents, les types de diagramme UML utilisé dans ce modèle est le diagramme de collaboration avec des relations stéréotypés, et en peut utiliser les diagrammes de clase, de package et le diagramme d'objet pour des raisons de réutilisation. Le ***Model of Code and Execution State Mobility*** décrit où, pourquoi et quand l'agent déplace, de plus quel sont les parties concernés par la migration (le code, les données ou l'état d'exécution). Une combinaison entre les deux diagrammes de composant et déploiement sont utilisé pour modéliser ces opérations de codage et d'exécution des agents mobiles.

- Mouratidis et al [Har] va introduire une extension des deux diagrammes d'UML (diagramme de déploiement et diagramme d'activité) pour rependre sur des questions sur les applications d'agent mobile. Le ***AUML deployment diagram*** est utilisé pour capturer le raison de déplacement d'un agent mobile d'un nœud à un autre et à quel location se trouve l'agent mobile. Le ***AUML activity diagram*** capture a quel moment (timing) l'agent mobile déplace d'un nœud à un autre.

- Mario K et G J [Kus] sont proposés une extension de diagramme de séquence d'UML, cette extension vise a modalisé le chemin de migration d'un agent mobile et les trois éléments de base de la mobilité (la création d'un agent mobile, le chemin qui suit l'agent mobile et la location actuel d'un agent mobile).

- Héla HACHICHA, Adlèn Loukil, et Khaled Ghedira [Hach], **[Adl]** sont présentés une approche pour modélisé et implémenté les applications d'agent mobile. Cette approche est matérialisée par une notation UML qui s'appelle **MA-UML** pour modéliser les agents mobile et un outil CASE qui s'appelle **MAMT** qui fait un mapping entre les diagrammes conceptuels et les classes d'implémentation java. Les auteurs dans cette approche introduisent des nouveaux diagrammes basés sur les diagrammes

d'UML standard en ajoutant des stéréotypes pour expliquer les concepts de mobilité, ces nouveaux diagrammes sont: le **Itinerary Diagram** est une extension de diagramme de classe d'UML, le **Navigation Diagram** est une extension de diagramme d'état transition, le **Mobile Agent Activity Diagram** est une extension de diagramme d'activité en introduisant des concepts de location, le **mobile agent lifecycle diagram** est une extension de diagramme d'état transition en ajoutant des concepts de location.

- Miao Kang, L Wang et K Taguchi [Kang] sont proposés le diagramme d'activité d'UML 2.0 comme un bon modèle pour les applications d'agent mobile, et ils sont démontrés que l'utilisation de ce modèle avec des stéréotypes est bien répondue sur les questions proposés sur la particularité des applications d'agent mobile.

- Kassem Saleh et Christo El-Morr [Kas01] présentaient une extension du langage UML qui s'appelle UML mobile (M-UML) en ajoutant pour chaque diagramme d'UML des signes graphiques et des stéréotypes pour spécifier et exprimer la mobilité. Cette extension couvre toute les aspects de la mobilité en déférentes vues et diagramme du langage UML. Pour mieux comprendre cette proposition en a présenté deux diagrammes de M-UML (le diagramme de cas d'utilisation mobile et le diagramme de séquence mobile).

Le diagramme de cas d'utilisation mobile (*Modile Use case diagram*): Le diagramme de cas d'utilisation encapsule les deux concepts *acteur* et *cas d'utilisation; les acteurs* sont des entités externes qui interagissent avec le système a travers des cas d'utilisation. Dans M-UML *les acteurs* peuvent être des agents qui sont en interaction dans le système, en appel *acteur mobile* l'agent mobile qui fait des interactions avec les autres agents du système dans des plateformes différentes contenant leur plateforme de basse. La description du système est faite par un groupe des cas d'utilisation, un ou plusieurs acteurs participent dans un cas d'utilisation, en appel *cas d'utilisation mobile* le cas d'utilisation qui permet d'un ou plusieurs *acteurs mobile* de participer dans lui, donc en minimum au moins un *acteur mobile* participe dans un *cas d'utilisation mobile*, dans les systèmes d'agent mobile il existe les deux sortes des cas d'utilisations (mobile et non mobile). Graphiquement en ajoutent un petit carré porte le nom 'M' dans l'extrémité haute gauche d'un cas d'utilisation pour dire que ce cas d'utilisation est un *cas d'utilisation mobile*, et pour les *acteurs mobile* en ajoutent un 'M' dans le cercle du petit honomme qui représente l'acteur.

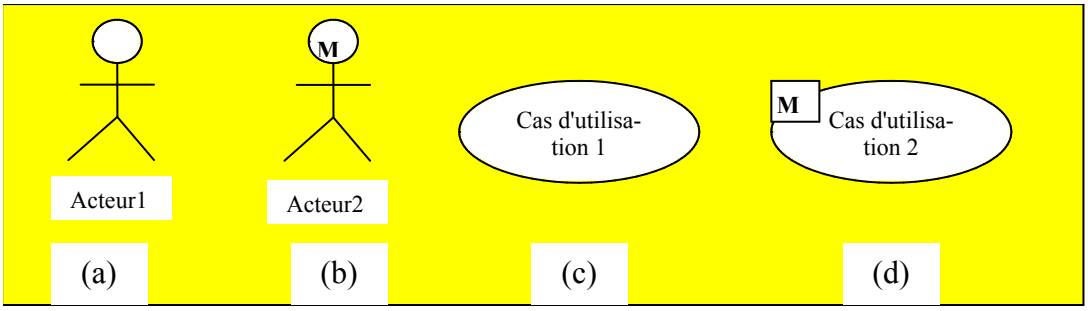

Figure 2.8 : les concepts du graphe de cas d'utilisation mobile (a): acteur non mobile ; (b): acteur mobile; (c): cas d'utilisation non mobile; (d): cas d'utilisation mobile

Le Diagramme de séquence mobile (*Mobile Sequence diagram)*: Le diagramme de séquence est l'un des deux types de diagramme d'interaction, il représente la succession chronologique des opérations réalisées par un acteur (agent), le temps y est représenté par une ligne verticale pointillée et il s'écoule de haut en bas. Les interactions sont représentées par des flèches qui démarrent de l'agent source vers l'agent destinataire. La ligne de vie d'un agent qui se trouve dans une plateforme qui défère de son plateforme de basse et qui fait une interaction avec les autres agents jusqu'a le retour de son plateforme de base est représenté graphiquement comme une ligne de vies de diagramme de séquence de UML standard avec un remplissage par le caractère 'M'. En appelle interaction mobile l'interaction qui dépare de la ligne de vie mobile. Une interaction mobile stéréotypé par «localised» est l'interaction qui reliée deux agents qui se trouvent dans une même plateforme. L'interaction mobile qui reliée entre l'agent mobile et deux autres agents qui ne se trouvent pas dans la même plate forme se représente graphiquement par une ligne horizontale reliée ses trois agents en ajoutant un petit carrée porte le caractère 'R' dans l'intersection entre l'interaction et la ligne de vie de l'agent qui situe dans la première plateforme. De plus une ligne pointillée est utilisée pour initialiser la plateforme source de l'agent mobile dans les interactions «localised» et a distante, cette représentation est justifiée car l'agent mobile peut faire une migration dans une plateforme a distance et faire une interaction avec l'agent situé. Finalement l'interaction qui modélise le retour de l'agent d'une plateforme quelconque vers leur plateforme de base est stéréotypée par **«agentreturn»**.

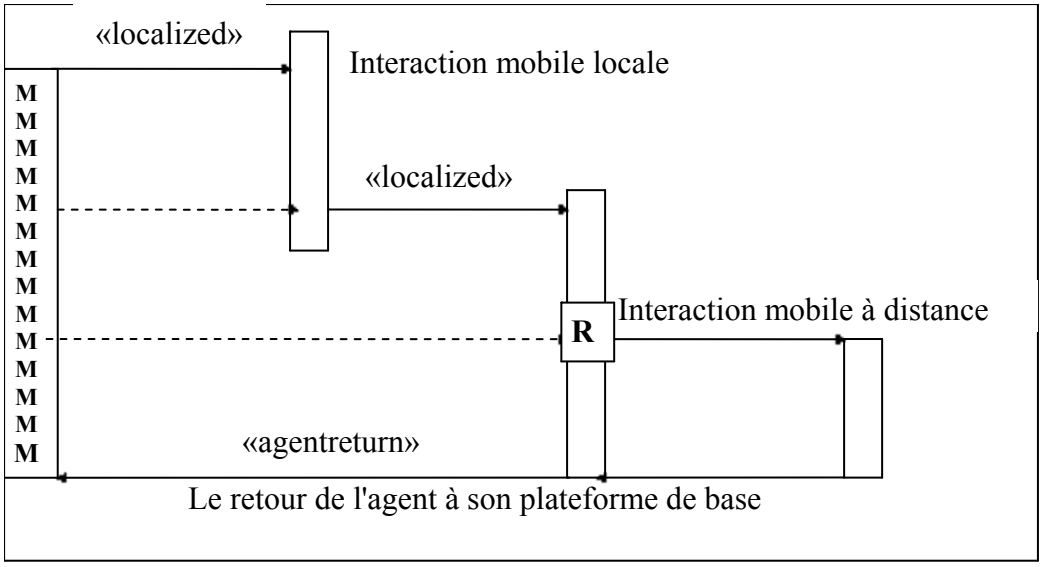

Figure 2.9: les concepts du graphe de séquence mobile **[Kas01].**

- Klein et al. [Klein] sont proposés une extension du langage UML pour modélisé les applications d'agent mobile. Ils sont introduits des nouveaux concepts pour modéliser la mobilité forte (*strong mobility*), la mobilité faible (*weak mobility*) et l'action de cloning (*cloning action*). Ils sont ajoutés des nouveaux stéréotypes pour modéliser les entités d'environnements des agents mobiles (l'agent mobile, système d'agent mobile, place et région). De plus ils sont proposés une extension du diagramme de sequence pour modéliser le mouvement d'un agent d'une location à une autre.

- Muscutariu et al. [Muscutariu] sont proposés un langage de description architectural (ADL (architectural description language)) pour modéliser les systèmes d'agent mobile. Cette langage ADL est défini un simple profile d'UML en présentant un minimum de concepts et d'interfaces d'opérations nécessaire pour faire une interopérabilité entre les systèmes d'agent mobile hétérogène. Ils proposaient une représentation graphique en utilisant les diagrammes de composant et de déploiement d'UML, les composants stéréotypés sont utilisés pour modéliser les agents mobiles et les nœuds stéréotypés sont utilisés pour modéliser les systèmes d'agent mobile. De plus en utilisant les packages pour modéliser les places et les régions. La modélisation de la mobilité est supportée par des relations stéréotypées et les opérations *move*, *beforeMove* et *afterMove* préparent le processus de migration.

2.5 Conclusion

Dans ce chapitre on a essayé de présenter une introduction au langage de modélisation UML, en commençant par présenter la modélisation dans un cadre général, ensuite l'historique, la notation, et finalement une brève présentation des extensions d'UML pour la mobilité.

En conséquence, UML possède plusieurs facettes. C'est une norme, un langage de modélisation objet, un support de communication, un cadre méthodologique. UML est tout cela à la fois. Malgré la commodité de la modélisation et de la compréhension des modèles UML, la vérification de tels modèles est une tâche assez délicate, à cause de la sémantique semi-formelle

Pour bénéficier des avantages d'UML dans la modélisation des systèmes sans omettre la tâche de vérification des modèles résultants d'une telle modélisation, plusieurs travaux se sont focalisés sur le principe de transformation de modèles pour obtenir des modèles dont la vérification est abordable.

Le chapitre suivant va introduire la notion de transformation de modèle et plus précisément la transformation des graphes.

Chapitre 03: La transformation de modèle

3.1 Introduction

La modélisation, au sens large, est en effet l'utilisation efficace d'une représentation simplifiée d'un aspect de la réalité pour un objectif donné **[Marc]**. Loin de se réduire à l'expression d'une solution à un niveau d'abstraction plus élevé que le code, la modélisation en informatique peut-être vue comme la séparation des différents besoins fonctionnels et préoccupations extra-fonctionnelles (telles que sécurité, fiabilité, efficacité, performance, ponctualité, flexibilité, etc.) issus des exigences.

Ce que propose l'approche de l'ingénierie des modèles (IDM, ou MDE en anglais pour Model Driven Engineering) est simplement de mécaniser le processus que les ingénieurs expérimentés suivent à la main. L'intérêt pour l'IDM a été fortement amplifié à la fin du 20ième siècle lorsque l'organisme de standardisation OMG (Object Modeling Group) a rendu publique son initiative MDA (Model Driven Architecture), qui peut être vue comme une restriction de l'IDM à la gestion de l'aspect particulier de dépendance d'un logiciel à une plateforme d'exécution.

Dans le contexte de l'IDM, la notion de transformation de modèle joue un rôle fondamental ; Le processus de conception peut alors être vu comme un ensemble de transformations de modèles partiellement ordonné, chaque transformation prenant des modèles en entrée et produisant des modèles en sortie, jusqu'à obtention d'artéfacts exécutables.

Dans ce chapitre, on va présenter le concept de transformation de modèles en commençant par une représentation de concepts de transformation de modèle dans le cadre général, ensuite, on présente une énumération des différents types de transformations, suivie par une classification des différents approches, et enfin on présente, aussi, un cadre spécifique des transformations de modèles basé sur les transformations de graphe.

3.2 Les approche MDA et IDM

Il s'agit de modéliser les applications que l'on veut créer de manière indépendante de l'implémentation cible (niveau matériel ou logiciel). Ceci permet une grande réutilisation des modèles. Les modèles ainsi créés (PIM - Platform Independant Model) sont associés à des modèles de plate-forme (PM -Platform Model), et transformés, pour obtenir un modèle d'application spécifique à la pate-forme (PSM – Platform Specific Model).

Des outils de génération automatique de code permettent ensuite de créer le programme directement à partir des modèles.

Ces approches permettent de plus de faire évoluer facilement, à partir des modèles, les applications : le développement d'un nouveau module, quand tous les modèles nécessaires sont disponibles, peut ne pas prendre plus de quelques minutes.

3.2.1 Définition de plateforme

Une plateforme est un ensemble de sous-système et de technologies qui fournissent un ensemble cohérent de fonctionnalités à travers des interfaces et des modèles d'utilisateurs spécifiés, qui peuvent être employés par n'importe quelle application soutenue par cette plateforme sans souci pour les détails de la façon dont la fonctionnalité fournie par la plateforme est mise en application. **[OMG 03]**.

3.2.2 L'ingénierie des logiciels (IDM)

L'Ingénierie Dirigée par les Modèles est une approche plus globale et générale que le MDA. Elle cherche à en appliquer les principes et à les généraliser pour tout espace technologique tels l'orienté-objet, les ontologies, les documents XML ou les grammaires de langages. Le MDA peut ainsi être perçu comme un processus spécifique de type IDM.

L'IDM repose sur les principes suivants :

A. ***Capitalisation*** : les modèles doivent être réutilisables,
B. ***Abstraction*** : les modèles doivent être indépendant des technologies de mise en œuvre afin d'adapter la logique métier à différents contextes et de permettre de faire plus facilement évoluer les applications vers de nouvelles technologies,
C. ***Modélisation*** : abordée selon une vision productive (pour générer le code final du logiciel pour une technologie de mise en œuvre donnée) par opposition à la traditionnelle vision contemplative (but de documentation, spécification, communication),
D. ***Séparation des préoccupations*** : l'IDM s'illustre généralement selon les deux principales préoccupations du métier et de la plateforme de mise en œuvre mais d'autres préoccupations sont possibles.

Pour passer à une vision productive des modèles, il est nécessaire qu'ils soient bien définis ; ce qui amène l'importante notion de *méta-modèle*. Ainsi, les modèles productifs pourront être manipulés et interprétés via des outils **[Laf]** répondant aux

besoins de l'approche IDM : définition de méta-modèles, définition de langages, définition de transformations et leur exécution (selon les types de transformations, cette catégorie d'outils peut être amenée à traiter simultanément des méta-modèles différents, non nécessairement issus des mêmes espaces technologiques), génération de code (considérée comme une transformation de modèles particulière), composition de modèles (appelée tissage de modèles dans certains cas), génération d'environnement graphique de modélisation, vérification de conformité de modèles, spécification de contraintes, etc.

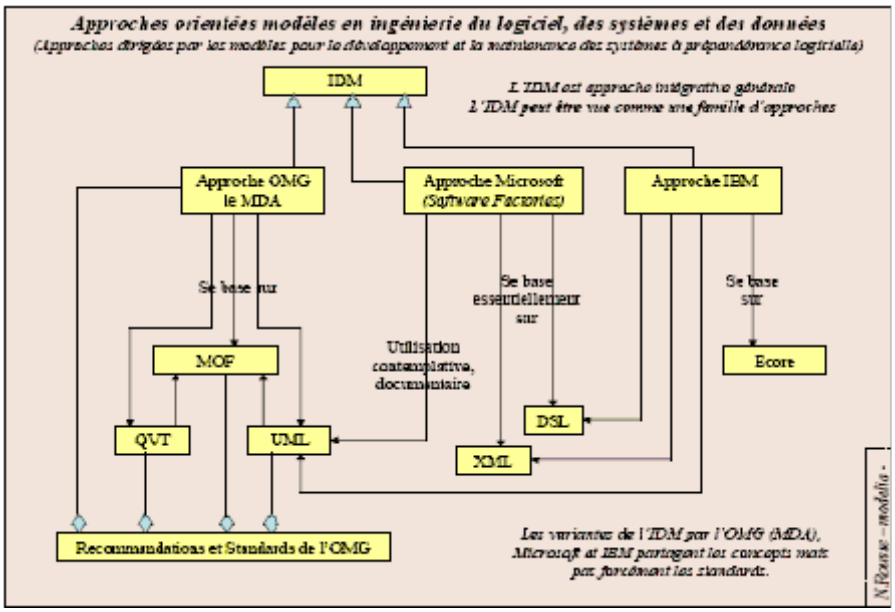

Figure 3.1: Les variantes de l'IDM **[Favre]**

3.2.3 L'approche MDA

Partant du constat de l'évolution continue des technologies et du coût élevé de l'adaptation des applications logicielles à ces technologies, l'OMG proposa le MDA **[OMG 04]** fin 2000. Le but principal de cette approche est de séparer les parties métier de leur mise en œuvre technologique et ainsi de garantir l'interopérabilité des modèles fonctionnels pour différents choix d'implémentation.

Conceptuellement, le MDA propose trois points de vue, associés à leurs modèles respectifs : Le CIM, le PIM et le PSM.

3.2.3.1 Le CIM

CIM (*Computer Independent Model*) correspondant aux modèles du « domaine » (*domain/business model*) indépendants de toute implémentation informatique et recensant les besoins des utilisateurs en utilisant le vocabulaire du praticien.

3.2.3.2 Le PIM

PIM (*Platform Independent Model*) correspondant à la spécification de la partie « métier » d'une application, conformément à une analyse informatique cherchant à répondre aux besoins métiers indépendamment de la technologie de mise en œuvre.

3.2.3.3 Le PSM

PSM (*Platform Specific Model*) correspondant à la spécification d'une application après projection sur une plate-forme technologique donnée.

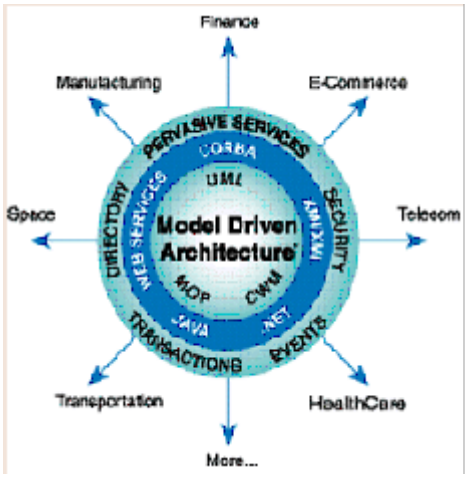

Figure 3.2: Illustration OMG du MDA **[OMG 04]**

3.2.4 L'architecture d'MDA

MDA est une Architecture à 4 niveaux d'abstraction (**Figure 3.3)**:

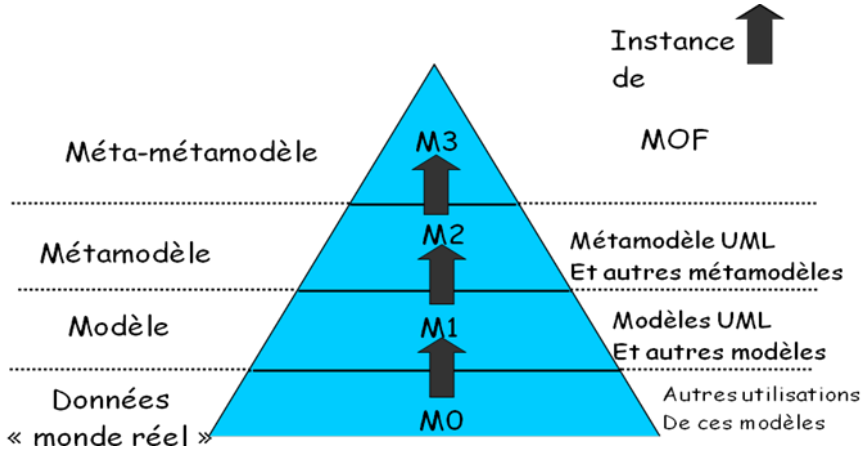

Figure 3.3: Les quatre niveaux d'abstraction pour MDA

A. **<u>Le niveau M0</u>**: Niveau de modélisation des données réelles, composé des informations que l'on souhaite modéliser. Exemple (Niveau du monde réel).

B. **<u>Le niveau M1:</u>** Composé de modèles d'informations (PIM, PSM).

- Modèle UML
- Tout modèle est exprimé dans un langage unique dont la définition est fournie explicitement au niveau M2

C. **Le niveau M2:** Composé de langages de définition des modèles d'informations = Métamodèles

- Le métamodèle UML
  - Décrit dans le standard UML,
  - Définit la structure interne des modèles UML.

D. **Le niveau M3:** composé d'une unique entité: langage unique de définition des métamodèles

- Méta-métamodèle ou MOF

3.2.5 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 **[OMG 04]**.
D. **CWM**, base de données pour méta-données.

L'OMG n'a pas jugé utile de standardiser un processus associé à ces outils. Leur rôle est de répondre aux besoins des utilisateurs de manière générique, et non de proposer de solutions définitives pour certains types d'applications précises.

Un processus de génie logiciel exploitant les possibilités de MDA a cependant été proposé : le 'Model-Driven Software Development' **[mdsd]**.

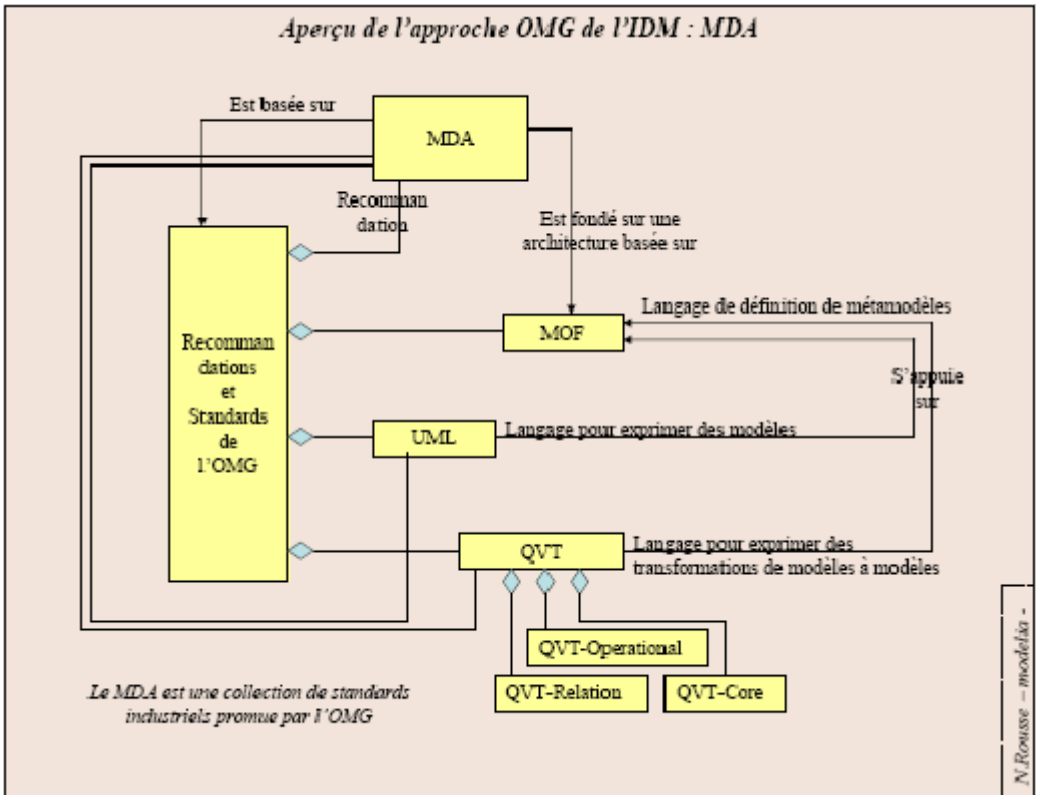

Figure 3.4: Les outils d'MDA **[Favre]**

3.2.6 La transformation des modèles dans l'approche MDA

La transformation de modèle MDA se fait par mapping entre un modèle initial et un modèle cible. Chaque modèle doit être décrit par un méta-modèle, qui recense les caractéristiques de ce modèle. Le mapping est alors définit comme une traduction entre le méta-modèle initial et le méta-modèle cible. **La figure 3.5** présente le principe de la transformation.

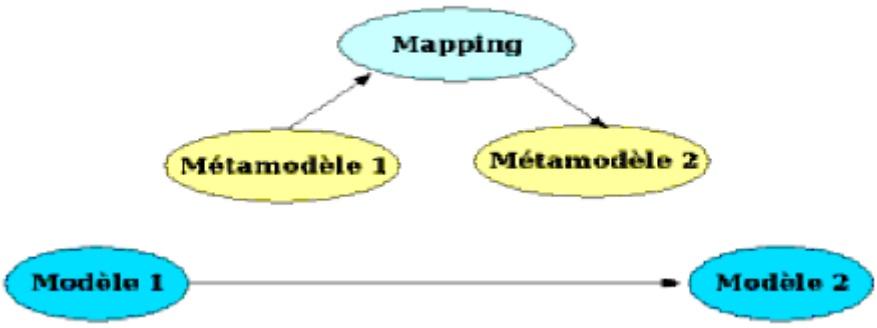

Figure 3.5: Le principe de la transformation des modèles dans l'approche

On distingue deux types de mappings : les mapping verticaux, qui changent le niveau d'abstraction, et les mapping horizontaux, qui le conservent.

A. Les mappings verticaux: On en distingue deux sortes :

- Mappings de modèle d'analyse vers le modèle d'implémentation (ou mappings de raffinement). Ils permettent de préciser le modèle de l'application, en fonction de l'implémentation souhaitée. Souvent, un seul modèle initial suffit. Parfois, des contraintes sont nécessaires (sinon le modèle cible est incomplet), ou bien plusieurs modèles sont utilisés comme source.
- Mappings d'abstraction. C'est la transformation inverse. Elle permet une meilleure compréhension du code (reverse engineering), ou la migration d'une plate-forme vers une autre.

B. Les mappings horizontaux: On en distingue trois sortes:

- Mappings de représentation. On l'utilise pour passer d'un format à un autre, le second disposant de plus d'outils de manipulation et de représentation. Peu utile s'ils ne sont pas réversibles.
- Mappings d'optimisation, pour améliorer la performance.
- Mappings de reconstruction, pour améliorer la maintenabilité et la lisibilité du code.

Les mappings sont réalisés soit par des règles générales, soit par corrélation, c'est à dire par appariement de propriétés du modèle source avec des propriétés du modèle cible.

3.3 La transformation de modèles

Les transformations de modèles sont au cœur de l'ingénierie dirigée par les modèles. Elles sont habituellement développées par des programmeurs spécialisés et leur code doit être remis à jour lors de toute variation des besoins ou des métamodèles impliqués. Ils peuvent également s'agir de transformer un modèle fonctionnel indépendant de l'application (PIM - Platform Independant Model) en modèle prenant en compte les contraintes de déploiement (PSM - Platform Specific Model).

3.3.1 Une classification des approches de transformation de modèles

Les approches de transformation de modèle ont été classées en se basant sur plusieurs points de vue **[Nav]**, chaque point de vue permet une classification particulière.

Selon **Naveen Prakash [Nav]** il existe trois points de vue pour classer les approches de transformation de modèles:

A. **Vue de la technique de transformation utilisée**: Cette point de vue des approches de transformation focalise sur les techniques utilisés pour transformer un modèle d'entré vers un modèle de sortie. Dans ce point de vue Il existe une classification **[Car]** qui classifie les techniques de transformation de modèle en trois grandes catégories:

- **Approche déclarative:** Recherche de certains patrons (d'éléments et de leurs relations) dans le modèle source, et chaque patron trouvé est remplacé dans le modèle cible par une nouvelle structure d'élément. Dans cette approche l'écriture de la transformation est « assez » simple mais ne permet pas toujours d'exprimer toutes les transformations facilement
- **Approche impérative:** est proche des langages de programmation, il s'agit d'un parcourt dans le modèle source dans un certain ordre et on génère le modèle cible lors de ce parcours. Dans cette approche l'écriture de la transformation est plus complexe mais permet de toutes les définir.
- **Approche hybride** : Cette approche est à la fois déclarative et impérative, Celle qui est utilisée en pratique dans la plupart des outils de transformation de modèle.

Il existe une autre classification qui faite par **CZarnneki [Cza]** qui se base sur les techniques de transformation utilisées dans les approches et les facettes qui les caractérisent.

B. **Vue du support du langage de transformation:** dans cette vue on a intéressé par les caractéristiques des langages de transformation qui est automatisés les taches de transformation comme les patrons d'application, perfectionnement et la définition des règles de transformation.

C. **Vue générique (domaine d'application des approches de transformation)**: Dans ce point de vue une autre classification a été faite par **Naveen Prakash [Nav]** qui propose une classification multidimensionnelle où il focalise sur une dimension non traitée dans les autres classifications qui est le domaine d'application et généricité (capacité de transformer n'importe quel modèle d'entré à n'importe quel modèle de sortie), il prend en considération l'aspect utilisation de transformation de modèles dans l'ingénierie des méthodes.

3.3.2 Présentation d'une classification des approches de transformation de modèle basé sur les techniques de transformation utilisées

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

3.3.2.1 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*).

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. Le projet **Jamda**, destiné à la génération de code Java est un exemple de cette approche **[Jam]**.
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* **[And]** (un générateur de code qui se repose notamment sur la technologie ouverte *Velocity* **[Jak]** 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.

3.3.2.2 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. **[OMG 04]**.

Il existe une 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 calcule des différentes vues du système, leur synchronisation et pour la vérification et la validation.

3.3.2.2.1 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.

A. **Spécification :** les pré-conditions et les post-conditions exprimées en **OCL** (Object Constraint Language) ou un autre langage se spécification sont utilisés dans certaines approche.

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 Hand Side*) et qui accède au modèle source, et un partie droit ***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'ordonnancement 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 basée 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.

3.3.2.2.2 Différentes approches

On peut globalement distinguer cinq types d'approches :

A. **Approches par manipulation directe**: La représentation interne des modèle source et cible et leurs manipulations par un ensemble d'APIs est destinées comme un principe pour ces approches. Généralement on à implémenter ces modèles par des cadres structurants orientés objets (object-oriented framework) qui fournissent un ensemble minimale d'infrastructure pour organiser la transformation. Les règles de transformations et leur ordonnancement sont implémentés par le développeur en utilisant un langage de programmation comme Java par exemple.

B. **Approches relationnelles:** Généralement, les transformations de modèle dans ces approches sont bidirectionnelles basés sur la logique déclarative reposant sur des relations d'ordre mathématique. Dans ces approches des contraintes sont utilisés pour faire des relations entre le modèle source et le modèle cible.

C. **Approches basées sur les transformations de graphes:** Ces approches sont similaires aux approches relationnelles dans la façon déclarative de ces transformations. En se basant sur le principe de ***pattern matching*** les règles de transformation sont définies pour des fragments de modèle dans le modèle source, et chaque fragment du modèle dans le modèle source qui correspond à certains critères est remplacé par un fragment de modèle qui correspond au modèle cible. Ces fragments de modèles, sont exprimés soit dans les syntaxes concrètes respectives des modèles soit dans leur syntaxe abstraite. **VIATRA**, **ATOM**, **GreAT**, **UMLX** et **BOTL** sont des approches basées sur les transformations de graphes.

D. **Approches basées sur la structure:** Le principe de ces approches consiste en construire la structure hiérarchique du modèle cible en premier lieu, ensuite on ajuster les attributs et références dans le modèle cible.

E. **Approches hybrides**: Les approches hybrides sont une combinaison des différentes techniques.

3.3.3 Les outils de transformation de modèles

De nombreux outils, tant commerciaux que dans le monde de l'open source sont aujourd'hui disponibles pour faire la transformation de modèles. On peut grossièrement distinguer quatre catégories d'outils **[Marc]** :

A. Les outils de transformation génériques qui peuvent être utilisées pour faire de la transformation de modèles. Dans cette catégorie on trouve notamment d'une part les outils de la famille XML, comme XSLT ou Xquery, et d'autre part les outils de transformation de graphes (la plupart du temps issus du monde académique). Les premiers ont l'avantage d'être déjà largement utilisés dans le monde XML, ce qui leur a permis d'atteindre un certain niveau de maturité. En revanche, l'expérience montre que ce type de langage est assez mal adapté pour des transformations de modèles complexes (c'est-à-dire allant au-delà des problématiques de transcodage syntaxique), car ils ne permettent pas de travailler au niveau de la sémantique des modèles manipulés mais simplement à celui d'un arbre couvrant le graphe de la syntaxe abstraite du modèle ce qui impose de nombreuses contorsions qui rendent rapidement ce type de transformation de modèles complexes à élaborer, à valider et surtout à maintenir sur de longues périodes.

B. Les facilités de type langages de scripts intégrés à la plupart des ateliers de génie logiciel. Dans la seconde catégorie, on va trouver une famille d'outils de transformation de modèles proposés par des vendeurs d'ateliers de génie logiciel. Par exemple, l'outil Arcstyler de Interactive Objects propose la notion de MDA-Cartridge qui encapsule une transformation de modèles écrite en JPython (langage de script construit à l'aide de Python et de Java). Ces MDA-Cartridges peuvent être configurées et combinées avant d'être exécutées par un interpréteur dédié. L'outil propose aussi une interface graphique pour définir de nouvelles MDA-Cartridges. Dans cette catégorie on trouve aussi l'outil Objecteering de Objecteering Software (filiale de Softeam), qui propose un langage de script pour la transformation de modèles appelé J, ou encore OptimalJ de Compuware qui utilise le langage TPL, et bien d'autres encore, y compris dans le monde de l'open source avec des outils comme Fujaba (From UML to Java

and Back Again). L'intérêt de cette catégorie d'outils de transformation de modèles et d'une part leur relative maturité (car certains d'entre eux comme le J d'Objecteering sont développés depuis plus d'une décennie) et d'autres pas leur excellente intégration dans l'atelier de génie logiciel qui les héberge. Leur principal inconvénient est le revers de la médaille de cette intégration poussée : il s'agit la plupart du temps de langages de transformation de modèles propriétaires sur lesquels il peut être risqué de miser sur le long terme. De plus, historiquement ces langages de transformation de modèles ont été conçus comme des ajouts au départ marginaux à l'atelier de génie logiciel qui les héberge. Même s'ils ont aujourd'hui pris de l'importance, ils ne sont toujours pas vus comme les outils centraux qu'ils devraient être dans une véritable ingénierie dirigée par les modèles. De nouveau ces langages montrent leur limitation lorsque les transformations de modèles deviennent complexes et qu'il faut les gérer, les faire évoluer et les maintenir sur de longues périodes.

C. Les outils conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standards. Cette catégorie regroupe les outils conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standards. On va y trouver par exemple Mia-Transformation de Mia-Software qui est un outil qui exécute des transformations de modèles prenant en charge différents formats d'entrée et de sortie (XMI, tout autre format de fichiers, API, dépôt). Les transformations sont exprimées comme des règles d'inférence semi-déclaratives (dans le sens où il n'y a pas de mécanisme de type programmation logique), qui peuvent être enrichies en utilisant des scripts Java pour des services additionnels tels que la manipulation de chaînes. PathMATE de Pathfinder Solutions est un autre environnement configurable de transformation de modèles qui s'intègre particulièrement bien avec les ateliers de modélisation UML dominant le marché. Dans le monde académique on va trouver de nombreux projets open source s'inscrivant dans cette approche : les outils ATL de l'INRIA, AndroMDA, BOTL (Bidirectional Object oriented Transformation Language),Coral (Toolkit to create/edit/transform new models/modeling languages at run-time), Mod-Transf (XML and ruled based transformation language), QVTEclipse (une implantation préliminaire de quelques unes des idées du standard QVT

dans Eclipse) ou encore UMT-QVT (UML Model Transformation Tool). Ces langages spécifiquement conçus pour la transformation de modèle ont l'avantage de permettre d'exprimer très simplement les transformations de modèles simples (comme par exemple des transcodages syntaxiques), mais ils trouvent rapidement leurs limites lorsque les transformations de modèles deviennent complexes du point de vue algorithmique, ou bien lorsqu'il faut gérer de nombreuses variantes (contexte des lignes de produits) et les faire évoluer et les maintenir sur de longues périodes.

D. Les outils de méta-modélisation " pur jus " dans lesquels la transformation de modèles revient à l'exécution d'un méta-programme. La dernière catégorie d'outils de transformation de modèles est celle des outils de métamodélisation. La transformation de modèles revient alors à l'exécution d'un métaprogramme orienté objet, pour lequel il est relativement aisé de construire un framework facilitant la gestion et la maintenance de nombreuses variantes de transformations nécessaires au contexte des lignes de produits. Le plus ancien et le plus connu de ces outils est certainement MetaEdit+ de MetaCase. Celui-ci permet de définir explicitement un métamodèle, et au même niveau de programmer tous les outils nécessaires, allant des éditeurs graphiques aux générateurs de code en passant par des transformations de modèles. Dans une veine similaire, XMF-Mosaic de Xactium est un environnement complet de définition de langage. Au coeur de XMF-Mosaic on trouve un noyau exécutable de définition de langage (qui est d'ailleurs auto-définissant). Tout est modélisé sur cette base, que ce soit les langages (e.g. UML), les outils, les GUI, parsers et autres analyseurs XML, mais à la différence de l'environnement MetaEdit+ obtenu par génération plutôt que par instanciation. Le langage de méta-modélisation de base est un langage orienté objet qui a toute la puissance d'un langage de programmation complet ce qui en fait un outil particulièrement puissant pour la transformation de modèles. Dans le monde de l'open source on trouvera dans cette catégorie l'environnement KerMeta **[Ker]** développé à l'INRIA qui se présente comme l'ajout d'un aspect d'exécutabilité dans le MOF par un mécanisme astucieux inspiré du tissage d'aspects lui permettant de garder une compatibilité totale avec les environnements MOF standard (par exemple EMF).

3.4 La transformation des graphes

La théorie des graphes ouvre un grand champ de modélisation conduisant à des solutions efficaces pour de nombreux problèmes, et dans sa définition intuitive un *graphe* est un schéma constitué par un ensemble de *points* et par un ensemble de *flèches* reliant chacune deux de ceux-ci.

Les graphes sont servis pour modéliser les systèmes complexes d'une manière simple et intuitive, et la transformation des graphes peut être exploitée pour spécifier comment évoluer ces modèles.

Une transformation de graphe **[M-And] [L-Bar] [G-Kar]** consiste en l'application d'une règle à un graphe, et la partie du graphe qui corresponde cette règle sera remplacée par un autre graphe. Ce processus est réitéré jusqu'à ce qu'aucune règle ne puisse être appliquée.

L'ensemble des règles de transformation de graphe constituent ce qu'on appelle le modèle de la grammaire de graphe. Une grammaire de graphe est une généralisation, pour les graphes, des grammaires de Chomsky. Chaque règle d'une grammaire de graphe se compose d'une partie gauche (**LHS**) et une partie droite (**RHS**). Chaque une des ces deux partie sont des graphes.

Dans le processus de transformation de graphe on distingue les graphes non terminaux qui sont des résultats intermédiaires sur lesquels les règles sont appliquées et les graphes terminaux (ne peut plus appliquer des 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**.

Pour bien comprendre le principe des transformations de graphes il faut comprendre quelque concept sur la théorie des graphes. Dans les paragraphes suivants on va présenter brièvement la notion de théorie des graphes.

3.4.1 Définition d'un graphe

On appelle graphe G le couple (S,A) constitué d'un ensemble S de sommets et d'un ensemble A d'arcs. Chaque arc de A relie deux sommets de S.

Un graphe peut être orienté ou non selon le sens que l'on donne aux arcs, les deux sommets qu'ils relient pouvant être considères comme ordonné ou non. **[Yann]**

A. **Graphe orienté (GO):** Il se représente par des points et des flèches entre les points : les points sont les sommets du graphe, les flèches des arcs (orientés) qui relient certains sommets entre eux. D'un point de vue mathématique, si S

est l'ensemble des sommets, un graphe représente une relation binaire entre des éléments de S.

B. Graphe non orienté (GNO): Un graphe non orienté n'est qu'un graphe orienté symétrique ; si un arc relie le sommet *a* au sommet *b*, un autre arc relie le sommet *b* au sommet *a* : on ne trace alors qu'un trait entre *a* et *b* que l'on appelle une « arête ».

C. Degré d'un graphe: Dans le cas d'un GO, le « degré sortant » d'un sommet *x* est le nombre d'arcs qui partent de *x*, noté $d^+(x)$, et son « degré entrant », noté $d^-(x)$, est le nombre d'arcs arrivant au sommet *x*. On a la relation : $\Sigma\, d^+ = \Sigma\, d^- = |A|$ (nombre d'arcs). Dans le cas d'un GNO, le degré est le nombre d'arêtes rattachées au sommet *x* (les boucles comptent pour 2). On a la relation : $\Sigma\, d = 2|A|$.

D. Chemin ou chaîne: C'est une succession d'arcs parcourus dans le même sens. Le nombre d'arcs parcourus s'appelle la longueur du chemin. On parle de « chaîne » si l'on ne tient pas compte de la direction des arcs ; on parle ainsi de chaîne dans les GNO. Si un chemin revient à son point de départ, on parle de « circuit » dans un GO, ou de « cycle » dans un GNO. L'adjectif « élémentaire » s'applique quand le chemin ou le cycle parcourt des sommets distincts. La « distance » entre deux sommets est la longueur du plus court chemin entre ces deux sommets. Enfin le « diamètre » d'un graphe est la plus grande distance séparant deux sommets de ce graphe.

E. Connexité: On dit qu'un GNO est « connexe » s'il deux éléments quelconques de ce graphe sont reliés par au moins une chaîne. On appelle « composante connexe » d'un GNO, un sous-ensemble maximal de sommets tels qu'il existe une chaîne entre deux sommets quelconques. Un GO est « fortement connexe » si, quels que soient deux sommets *x* et *y*, il existe un chemin reliant *x* à *y* et un reliant *y* à *x*. Un graphe est « *k*-connexe » si on peut le déconnecter en retirant *k* sommets et qu'on ne peut pas en en supprimant $k - 1$; son « degré de connexité » est alors *k*. Cela permet, par exemple, de mesurer la résistance aux pannes d'un réseau informatique.

Figure 3.6: Graphe à deux composantes connexes

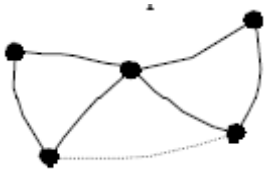

Figure 3.7: Graphe 1-connexe (traits pleins) Avec les pointillés, il devient 2-connexe.

On définit de façon analogue la « *k*-arête connexité » qui concerne les arêtes au lieu des sommets.

3.4.2 Les déférents types de graphes

A. **Arbre :** Ce sont des GNO connexes sans cycle. Ou encore des GNO connexes dont le nombre de sommets est égal au nombre d'arêtes plus 1.

Figure 3.8: l'arbre

B. **Graphe complet (clique):** C'est un GNO avec toutes les connexions possibles ; un graphe complet à *n* sommets est noté K*n*.

Figure 3.9: le graphe complet K_n

Dans le cas d'un graphe complet K*n*, le nombre d'arêtes est ***n(n-1)/2*** .

On appelle « tournoi » **[Sopena] (Figure 3.10)** un graphe complet orienté.

Une flèche peut alors représenter la relation « a gagné sur ».

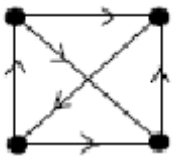

Figure 3.10: Le graphe tournoi **[Sope-**

C. **Graphe biparti:** Un ensemble I de sommets d'un graphe est « indépendant » si aucun élément de I n'est connecté à un autre élément de I. Un graphe « bi-

parti » est un graphe qui représente des relations entre deux ensembles indépendants I et J.

On définit de même un graphe multiparti.

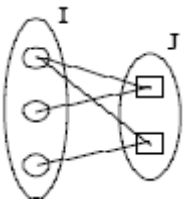

Figure 3.11: Le graphe biparti

D. **Graphe valué ou pondéré:** Quand les arêtes représentent un coût (en argent, en_temps…), on leur attribue un nombre ; ce qui donne un graphe valué ou pondéré. **(Figure 3.12)**

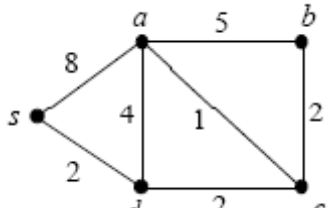

Figure 3.12: Le graphe valué ou pondéré **[Sopena]**

E. **Hypergraphe:** Quand les relations ne sont pas toutes binaires, on parle d'hypergraphe ; on peut alors le transformer en graphe biparti **(voir Figure 3.13).**

Figure 3.13: Une transformation d'un hypergraphe vers un graphe biparti **[Sopena]**

3.5 Conclusion

Dans ce chapitre, on a essayé de présenter le concept de transformation de modèles qui est l'une des pièces centrales de l'approche MDA, car elle permet l'automati-

sation et/ou l'assistance des tâches qui concourent à l'élaboration des modèles, depuis les phases amonts du développement, jusqu'au test en passant par la production de code.

Dans ce contexte, on a tenté de présenter les différentes approches existantes dans la littérature, en commençant par une brève introduction aux approches IDM et MDA, ensuite, on présente une énumération des différents types de transformations, suivie par une classification des différents approches, et enfin on présente un cadre spécifique des transformations de modèles basé sur les transformations de graphe.

Le but d'introduire ce concept est de permettre la compréhension de notre travail qui consiste à la proposition d'une grammaire de graphe permettant la transformation des modèles UML mobile vers les RDP nested nets. Le chapitre suivant porte plus de détail dans ce cadre.

Chapitre 04: Les réseaux de Petri

4.1 Introduction

Le réseau de Petri (ou par abréviation le réseau) est un outil graphique et mathématique pour modéliser et analyser les systèmes discrets, particulièrement les systèmes concurrents, parallèles, non-déterministes, etc. En étant rôle d'outil graphique, il nous aide à comprendre facilement le système modélisé, et plus il nous permet de simuler les activités dynamiques et concurrentes. Avec le rôle d'outil mathématique, il nous permet d'analyser le système modélisé grâce aux modèles de graphes, aux équations algébriques, etc**. [TRAN]**.

Dans ce chapitre, la modélisation par Réseaux de Petri est rappelée, les propriétés des RdPs sont ensuite discutées. Les extensions des RdPs pour modéliser les applications mobiles sont mises en perspective. Le chapitre se termine sur une discussion succincte sur la modélisation des systèmes adaptatifs et mobiles par les RdPs.

4.2 Les réseaux de Petri (RdP)

4.2.1 Définition

On appelle Réseau de Petri non marqué le quadruplet Q =< P; T; I;O > où

– P est un ensemble fini non vide de **places**;

– T est un ensemble fini non vide de **transitions**;

– P∩T = ø ;

– I(Ti) est l'ensemble des places qui sont en entrée de la transition i ;

– O(Ti) est l'ensemble des places qui sont en sortie de la transition i.

On appelle Réseau de Petri **marqué** R =< Q;M0 > où M0 est le marquage initial.

4.2.2 Historique des RdPs

Historiquement **[TRAN]**, le réseau est présenté par Carl Adam Petri dans sa thèse "Communication avec des Automates" en Allemagne _a Bonn en 1962. Ce travail a continue 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 Petri et des méthodes relationnels ont été organisés à MIT. En 1981 le premier livre du réseau de Petri a été publié en Anglais par J. Peterson. Aujourd'hui, suivre Petri-Net Newsletter, chaque année il y a des 600 aux 800 d'œuvres des réseaux de Petri sont publiés.

4.2.3 Représentation graphique de RdP

Les places **Pi** et les transitions **Ti** d'un réseau de Petri, en nombre fini et non nul, sont reliées par des **arcs orientés**. Un réseau de Petri est dit **graphe biparti alterné**, c'est à dire qu'il y a alternance des types de nœuds : tout arc, qui doit obligatoirement avoir un nœud à chacune de ses extrémités, relie soit une place à une transition soit une transition à une place.

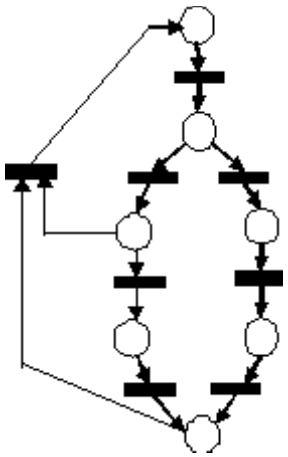

Figure 4.1: Exemple de réseau de Petri comportant 7 places, 6 transitions et 15 arcs orientés

4.2.4 Définition de l'état d'un réseau de Petri

Pour définir l'état d'un système modélisé par un réseau de Petri, il est nécessaire de compléter le réseau de Petri par un **marquage**. Ce marquage consiste à disposer un nombre entier (positif ou nul) de marques ou jetons dans chaque place du réseau de Petri. Le nombre de jetons contenus dans une place **Pi** sera noté **mi**. Le marquage du réseau sera alors défini par le vecteur **M = {mi}**.

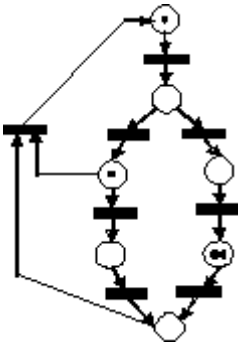

Figure 4.2: Exemple de réseau de Petri marqué avec un vecteur de marquage M : **M = (1,0,1,0,0,2,0)**.

4.2.5 Evolution temporelle d'un réseau de Petri

L'évolution d'état du réseau de Petri correspond à une évolution du marquage. Les jetons, qui matérialisent l'état du réseau à un instant donné, peuvent passer d'une place à l'autre par **franchissement** ou **tir** d'une transition. Dans le cas des réseaux dits à arcs simples ou de poids 1, la règle d'évolution s'énonce de la manière suivante :

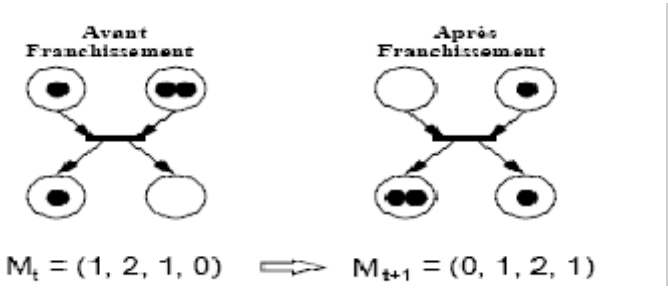

Figure 4.3: Exemple d'évolution temporelle d'un réseau de Petri **[Cld-K]**

Le franchissement d'une transition consiste à retirer un jeton dans chacune des places en amont de la transition et à ajouter un jeton dans chacune des places en aval de celle-ci.

4.2.6 Les règles générales d'évolution temporelle d'un réseau de Petri

Les règles générales d'évolution des réseaux de Petri marqué simple sont les suivantes :

A. Une transition est **franchissable** ou **sensibilisée** ou encore **validée** lorsque chacune des places en amont possède au moins un jeton.
B. Le réseau ne peut évoluer que par franchissement d'**une seule transition à la fois**, transition choisie parmi toutes celles qui sont validées à cet instant,
C. Le franchissement d'une transition est indivisible et de durée nulle.

Si une transition est validée, cela n'implique pas qu'elle sera immédiatement franchie. Ces règles introduisent en effet un certain indéterminisme dans l'évolution des réseaux de Petri, 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 **[Cld-K]**.

4.3 Les propriétés des réseaux de Petri

4.3.1 Conflits et parallélisme

A. <u>Structure potentiellement conflictuelle</u>: t1 et t2 sont en conflit structurel potentiel pour le partage des jetons de la place p. **(Figure 4.4)**

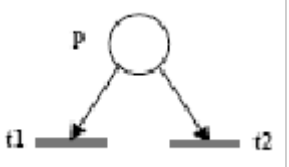

Figure 4.4: Le conflit structurel

B. **<u>Conflit relatif au marquage:</u>** Structure potentiellement conflictuelle + marquage insuffisant *t*1 et *t*2 en conflit pour le partage du jeton : non déterminisme du tir (*t*1 ou *t2*)**. (Figure 4.5)**

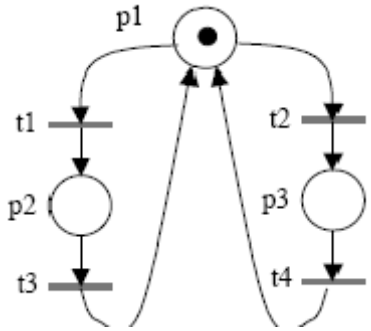

Figure 4.5: Le conflit relatif au marquage

C. **<u>Parallélisme</u>** : *t*1 et *t*2 sont tirées en parallèle **(Figure 4.6)**

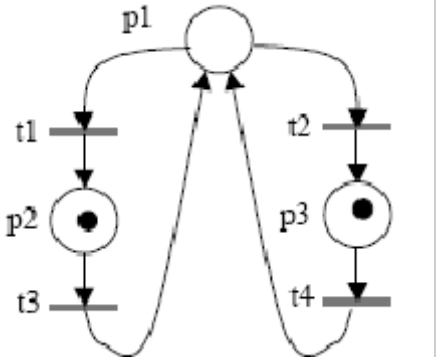

Figure 4.6: Le parallélisme sur RdP

4.3.2 Réseau propre (réinitialisable)

SSI ∀ marquage M_i accessible depuis M_0, ∃ une séquence de tirs conduisant à M_0.

4.3.3 Réseau vivant (sans blocage)

M0 → Mk

∀ ti ∃ séquence de tirs passant par ti

M0 = 1 1 0 0

M1 = 0 0 1 1

M2 = 0 1 2 0 → blocage ! **(Voir Figure 4.7**)

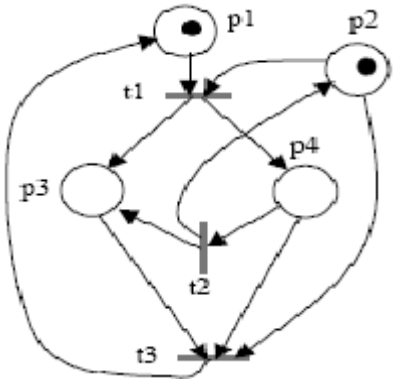

Figure 4.7: La vivacité dans les RdP

4.3.4 Réseau borné

Si ∀marquage M_i accessible depuis M_0, et ∀ $p_j \in P \rightarrow M_i(p_j) \leq MAX$.

Marquages successifs :

M0 0 0 1

M1 1 1 0

M2 0 1 1

M4 1 2 0

M5 1 2 1

M6 1 3 1 … croissance infinie

Si MAX = 1 → **Réseau de Petri Sauf**

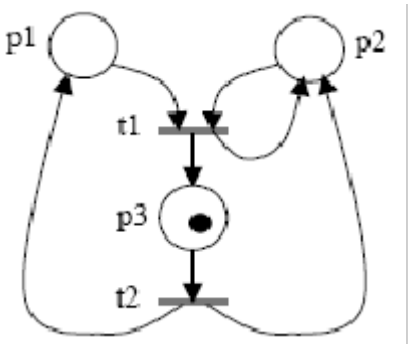

Figure 4.8: La bornitude dans les RdP

4.3.5 Réseau conforme

Réseau de Petri **vivant** et **sauf** : 1 jeton, sans blocage

4.3.6 Machine à états finie

Chaque transition n'a qu'une place amont et une place aval.

4.4 Les méthodes d'analyse des réseaux de Petri

On aborde deux méthodes d'analyse : graphe de marquages et équation de matrice. La première méthode est de construire le graphe de tous les marquages de réseau, et on déduit des propriétés grâce aux techniques de théorie de graphe. La deuxième méthode est de trouver une représentation de matrice de réseau, et on utilise des techniques d'algèbre linéaire pour obtenir des propriétés du réseau. Il existe une autre méthode ce qu'on appelle **réduction des RdPs**, elle est utilisé pour faciliter et simplifier l'analyse des RdPs par les deux méthodes précédentes.

4.4.1 Graphe de marquages

L'évolution temporelle d'un RdP peut être décrite par un graphe de marquage représentant l'**ensemble des marquages accessibles** et d'arcs correspondant aux franchissements des transitions faisant passer d'un marquage à l'autre pour un marquage initial M0.

Le graphe des marquages accessibles, notée GA(N;M0) est le graphe ayant comme sommets les marquages de R(N;M0) et tel qu'il existe un arc entre deux sommets M1 et M2 si et seulement si M1[t > M2.

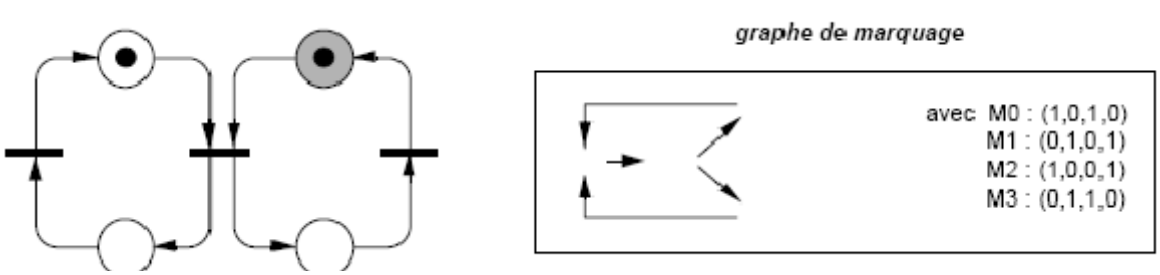

Figure 4.9: Le graphe de marquage d'un réseau de Petri **[CId-K]**

Cette représentation permet de déterminer certaines propriétés d'un graphe. Par exemple si le graphe présente une zone non bouclé, cette partie du marquage une fois atteinte constitue un arrêt de l'évolution du RdP et celui-ci sera déclaré avec blocage.

Si le graphe de marquages accessibles est fini, c'est la situation la plus favorable car alors toutes les propriétés peuvent être déduites simplement par inspection de celui-ci.

Avec cette présentation l'arbre peut-être infini si le réseau n'est pas borné. Pour l'éviter, on doit accepter de perdre des informations. Pour l'arbre infini, le nombre de jetons dans les places pouvant avoir une quantité arbitrairement grande de jetons vont être considéré comme une valeur particulières, noté par ω. Il a la propriété que pour chaque entier n,

ω + n = ω

ω - n = ω

n<= ω

ω >= ω

4.4.2 Equation de matrice

Deuxième approche pour analyser le réseau de Petri se base sur la représentation de matrice de réseau. Du réseau (N;M0) on définit D- et D+ étant deux matrices de m-ligne (une ligne pour 1 transition) et n-colume (une colonne pour une place) comme suivant :

D-[ti; pj] = W(pi; tj); D+[ti; pj] = W(ti; pj)

On note e[tj] est m-vecteur (l'élément tj correspond à la transition tj) dans lequel il n'y a que l'élément tj est égal à 1, les autres sont égales à zéro.

$$e[t_j] = (0, \ldots, 0, \underbrace{1}_{t_j}, 0, \ldots, 0)$$

Donc une transition tj est franchissable dans un marquage M si

$$M \geqslant e[t_j]D^-$$

et le résultat du franchissement de la transition tj dans le marquage M, s'il est franchissable, est

$$\begin{aligned} M' &= M - e[t_j] \cdot D^- + e[t_j] \cdot D^+ \\ &= M + e[t_j] \cdot (-D^- + D^+) \\ &= M + e[t_j] \cdot D \end{aligned}$$

où D = -D- + D+.

Pour une séquence du franchissement des transitions σ = *t1t2…tk*, on a

$$\begin{aligned} M' &= M + e[t_1] \cdot D + \ldots + e[t_k] \cdot D \\ &= M + (e[t_1] + \ldots + e[t_k]) \cdot D \\ &= M + f(\sigma) \cdot D \end{aligned}$$

Le vecteur f(σ) = *e[t1]+…+e[tk]* s'appelle le vecteur **Parikh** de la séquence *t1t2…tk*. L'élément *t* de ce vecteur est le nombre de fois de transition *t* de franchissement dans la séquence σ = *t1t2…tk.*

Cette approche nous donne un outil pour attaquer le problème d'accessibilité. Si un marquage M' est accessible à partir d'un marquage M, donc il existe une séquence σ (possible vide) du franchissement de transition qui se produit à M'. C'est-à-dire f(σ) est une solution de entière non négative, pour *x* dans l'équation de matrice :

$$M' = M + x.D$$

Donc si M' est accessible à partir de M, L'équation a une solution de entière non négative. Si cette équation n'a pas de solution, donc M' n'est pas accessible à partir de M.

4.4.3 Etude des propriétés des Réseaux de Petri par réduction

Pour l'analyse des propriétés d'un RdP, des difficultés peuvent apparaitre dans l'utilisation du graphe de marquages dans le cas d'un RdP de taille significative **[Scor]**. Dans ce cas on a présenté des règles permettant d'obtenir à partir d'un RdP marqué, un RdP marqué plus simple, c'est-`a-dire avec un nombre réduit de places et un nombre réduit de transitions (règles de réduction). Ce dernier doit être

1. pour les propriétés étudiées, équivalent au RdP marqué de départ ;
2. suffisamment simple pour que l'analyse de ses propriétés soit simple.

En général, le RdP simplifié ne peut pas s'interpréter comme le modèle d'un système. On peut distinguer deux types de règles de réduction :

1. règles de réduction préservant la vivacité et la bornitude (et leurs propriétés associées) ;
2. règles de réduction préservant les invariants de marquage.

4.5 Les extensions des réseaux de Petri

4.5.1 Réseaux de Petri autonomes

Un RdP autonome décrit le fonctionnement d'un système de façon autonome, c'est-`a-dire dont l'évolution est régi par ses propres lois. La succession des quartes saisons peut être ainsi modélisée par un RdP autonome.

4.5.1.1 Réseaux de Petri ordinaire

Tous les arcs ont des poids égaux à 1.

4.5.1.2 Réseaux de Petri à arcs étiquetés ou généralisé

Un exemple de RdP généralisé et de RdP ordinaire équivalent est présenté **Figure 4.10**. L'intérêt de l'utilisation de RdP généralisés est évident : il permet d'obtenir des modèles RdP plus concis. Les RdPs généralisés font partie des classes de modèles RdPs appelées abréviations. Ces classes de modèles RdPs permettent de représenter les mêmes systèmes que les RdPs mais avec une représentation graphique beaucoup plus concise. Une seconde classe d'abréviations est les RdPs colorés.

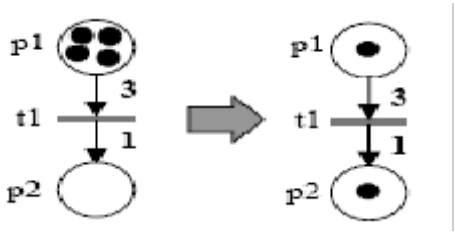

Figure 4.10: RdP à arc étiquetés

4.5.1.3 Réseau de Petri interprété

Tir de la transition si :

1) transition sensibilisée
2) prédicat vrai (ab' = 1)

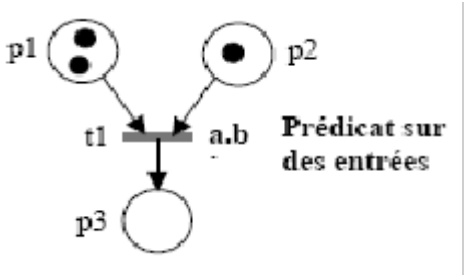

Figure 4.11: RdP interprété

4.5.1.4 Les réseaux de Petri coloré

Lorsque le nombre d'entités du système à modéliser est important, la taille du réseau de Petri devient rapidement énorme ; et si les entités présentent des comportements similaires, l'usage des réseaux colorés permet de condenser le modèle. Les réseaux de Petri colorés sont des réseaux de Petri 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 **[Kurt1 97]**. 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 Petri, ils permettent juste une condensation de l'information. A tout réseau de Petri coloré marqué correspond un réseau de Petri 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 paraport au langage assembleur; car il est bien structuré, bien typé et modulé. **[Kurt2 97]**.

Le passage du RdP **(Figure 4.12 gauche)** au RdP coloré **(Figure 4.12 droite)** est appelé **pliage** et le passage du RdP coloré au RdP dépliage.

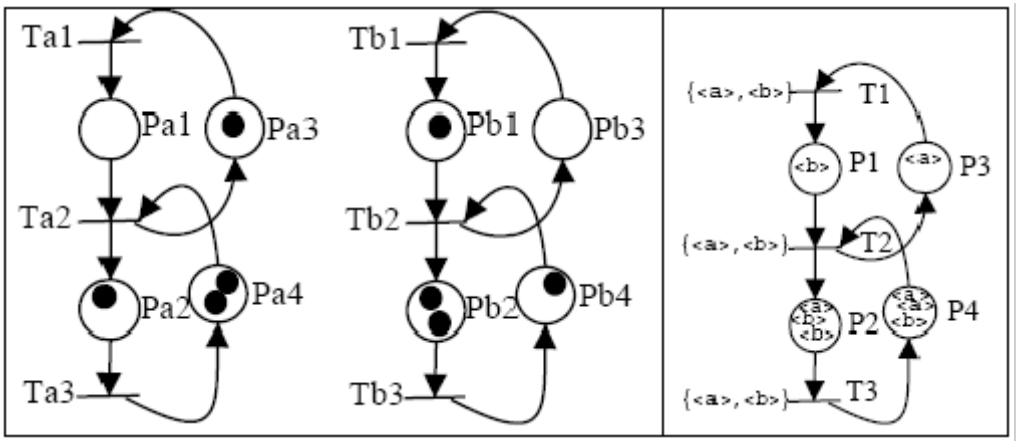

Figure 4.12: RdP (gauche) et RdP coloré (droite) **[Scor]**

4.5.2 Réseaux de Petri non autonomes

L'évolution du RdP va maintenant dépendre de l'occurrence d'événements externes ou du temps. Par simplicité, on se concentrera sur l'extension des RdPs ordinaires, même si cela peut s'appliquer aux RdPs généralisés.

4.5.2.1 Réseaux de Petri synchronisés (RdPS)

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 mais 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.

Les notions de bornitude et de vivacité peuvent se généraliser aux RdPs synchronisés. Un RdP synchronisé est borné si, pour tout marquage stable et instable, il est borné. Une transition *Tj* est vivante si pour tout marquage accessible, il existe une séquence d'événements externes telle que *Tj* soit franchie. Si on considère le RdP autonome associé au RdP synchronisé **[Scor]**.

4.5.2.2 Réseaux de Petri temporels et temporisés

Il existe deux principales familles d'extension temporelle des réseaux de Petri : les réseaux de Petri **temporisés** introduits par Ramchandani **[RAM 74]** et les réseaux de Petri **temporels** introduits par Merlin **[MER 74]**. Pour les réseaux de Petri temporisés, les temporisations ont d'abord été associées aux transitions (t-temporisés), puis aux places (p-temporisés). La temporisation représente alors la durée minimale de tir ou le temps de séjour minimum d'un jeton dans une place (ou durée exacte avec une règle de fonctionnement au plus tôt). Les RdP t-temporisés et p-temporisés sont équivalents et sont une sous-classe des réseaux de Petri temporels.

Concernant les réseaux de Petri temporels, l'extension temporelle s'exprime sous la forme d'un intervalle associé principalement aux transitions (t-temporel) **[MER 74]**, ou aux places (p-temporel). En ce qui concerne l'expressivité des réseaux de Petri p-temporels et t-temporels, ces deux modèles sont incomparables **[FrC]**. Enfin, les réseaux de Petri t-temporels forment une sous classe des *Time Stream Petri Nets* qui ont été introduits pour modéliser des applications multimédia **[FrC]**.

Le problème de la *bornitude* est indécidable pour ce type de RdP et les travaux sur ce modèle reportent les résultats de décidabilité (comme l'accessibilité) sur l'hypothèse de bornitude du réseau **[FrC]**. La bornitude (et les autres problèmes) sont alors résolus par le calcul de l'espace d'états (quand celui-ci termine).

4.5.2.3 Réseaux de Pétri stochastiques

Les réseaux de Petri stochastiques (RdPS) ont été développés. En effet, en introduisant une temporisation dans les RdP, le but est de concevoir un modèle unique permettant à la fois une validation qualitative et quantitative de réseaux.

Un RdPS est un RdP temporisé doté d'une mesure de probabilité sur l'espace des trajectoires, c'est à dire que les séquences de franchissement sont mesurables en considérant un espace aléatoire. De façon formelle :

Un RdPS est un couple S = < R, ϕ >, tel que :

- R = < P, T, Pré, Post, M_0 > est le réseau sous-jacent.

- ϕ : T $\rightarrow$ R^+, la fonction qui associe à chaque transition un taux de franchissement fixe.

- M_0 : marquage initial du réseau.

4.6 La modélisation des systèmes mobiles par les RdPs

La modélisation des systèmes adaptative et mobiles par les RdPs ordinaire ou par ses abréviations (RdPs coloré par exemple) est insuffisant car les particularités des applications mobiles et les nouveaux concepts qui sont introduits par ces applications. Dans ce cas il faut introduire des nouveaux extensions sur les RdPs correspond aux applications mobile.

Dans ce sens il existe plusieurs propositions, nous avons présenté quelque une dans ce chapitre.

4.6.1 Le paradigme (*nets-within-nets*)

Le paradigme ***nets-within-nets*** introduit par **[Koh 2007]** est un formalisme dont les jetons sont des RdPs elle-même **(Figure 4.13)**. L'avancement conceptuel de la notion de jeton (jeton simple, jeton coloré, token nets) due d'apparaître de ce paradigme pour modéliser les systèmes récursives et hiérarchiques de façon très simple, et on peut modéliser les systèmes adaptatives et mobiles pour ce type de paradigme.

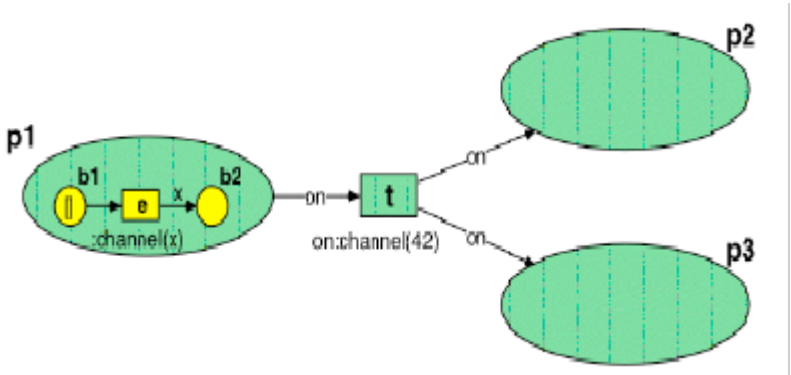

Figure 4.13: un exemple d'un RdP Nets-within-Nets**[Koh 2007]**

Le RdP Nets-within-Nets a deux niveaux d'abstraction; le niveau inférieur qui s'appelle ***System Nets*** (le RdP dont ses places contient des jetons comme des RdPs elle-même) et le niveau supérieur qui s'appelle ***object nets*** (les RdPs dont ses places contient des jetons simple). Dan la ***Figure 4.14*** la place **P1** de ***System Nets*** contient un ***Object Nets*** (Le RdP dont ses places sont **b1** et **b2**).

Le franchissement d'une transition dans le niveau inférieur peut être synchronisé par le franchissement d'une transition dans le niveau supérieur. Prend l'exemple de la ***Figure 4.14*** la transition ***t*** n'est franchissable que la transition ***e*** est franchissable. Cette synchronisation se fait par des fonctions qui s'appellent ***down-link*** et ***up-link***. ***down-link*** est la fonction qui mène la transition de niveau inférieur (System Nets) et ***up-link*** est la fonction qui mène la transition de niveau supérieur (Object Nets).

4.6.2 Le paradigme (Object Nets for Mobility)

Ce paradigme est basé sur le travail de ***Nets-within-Nets*** qu'on a déjà vu dans le paragraphe précédent, plus les concepts de paradigme ***Nets-within-Nets*** (System Nets, Object Nets, fonction de synchronisation, …etc.) des nouveaux concepts sont introduits dans ce paradigme pour modéliser les systèmes d'agent mobile. Sémantiquement on peut distingue trois type de sémantique pour les ***Object Nets*** : la sémantique de référence (***Reference Semantics***), la sémantique de valeur (***Value Semantics***) et la sémantique mobile (***New: Mobile Semantics***) **[Koh 02]**. La sémantique mobile est un bon moyen pour modéliser les applications d'agent mobile **[Koh 02]**.

4.6.3 Le paradigme (Nested nets)

Dans le paradigme ***Nested Nets*** chaque jeton peut être un réseau de Petri elle-même. Ce paradigme est basé sur les notions introduits par le paradigme **Nets-within-Nets** en se passant sur l'étudié de la sémantique valeur autre que la sémantique de référence pour modéliser les systèmes adaptatives et mobiles **[Kee]**.

Ce paradigme est une extension des RdPs coloré dont les jetons peuvent être changé leur couleur sans franchissement d'une aucune transition **[Kee]**.

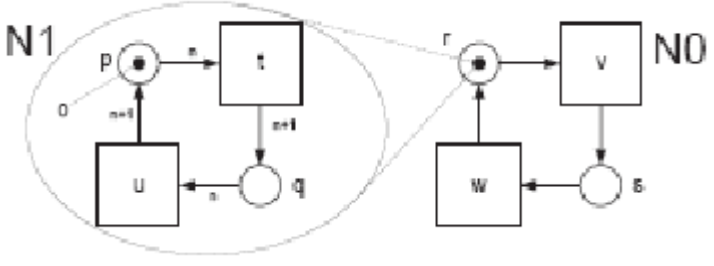

Figure 4.14: Nested Nets **[Kee]**

Il existe N niveaux d'abstraction, et pour chaque niveau le franchissement d'une transition peut être synchronisé avec le franchissement une transition de niveau supérieur, et ainsi de suit.

4.6.4 Le paradigme (*A Petri Net View of Mobility*)

Le paradigme ***(A Petri Net View of Mobility)*** proposé par Charles Lakos **[Lacos]** est basé sur les concepts suivants:

- Le système contient un nombre des RdPs qui s'appelle ***modules*** ou ***subnets*** et qui sont interconnectés par des places et transitions de fusion **(voir Figure 4.15)**;
- ***Location***: est une ***subnets*** avec une fusion de contexte;
- ***Subsystem***: est une ***location*** non marqué.

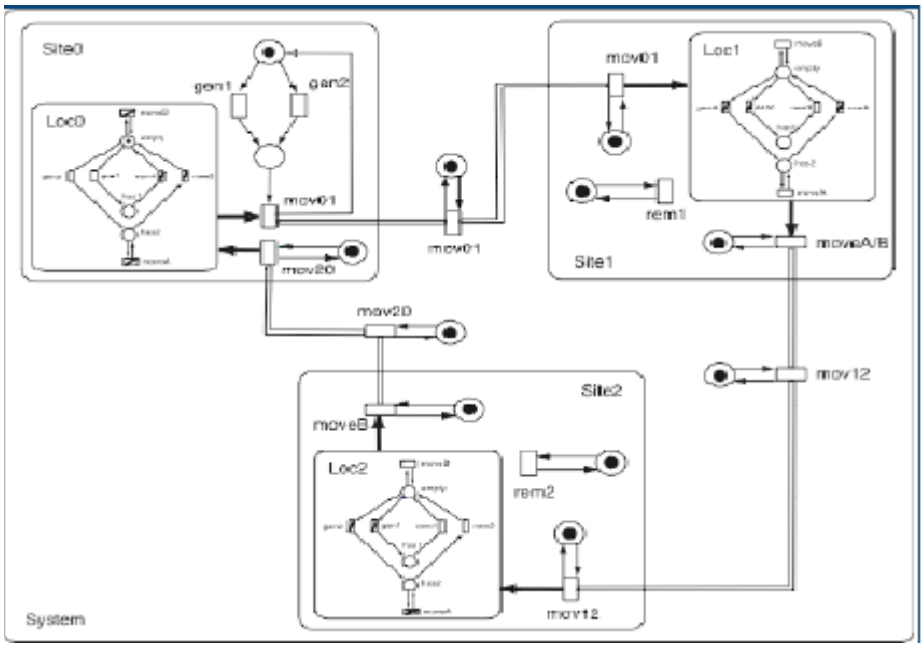

Figure 4.15: Exemple de RdP (***A Petri Net View of Mobility***)

4.7 Conclusion

La vérification et la validation de systèmes informatiques constitue aujourd'hui un enjeu crucial. C'est en particulier le cas pour les systèmes critiques dont les pannes peuvent avoir des conséquences irréversibles et dramatiques sur leur environnement (système mobile par exemple). Dans ce cadre, on a essayé de présenter le paradigme de réseau de Petri et ses extensions, en commençant par la modélisation par Réseaux de Petri dans le cadre général, ensuite les propriétés des RdPs et ses types, et finalement une brève présentation des extensions de RdP pour la mobilité.

Dans le chapitre suivant on a essayé de proposé une grammaire de graphe pour transformer automatiquement un digramme d'UML mobile vers les réseaux de Petri Nested Nets pour des raisons de vérification.

Chapitre 05: Notre approche de transformation de modèle

5.1 Introduction

Les travaux de recherche portant sur les méthodes et les outils pour l'analyse et la conception des systèmes d'agents mobiles ont aboutis à plusieurs propositions **[Adl], [Chr], [Edg01]**, **[Edg02]**, **[Har]**, **[Hach]**, **[Kas01], [Kas02]**, **[Kang], [Kus]**. Ces propositions partagent le même principe d'ajout de concepts de mobilité aux diagrammes d'UML, pour capturer les concepts propres de la mobilité tel que : la localité, la migration, le clonage ainsi que le suivit du cycle de vie de l'agent mobile. Certaines de ces propositions **[Chr]**, **[Har]**, **[Kang], [Kus]** s'intéressent particulièrement à un sous ensemble de diagrammes d'UML. Par ailleurs d'autres propositions **[Adl], [Hach], [Edg01]**,**[Edg02]** utilisent un éventail plus large de diagrammes d'UML standard ainsi que d'autres aspects. **M-UML** et une proposition **[Kas01], [Kas02]** d'enrichissement de l'ensemble des diagrammes d'UML pour la prise en charge des concepts de la mobilité. Dans notre travail on s'intéresse particulièrement au diagramme d'état transition mobile.

UML et ses extensions sont des bons outils de modélisation de n'importe quelle application dans n'importe quel domaine mais malgré la commodité de la modélisation et de la compréhension des modèles UML, la vérification de tels modèles est une tâche assez délicate, à cause de la sémantique semi-formelle d'UML. Ce problème due à l'apparition des plusieurs travaux qui s'intéresse à la vérification des modèles UML **[Varro]**, **[Cser], [Lara 01], [SAD], [Sald], [Pablo], [Guerra]**, **[Zhax]**, **[Zhao]**, **[Simo]** soit par une vérification formelle en utilisant le formalisme de réseau de Petri, soit par une transformation de modèle de certaine diagrammes d'UML vers les RdP. Par exemple Juan de Jara **[Lara 01]** est proposé une grammaire pour transformer les diagrammes d'état transition d'UML ver les RdPs.

Dans ce chapitre, nous avons utilisé l'outil **Atom3 [Atom]** pour proposer un grammaire de graphe pour transformer les diagrammes d'état transition mobile ver les RdPs Nested Nets en commençant par une proposition d'un métamodèle pour le diagramme d'état transition mobile, suivie par une autre proposition d'un métamodèle pour le formalisme de RdP Nested Nets, et enfin nous avons proposé un grammaire de graphe pour transformer le formalisme source vers le formalisme cible.

5.2 Présentation de l'outil de transformation utilisée : AToM3

AToM^3 **[Atom]** est un acronyme de : A Tool for Multi-formalism Meta-Modeling, c'est un outil de transformation de modèles, écrit dans le langage de programma-

tion **Python.** Il utilise et implémente un ensemble de concepts (la modélisation multi formalismes, le méta modélisation et les grammaires de graphes). 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 ou bien 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.

Le formalisme est automatiquement généré à partir d'un méta-modèle qui est simplement un modèle qui définit une spécification complète de la création d'un type donné de modèles. En AToM3 les méta-modèles sont spécifiés à l'aide du diagramme de classe d'UML ou même à l'aide du diagramme d'entité-association.

L'aspect de modélisation multi-formalismes de l'AToM3 est dû au fait qu'il permet de transformer un modèle dans un formalisme donné vers un autre formalisme, ceci en utilisant les grammaires de graphes. Les grammaires de graphes sont eux même des modèles qui contiennent également des éléments du premier formalisme de la transformation que ceux du deuxième formalisme, et aussi des éléments du formalisme générique. Dans le niveau de modélisation, l'AToM3 est très similaire aux différents outils de modélisation. Chaque formalisme est muni d'un ensemble de boutons pour permettre le dessin des entités sur un canevas, les entités ont des représentations graphiques qui sont définies dans le formalisme puisqu'il représente un méta-modèle. Ce qui différencie l'AToM3 des autres outils et ses capacités de méta-modélisation et de transformation de modèles.

Récapitulant l'AToM3 : c'est un outil qui possède une couche de méta-modélisation qui lui permet une modélisation graphique des différents formalismes. A partir de la méta-spécification (établie dans le modèle entité-association), AToM3 génère un outil pour la manipulation des différents modèles décrits dans le formalisme spécifié. Les modèles ont une représentation interne sous forme de graphes de syntaxes abstraites. Par conséquent, la transformation entre différents formalismes est réduite en une réécriture de graphes. Ainsi, La transformation elle-même est définie par des modèles de grammaires de graphes. Bien que les grammaires de graphes ont été utilisées dans divers domaines tels que les éditeurs graphiques, la génération de code, l'architecture des ordinateurs, etc., La **figure 5.2** présente l'interface de l'AToM3 avec une préseta-

tion d'un méta modèle, il offre un canevas pour la modélisation graphique des différentes entités.

5.3 UML mobile

5.3.1 Diagramme d'état transition mobile (Mobile Statechart diagram)

Le diagramme d'état transition d'UML consiste en des états et des relations qui relient ses états. Normalement le diagramme d'état transition fait la description du fonctionnement des acteurs en termes de changement d'états. Chaque acteur ou objet se trouvé dans un état dans un instant donnée. ***L'état mobile*** est un état dont l'acteur ou l'objet se trouve dans une plateforme qui est loin de sa plateforme de basse, graphiquement en représentant ***l'état mobile*** en ajoutant un carré qui porte le symbole 'M' dans l'extrémité haute gauche de l'état d'UML standard. Une ***transition*** est une ligne unidirectionnel reliant deux états (l'état source et l'état cible). On appel ***transition mobile*** une transition qui relie deux états d'un agent dont celle-ci se trouve dans deux plateformes différentes. Une ***transition non mobile*** peut relier deux ***états mobiles***, et une ***transition mobile*** relie deux ***états mobiles*** ou bien un ***état mobile*** à un ***état non mobile*** et vis versa. Graphiquement une ***transition mobile*** est représentée comme une transition standard en ajoutant un petit carré portant le symbole 'M' au milieu de cette transition. De plus, si un agent arrive dans un état alors en ajoute un petit carré pointillé portant le symbole 'M' dans l'extrémité haute gauche de celle-ci pour indiquer l'état actuel. De la même façon, si un agent arrive dans un état et réalise une interaction à distante avec un autre agent alors en ajoute un petit carré portant le symbole 'R' dans l'extrémité haute gauche de cette transition. Finalement une transition mobile stéréotypé ***«agentreturn»*** correspond au retour d'un agent vers sa plateforme de base avec un changement d'état.

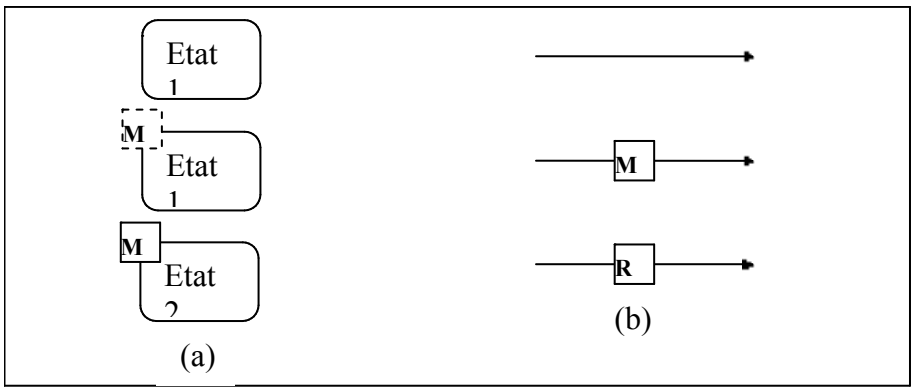

Figure 5.1: Concept du graphe d'état transition mobile
(a): représentation graphique des états ; (b): représentation graphique des transitions.

5.3.2 Méta-modèle du diagramme d'état transition mobile

Notre méta-modèle est composé de 5 classes et 9 transitions (**Figure 5.2**):

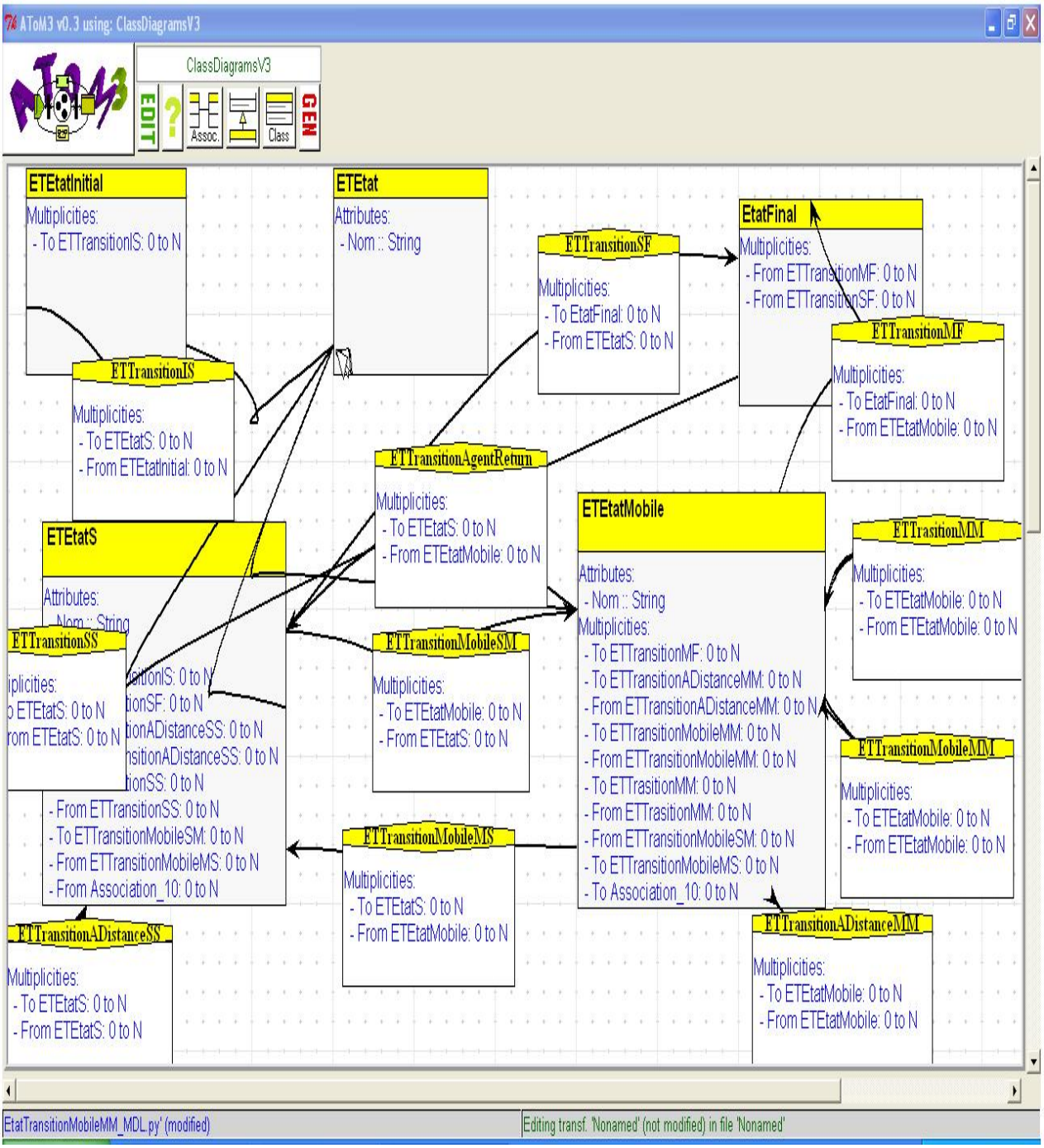

Figure 5.2: Méta-model pour le diagramme d'état transition mobile.

- **Classe Etat** : cette classe représente l'état dans le cas général d'un diagramme d'état transition mobile. Elle possède un attribut « ***nom*** » de type **String** pour designer le nom de l'état. Cette classe est considérée comme la classe mère

des classes suivantes: la classe ***Etat initial***, la classe ***Etat final***, la classe ***Etat simple*** et la classe ***Etat mobile***.

- **Classe Etat initial** : Cette classe représente l'état initial, elle hérite la ***classe Etat*** et elle se représente graphiquement par un petit cercle plain noir. Elle est connecté avec la ***classe Etat simple*** par ***l'association TransitionIS***, Cette association possède les attributs **PFS** et **PFD** de type String pour designer respectivement la plateforme source et la plateforme destination. L'attribut **Evenement**s de type String est utilisé pour déterminer l'événement déclencheur de cette transition, et l'attribut **Activites** de type String aussi est utilisé pour déterminer les activités produites après l'exécution de cette transition. L'association ***TransitionIS*** relie une seule instance de la classe ***Etat initial*** à une seule instance de la classe ***Etat simple***. Elle est représentée graphiquement par une flèche de couleur noir portant tous les attributs.
- **Classe Etat mobile** : cette classe représente l'état mobile, elle hérite les attributs de la ***classe Etat*** et elle se représente graphiquement par un rectangle blanc qui porte à son extrémité haute gauche un petit carré marqué par 'M', mais dans le cas où cet état est l'état actuel elle se représente par un rectangle blanc qui porte à son extrémité haute gauche un petit carré pointillé marqué par 'M' **(Figure 3**). Cette classe se connectée à la ***classe Etat simple*** par l'association ***TransitionMobileMS*** si une transition mobile existe entre cet état et un état simple, et par l'association ***TransitionAGR*** s'il existe une transition mobile qui modélise le retour de l'agent mobile de la plateforme (transition **«agentreturn»**). Ces associations possèdent comme toutes les associations du méta model, les attributs **PFS, PFD, Evenements et Activites.** Elles relient une seule instance de la classe ***Etat mobile*** à une seule instance de la classe ***Etat simple***, et elles sont représentées graphiquement par une flèche de couleur noir portant tous les attributs en ajoutant un petit carrée marqué par "M" pour indiquer que ces transitions sont mobiles. De plus, nous ajoutons un stéréotype **«agentreturn»** pour l'association ***TransitionAGR*** afin de modélise le retour de l'agent mobile vers sa plate forme de basse.

  La classe ***Etat mobile*** est reliée à elle-même par trois associations:

  -l'association ***TransitionMM*** modélise la transition simple entre deux états mobiles, et est représentée graphiquement par une flèche noire portant tous

les attributs. Cette association relie une seule instance de la classe ***Etat mobile*** à une seule instance d'elle-même.

- la deuxième association ***TransitionMobile***MM modélise la transition mobile entre deux états mobiles. Elle se représente graphiquement par une flèche noire portant tous les attributs, en ajoutant un petit carrée marqué par "M" pour indiquer que la transition est mobile. cette association relie une seule instance de la classe ***Etat mobile*** à une seule instance d'elle-même.

- la dernière association ***TransitionADistanceMM*** modélise la transition à distance entre deux états mobiles. Elle est représentée graphiquement par une flèche noire qui porte tous les attributs en ajoutant un petit carrée marqué par **"R"** pour indiquer la transition à distance. Cette association relie une seule instance de la classe ***Etat mobile*** à une seule instance d'elle-même. La classe ***Etat mobile*** est connectée à la ***classe Etat final*** par ***l'association TransitionMF***, Cette association possède les attributs **PFS, PFD, Evenement**s et **Activites**. L'association ***TransitionMF*** relie une seule instance de la classe ***Etat mobile*** à une seule instance de la classe ***Etat final***, et est représentée graphiquement par une flèche de couleur noir portant tous les attributs comme toute relation simple.

- **Classe Etat Simple:** Cette classe représente l'état simple (l'état où l'agent mobile se trouve dans sa plateforme de base) de l'agent mobile, elle hérite les attributs et les relations de la ***classe Etat.*** Elle est représentée graphiquement par un rectangle blanc **(Figure 3**), de plus un petit carré pointillé marqué par 'M' est ajouté à l'extrémité haute gauche du rectangle pour designer l'état actuel (la plateforme qui possède l'agent mobile à l'instant courant). Cette classe est connectée à la classe ***Etat mobile*** par l'association ***TransitionMobileSM*** qui possède aussi les attributs **PFS, PFD, Evenement**s et **Activites**. Cette association relie une seule instance de la classe ***Etat simple*** à une seule instance de la classe ***Etat mobile*** et est représentée graphiquement comme toutes les transitions mobiles dans le méta model.

  La classe ***Etat simple*** est connectée avec elle-même par deux associations:

  -l'association ***TransitionSS*** modélisant la transition simple entre deux états simple, elle est représente graphiquement par une flèche noire portant tous les attributs. Cette association relie une seule instance de la classe ***Etat simple*** à une seule instance d'elle-même.

- la deuxième association ***TransitionADistanceSS,*** modélise la transition à distance entre deux états simples, et est représentée graphiquement par une flèche noire portant tous les attributs en ajoutant un petit carrée marqué par **"R"** pour montrer qu'il sagit d'une transition à distance. Cette association relie une seule instance de la classe ***Etat simple*** à une seule instance d'elle-même. La classe ***Etat simple*** est connectée avec la classe ***Etat final*** par ***l'association TransitionSF***, Cette association possède les attributs **PFS, PFD, Evenement**s et **Activites**. L'association ***TransitionMF*** relie une seule instance de la classe ***Etat simple*** à une seule instance de la classe ***Etat final***, et est représentée graphiquement par une flèche de couleur noir portant tous les attributs.

- **Classe Etat final:** Cette classe représente l'état final, elle hérite la ***classe Etat*** et elle est représentée graphiquement par un cercle de fond noir inscrit à l'intérieur d'un anneau de même couleur.

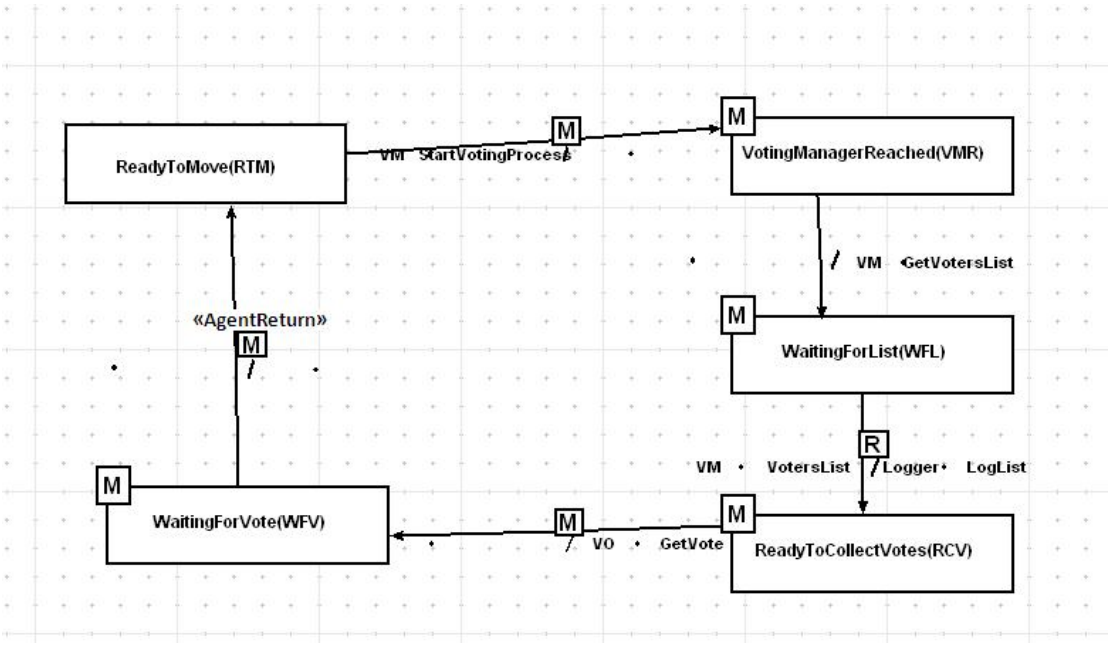

Figure 5.3: Outil de modélisation des modèles diagramme d'état transition

5.4 Les réseaux de Petri Nested Nets

5.4.1 Les réseaux de Petri Nested Nets

Les réseaux de Petri (RDPs) introduisent la possibilité d'avoir un ensemble infini d'états et une notion de localité où un système est considéré composé d'un ensemble de sites.

Dans ce travail on a choisi le paradigme ***Nested Nets*** comme une bonne manière de représenter les concepts de la mobilité comme la localité, la migration, le clonage ainsi que le suivit du cycle de vie de l'agent mobile. Comme nous avons déjà vu dans le chapitre précédant il existe **n** (n ϵ N) niveaux d'abstraction pour un réseau de Petri Nested Nets, et pour ce travail on a choisi deux niveaux d'abstraction; le niveau 0 est utilisé pour modéliser les locations, la migration et le clonage des agents mobiles et le niveau 1 est utilisé pour modéliser le cycle de vie de l'agent mobile.

Le Figure (**Figure 5.4**) illustre le franchissement d'une transition de niveau 0 (transition **T1**) qui est synchronisé avec la transition **r1** de niveau 1 par la fonction de synchronisation **f1**.

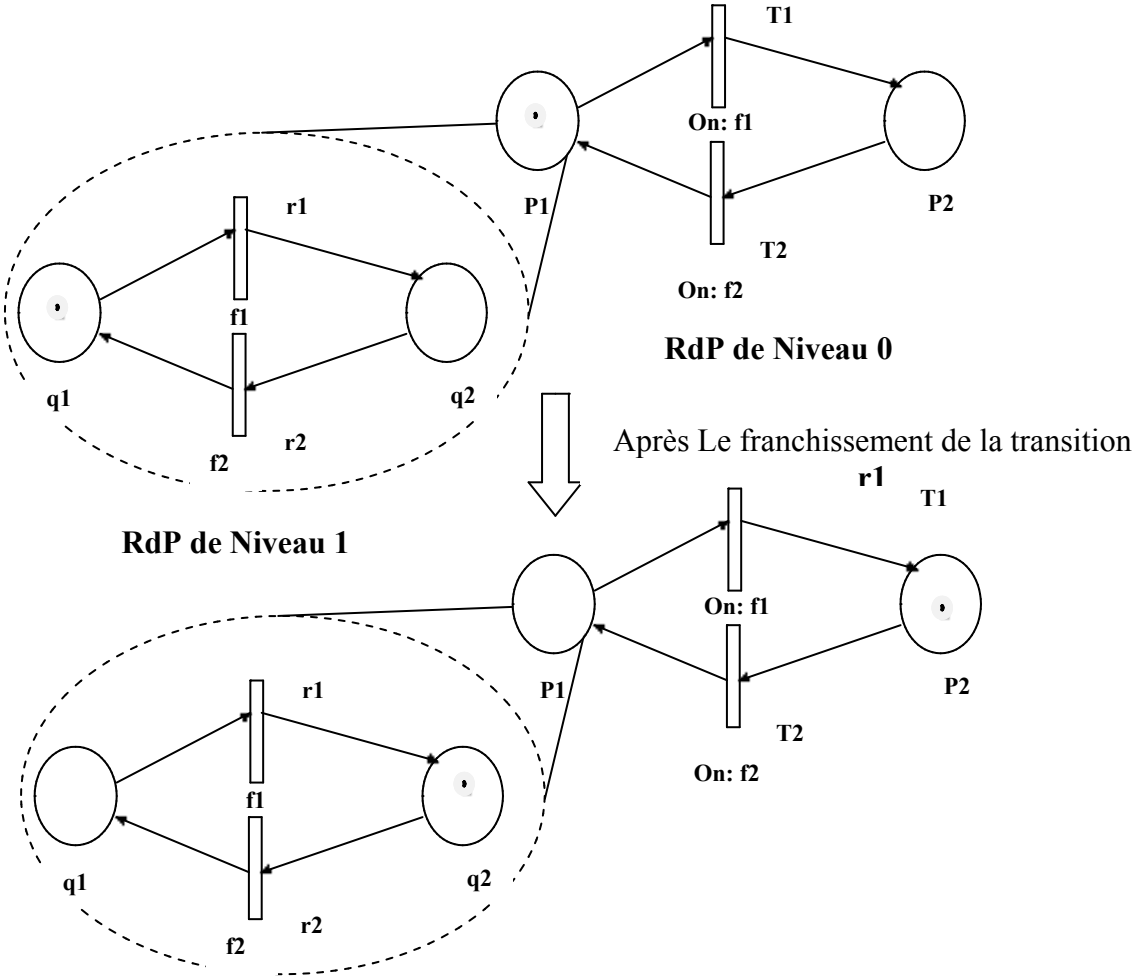

Figure 5.4: Exemple de franchissement d'une transition dans le RdP Nested Nets

5.4.2 Méta-modèle du Nested Nets

Notre méta-modèle est composé de six classes et six transitions (**Figure 5.5**):

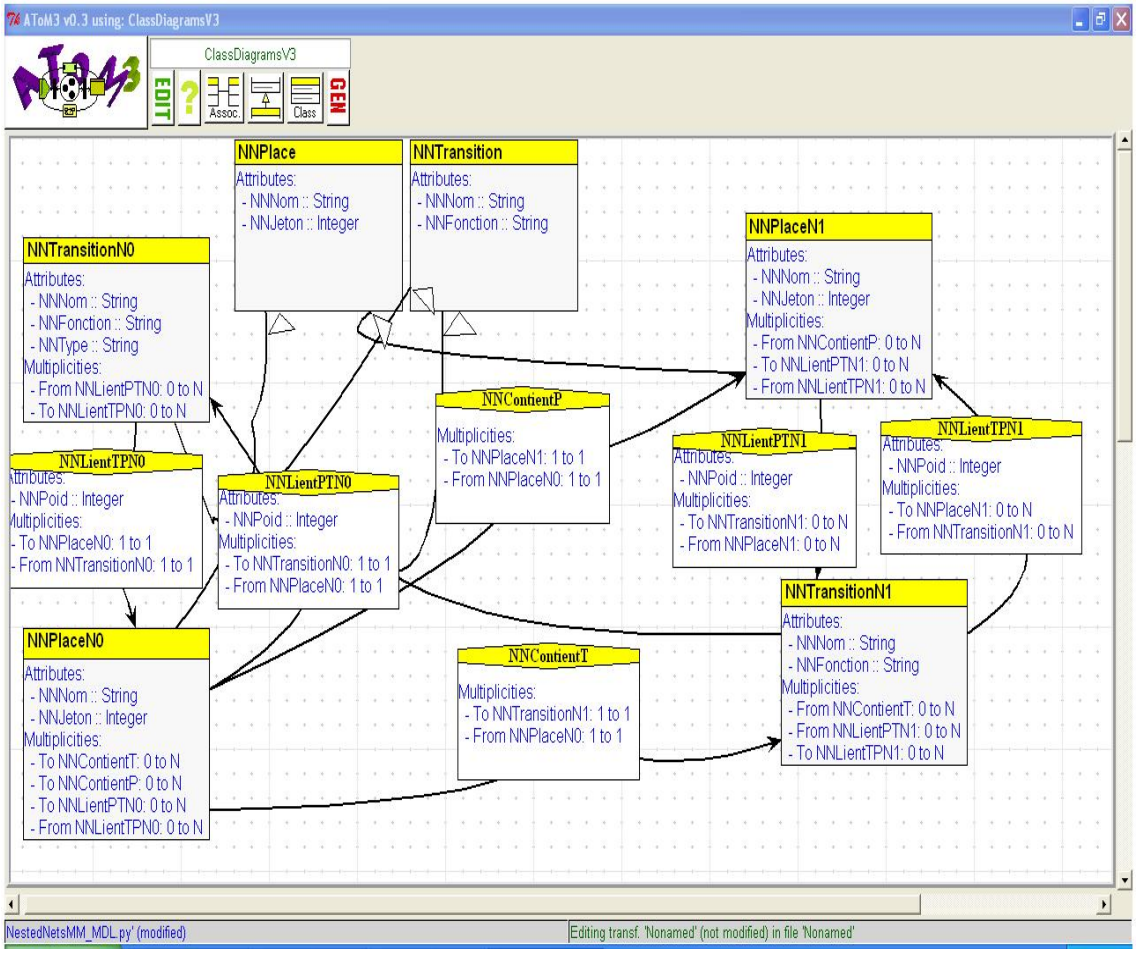

Figure 5.5: le méta-model pour les Nested Nets.

- **Classe Place:** Cette classe représente la place du réseau de Petri dans le cas général, elle possède un attribut ***nom*** de type **String** pour déterminer le nom de la place et un attribut ***jeton*** de type **entier** pour déterminer le nombre de jeton dans une place donnée. Elle est considérée comme la classe mère des classes suivantes: la classe ***PlaceN0*** et la classe ***PlaceN1***.

- **Classe Transition:** cette classe représente la transition dans le réseau de Petri dans le cas général. Elle possède deux attributs, le premier s'appelle ***nom*** et est utilisé pour déterminer le nom de la transition, le deuxième s'appelle ***Fonction*** et est utilisé pour déterminer la fonction de synchronisation entre les différents niveaux du réseau de Petri Nested Nets. La classe Transition est considérée comme la classe mère des classes ***TransitionN0*** et ***TransitionN1***.

- **Classe PlaceN0** : Cette classe représente la place dans le réseau de Petri de niveau 0, elle est utilisée pour modéliser les différentes plateformes dans le système d'agent mobile, et hérite les attributs de la classe ***Place***. Graphiquement

cette classe est représentée par un grand cercle orange qui inscrit l'attribut ***nom*** en haut à droite du cercle et l'attribut ***jeton*** au milieu. Elle est connectée à la classe ***PlaceN1*** par l'association ***ContientP*** et à la classe ***transitionN1*** par l'association ***contientT***. Ces associations sont représentées graphiquement par une fine ligne rouge. De plus, il existe une association appelée ***LientPTN0*** qui relie une seule instance de la classe ***PlaceN0*** à une seule instance de la classe ***TransitionN0***, cette association possède le seul attribut ***Poids*** de type entier utilisé pour déterminer le poids de ce lient. L'association ***LientPTN0*** est représentée graphiquement par une flèche noir portant l'attribut ***Poids.***

- **Classe TransitionN0**: cette classe représente la transition dans le réseau de Petri de niveau 0, elle hérite les attributs de la classe ***Transition*** et elle possède un autre attribut qui détermine le type de la transition de niveau N0, cet attribut s'appelle ***Type*** et il existe trois type de transition de niveau 0:

  - mobile: Ce type de transition est utilisé pour modéliser le déplacement de l'agent mobile d'une plateforme à une autre. Dans ce type de transition la classe ***transitionN0*** se représente graphiquement par un grand rectangle de couleur marron portant un petit carré marqué par le symbole 'M', l'attribut ***Nom*** est présenté en police gras situé en haut à droite du rectangle, par contre, l'attribut ***Fonction*** est présenté en police normal situé a droite de ce rectangle.

  - ***a distance:*** Ce type de transition est utilisée dans le cas où l'agent mobile fait un appelle à distance entre deux plateformes différentes. Dans ce cas, la classe ***transitionN0*** est représentée graphiquement avec la même représentation dans la transition ***mobile,*** en remplaçant le petit carré marqué par le symbole 'M' par un petit carré marqué par le symbole 'R'.

  - ***agent reteurn***: Ce type de transition est utilisé dans le cas où l'agent mobile fait un retour vers sa plateforme de base. Dans ce cas la classe ***transitionN0*** est représente graphiquement avec la même représentation dans la transition ***mobile,*** en remplaçant le petit carré marqué par le symbole 'M' par un petit carré marqué par le symbole 'AR'.

  La classe ***TransitionN0*** est connectée avec la classe ***PlaceN0*** par l'association ***LientTPN0***. Cette association relie une seule instance de la classe ***TransitionN0*** avec une seule instance de la classe ***PlaceN0***. L'association

LientTPN0 possède le seul attribut ***Poids*** de type entier utilisé pour déterminer le poids du lient. Cette association est représentée graphiquement par une flèche noir portant l'attribut ***Poids.***

- **Classe PlaceN1:** cette classe représente la place dans le réseau de Petri de niveau1, elle est utilisée pour modéliser le comportement d'un agent ou la communication entre plusieurs agents dans le système. Elle hérite les deux attributs de la classe ***Place***, et est représentée graphiquement par un petit cercle jaune où l'attribut ***nom*** de cette place est situé en haut à droite du cercle et l'attribut ***jeton*** est situé au le milieu. Cette classe se connectée à la classe ***TrasitionN1*** par l'association **LientPTN1** qui relie une seule instance de la classe ***PlaceN1*** à une seule instance de la classe ***TransitionN***1, cette association possède le seul attribut ***Poids*** de type entier utilisé pour déterminer le poids du lient, l'association ***LientPTN0*** est représentée graphiquement par un flèche noir portant l'attribut ***Poids.***

  Classe TransitionN1: cette classe représente la transition dans le réseau de Petri de niveau 1, elle hérite les attributs de la classe ***Transition***, et est représente graphiquement par un petit rectangle de couleur jaune, l'attribut ***Nom*** est présenté en police gras situé en haut à droite du rectangle par contre l'attribut ***Fonction*** est présenté en police normal situé à droite de ce rectangle. La classe ***TransitionN1*** est connectée à la classe ***PlaceN1*** par l'association ***LientTPN1***. Cette association relie une seule instance de la classe ***TransitionN1*** à une seule instance de la classe ***PlaceN***1, l'association ***LientTPN1*** possède le seul attribut ***Poids*** de type entier utilisé pour déterminer le poids du lient. Cette association est représentée graphiquement par une flèche noir portant l'attribut ***Poids.***

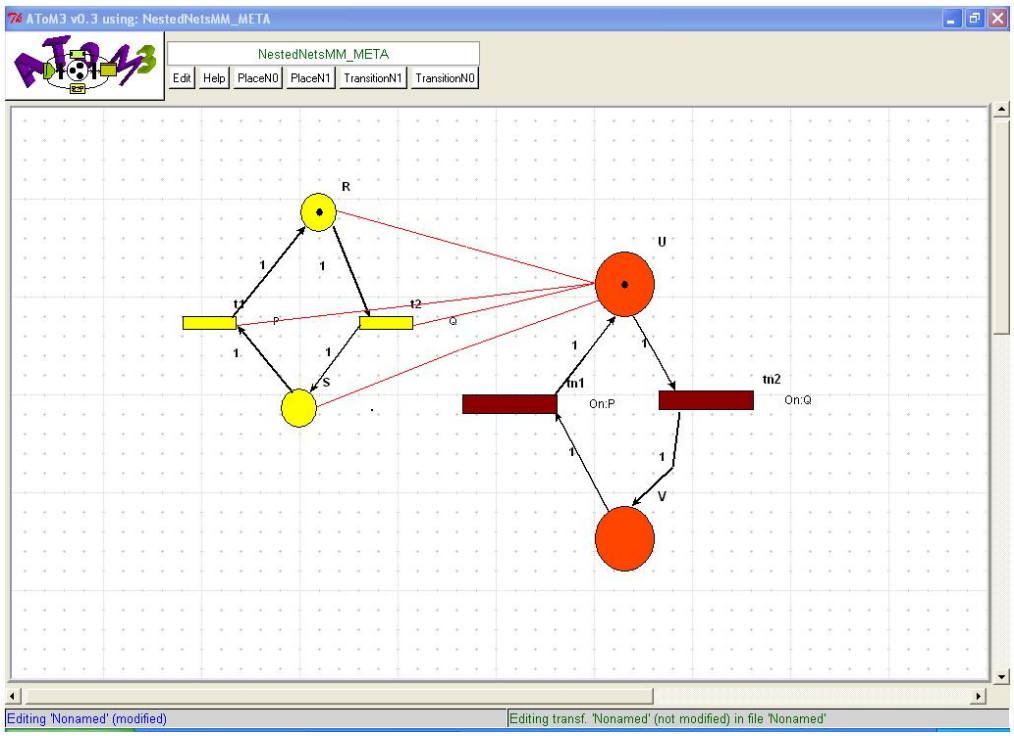

Figure 5.6: Outil de modélisation des modèles diagramme d'état transition

5.5 La grammaire du graphe proposé

Notre grammaire (grammaire **EtatTransitionMobileVersNestedNets**) est composée de trente-huit règles réparties en six catégories.

5.5.1 Les règle de transformation de l'ensemble des états en places : cette catégorie regroupe six règles qui visent à transformer les états existant dans le modèle source.

Règle 1 : **Transformation d'un état simple**

- **Nom** : EtatSToPlaces
- **Priorité** : 1
- **Rôle** : cette règle permet de transformer un état simple du diagramme d'état transition vers deux places **(Figure 5.7)**. La première place de niveau 1 représente un état d'un agent et porte le nom de cet état. La deuxième représente la plateforme qui comporte l'agent se trouvant dans cet état, cette place de niveau 0 porte un nom par défaut à cet instant. Le nombre de jetons sur les deux places (niveau 1 et niveau 0) est égale a 0 car cet état n'est pas considère comme un état actuel.

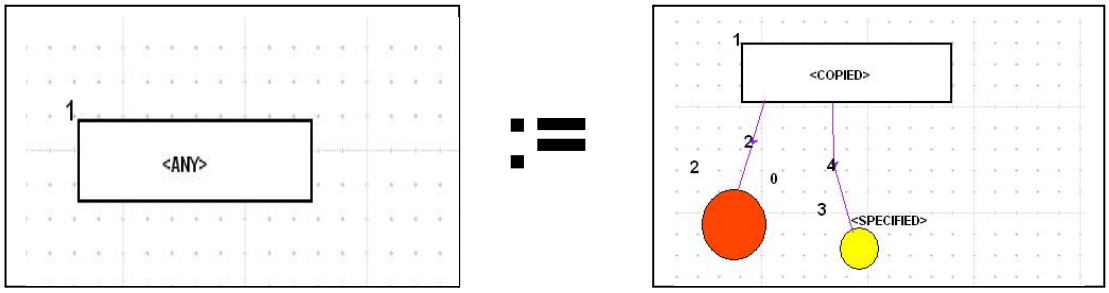

Figure 5.7: Transformation d'état simple

Règle 2 : **Transformation d'un état actuel simple**

- **Nom** : EtatSAToPlaces
- **Priorité** : 1
- **Rôle** : cette règle permet de transformer un état actuel simple (l'état de l'agent dans cet instant) du diagramme d'état transition vers deux places (**Figure 5.8)**. La première place de niveau 1 représente un état d'un agent et elle porte le nom de cet état. La deuxième représente la plateforme qui comporte l'agent se trouvant dans cet état, cette place de niveau 0 porte un nom par défaut à cet instant. Le nombre de jetons sur les deux places (niveau 1 et niveau 0) est égale a 1 par ce que cet état est considéré comme un état actuel.

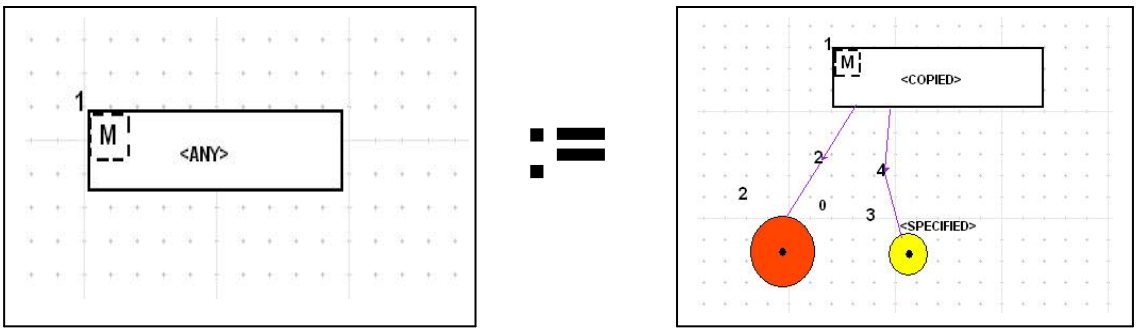

Figure 5.8: Transformation d'état actuel simple

Règle 3 : **Transformation d'un état mobile**

- **Nom** : EtatMToPlaces
- **Priorité** : 1
- **Rôle** : cette règle permet de transformer un état mobile du diagramme d'état transition vers deux places (**Figure 5.9)**. La première place de ni-

veau 1, représente un état d'un agent et elle porte le nom de cet état. La deuxième représente la plateforme qui comporte l'agent se trouvant dans cet état, cette place de niveau 0 porte un nom par défaut à cet instant. Le nombre de jetons sur les deux places (niveau 1 et niveau 0) est égale a 0 par ce que cet état n'est pas considéré comme un état actuel.

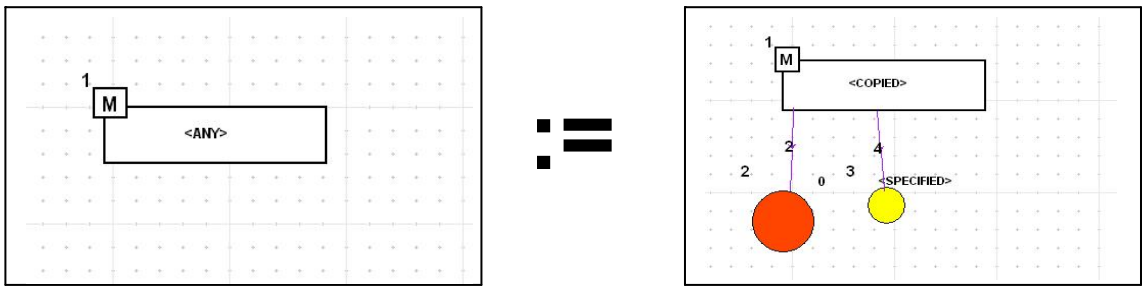

Figure 5.9: Transformation d'état mobile

Règle 4 : **Transformation d'un état actuel mobile**

- **Nom** : EtatMAToPlaces
- **Priorité** : 1
- **Rôle** : cette règle permet de transformer un état actuel mobile du diagramme d'état transition vers deux places (**Figure 5.10**). La première place de niveau 1, représente un état d'un agent et elle porte le nom de cet état. La deuxième représente la plateforme qui comporte l'agent se trouvant dans cet état, cette place de niveau 0 porte un nom par défaut à cet instant. Le nombre de jetons sur les deux places (niveau 1 et niveau 0) est égale a 1 par ce que cet état est considéré comme un état actuel.

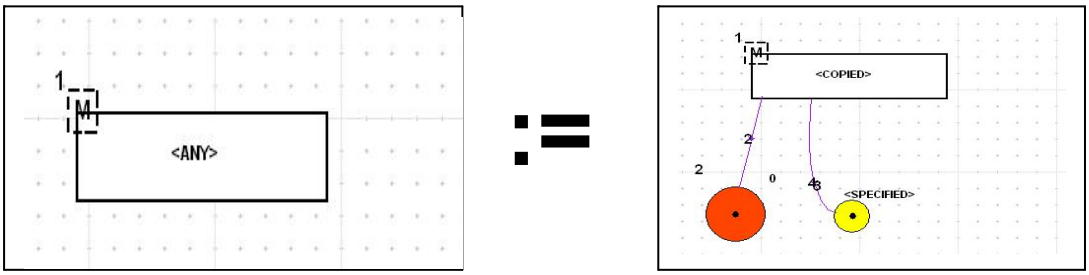

Figure 5.10: Transformation d'état actuel mobile

Règle 5 : **Transformation d'un état initial**

- **Nom** : EtatIToTransitionN1
- **Priorité** : 1

- **Rôle** : cette règle permet de transformer un état initial du diagramme d'état transition vers une transition de niveau N1 (**Figure 5.11)**. Cette transition porte le nom initial.

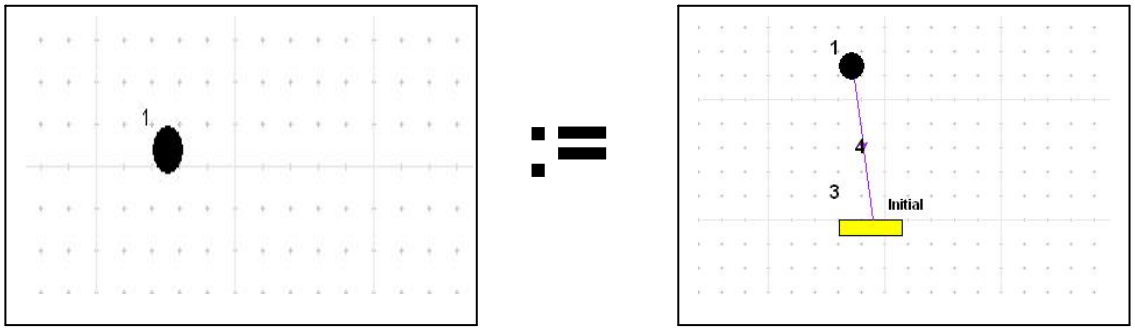

Figure 5.11: Transformation d'état initial

Règle 6 : **Transformation d'un état final**

- **Nom** : EtatFToTransitionN1
- **Priorité** : 1
- **Rôle** : cette règle permet de transformer un état final du diagramme d'état transition vers une transition de niveau N1 (**Figure 5.12)**. Cette transition porte le nom final.

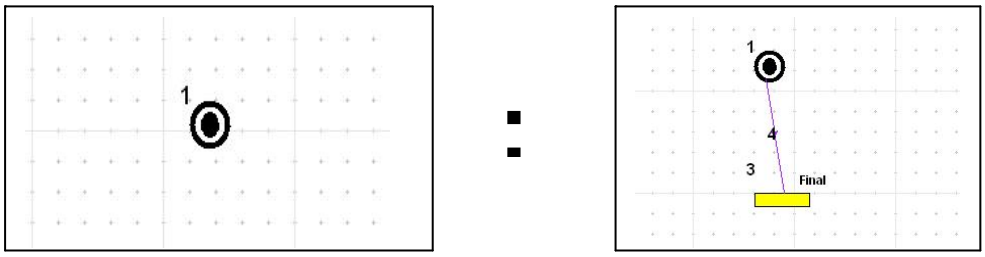

Figure 5.12: Transformation d'état final

5.5.2 Les règles de transformation de l'ensemble des transitions vers des transitions de niveaux 1: cette catégorie regroupe treize règles qui visent à transformer les transitions du modèle source vers des transitions de niveau 1 dans le model de destination.

Règle 7 : **Transformation d'une transition simple entre deux états simples**

- **Nom** : EtatSToEtatSR
- **Priorité** : 2

- **Rôle** : cette règle permet de transformer une transition simple entre deux états simples, du diagramme source (diagramme d'Etat Transition Mobile), vers une transition de niveau 1 dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation des attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.13)**. Dans ce cas, il n'existe aucune fonction de synchronisation car il n'ya pas de déplacement de l'agent d'une plateforme à une autre (les deux états se trouvent dans la même plateforme). De plus, on met le nombre de jetons dans la deuxième place (le nœud 5 dans le LHS) à 0, car si l'agent se trouve dans cette plateforme à cet instant et d'après la **règle**2 le nombre de jetons dans les deux places (nœud 4 et 5 dans LHS) sont marqués à 1, alors on élimine le jeton de la deuxième place (nœud 5 dan LHS), par ce que l'agent ne peut se trouver à un instant donnée que dans un seul état.

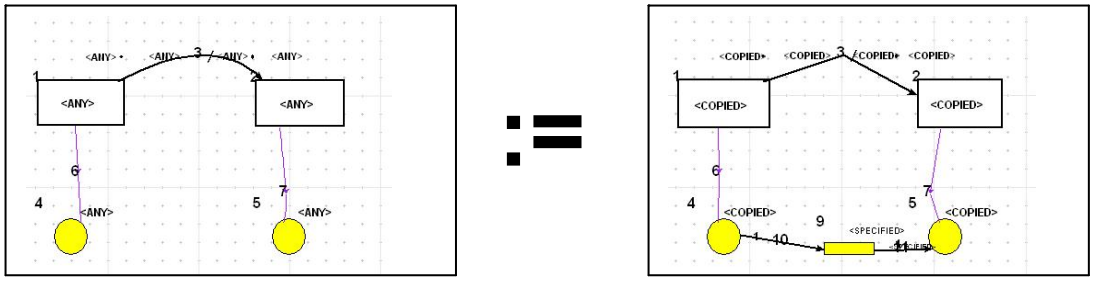

Figure 5.13: Transformation de transition simple entre deux états simples

<u>Règle 8</u> : **Transformation d'une transition simple entre deux états mobiles**

- **Nom** : EtatMToEtatMR
- **Priorité** : 3
- **Rôle** : cette règle permet de transformer une transition simple entre deux états mobiles du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 1 dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation des attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.14)**. Dans ce cas il n'existe aucune fonction de synchronisation car il n'ya pas de déplacement de l'agent d'une plateforme à une autre (les deux états se trouvent dans la même plateforme).

De plus, on met le nombre de jetons dans la deuxième place (le nœud 6 dans le LHS) à 0, car si l'agent se trouve dans cette plateforme à cet instant et d'après la **règle** 4 le nombre de jetons dans les deux places (nœud 4 et 6 dans LHS) sont marqués à 1, alors on élimine le jeton de la deuxième place (nœud 6 dan LHS) par ce que l'agent ne peut se trouver à un instant donnée que dans un seul état.

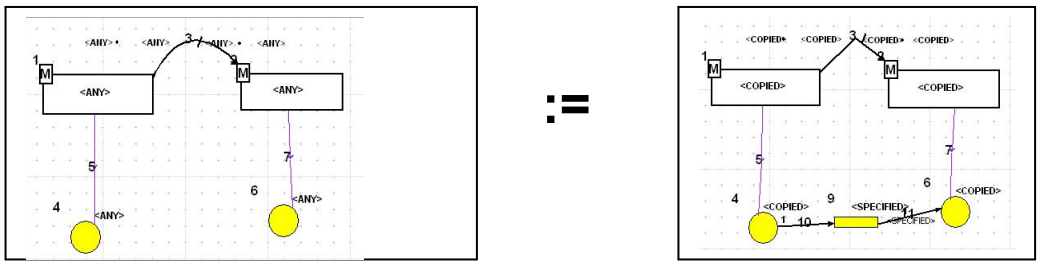

Figure 5.14: Transformation de transition simple entre deux états mobile

<u>Règle 9</u> : Transformation d'une transition mobile entre un état simple et un état mobile

- **Nom** : EtatSToEtatMR
- **Priorité** : 4
- **Rôle** : cette règle permet de transformer une transition mobile entre un état simple et un état mobile du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 1 dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation des attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.15)**. La fonction de synchronisation prend comme nom la concaténation des attributs suivants de la transition du diagramme source: Evénements et Activités

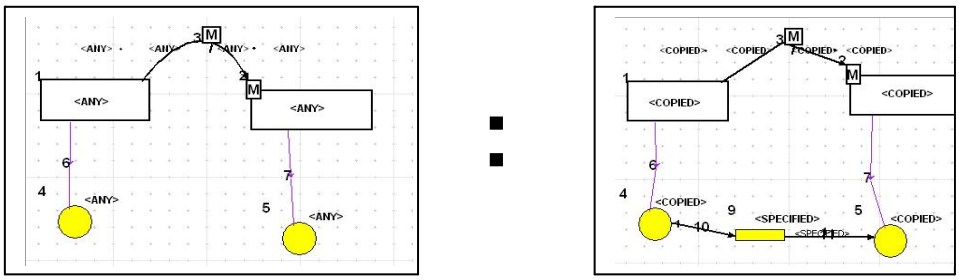

Figure 5.15: Transformation de transition mobile entre un état simple et un état mobile

Règle 10 : **Transformation d'une transition mobile entre un état mobile et un état simple**

- **Nom** : EtatMToEtatSR
- **Priorité** : 5
- **Rôle** : cette règle permet de transformer une transition mobile entre un état mobile et un état simple du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 1 dans le Nested Nets. cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.16)**. La fonction de synchronisation prend comme nom, à partir de la transition du diagramme source, la concaténation des attributs suivants : Evénements et Activités.

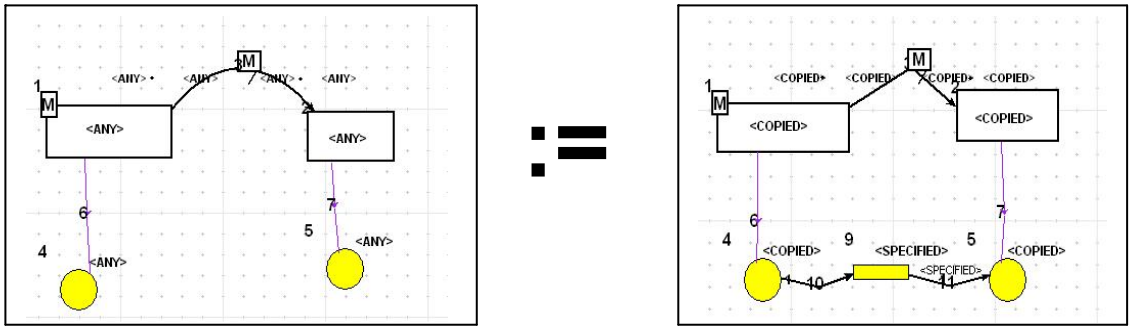

Figure 5.16: Transformation de transition mobile entre un état mobile et un état simple

Règle 11 : **Transformation d'une transition mobile entre deus états mobiles**

- **Nom** : EtatMToEtatM
- **Priorité** : 5
- **Rôle** : cette règle permet de transformer une transition mobile entre deux états mobiles du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 1 dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.17)**. La fonction de synchronisation prend comme nom, à partir de la transition du diagramme source, la concaténation des attributs suivants: Evénements et Activités.

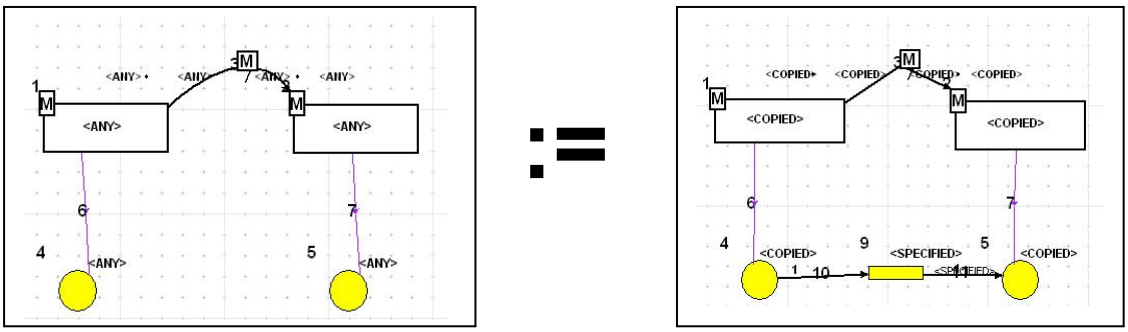

Figure 5.17: Transformation de transition mobile entre deux états mobiles

Règle 12 : **Transformation d'une transition à distance entre deus états simples**

- **Nom** : EtatSToEtatSRR
- **Priorité** : 6
- **Rôle** : cette règle permet de transformer une transition à distance entre deux états simples du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 1 dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.18)**. La fonction de synchronisation prend comme nom, à partir de la transition du diagramme source, la concaténation des attributs suivants: Evénements et Activités.

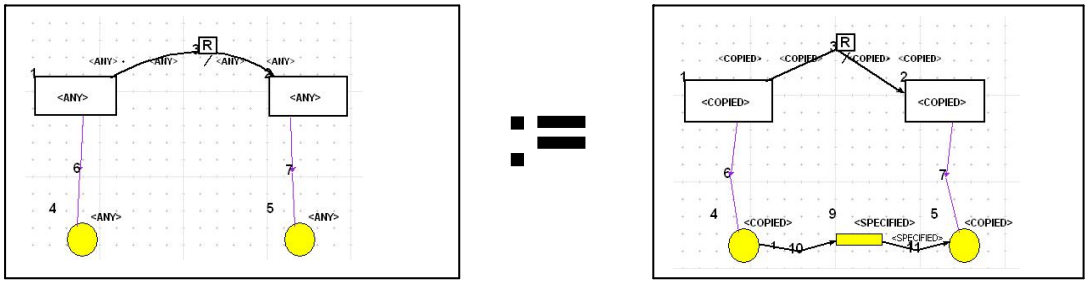

Figure 5.18: Transformation de transition à distance entre deux états simples

Règle 13 : **Transformation d'une transition à distance entre deus états mobiles**

- **Nom** : EtatMToEtatMRR
- **Priorité** : 7
- **Rôle** : cette règle permet de transformer une transition à distance entre deux états mobiles du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 1 dans le Nested Nets. Cette tran-

sition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs: PFS, PFD, Evénements et Activités (**Figure 5.18)**. La fonction de synchronisation prend comme nom, à partir de la transition du diagramme source, la concaténation des attributs suivants: Evénements et Activités.

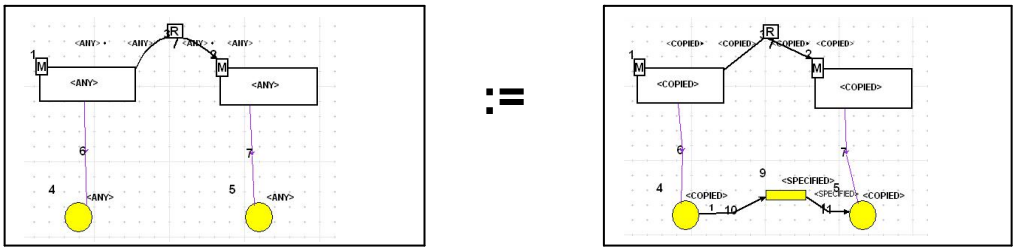

Figure 5.19: Transformation de transition à distance entre deux états mobiles

<u>Règle 14</u> : **Transformation d'une transition mobile «agent return» entre un état mobile et un état simple**

- **Nom** : EtatMToEtatSAG
- **Priorité** : 9
- **Rôle** : cette règle permet de transformer une transition mobile stéréotypée «agentReturn» entre un état mobile et un état simple du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 1 dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.20)**. La fonction de synchronisation prend comme nom, à partir de la transition du diagramme source, la concaténation des attributs suivants: Evénements et Activités

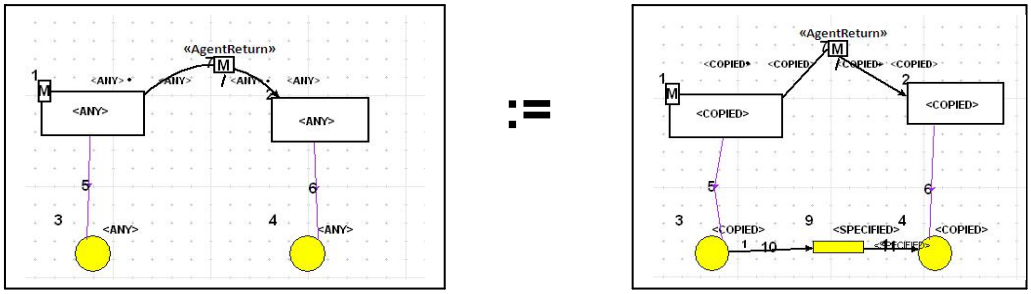

Figure 5.20: Transformation d'une transition mobile «agent return» entre un état mobile et un état simple

<u>Règle 15</u> : **Transformation d'une transition entre un état initial et un état simple**

- **Nom** : EtatInitialToEtatSR
- **Priorité** : 10
- **Rôle** : cette règle permet de transformer une transition simple entre un état initial et un état simple du diagramme source (diagramme d'Etat Transition Mobile) vers un lient de poids égale à 1 dans le Nested Nets (**Figure 5.21)**.

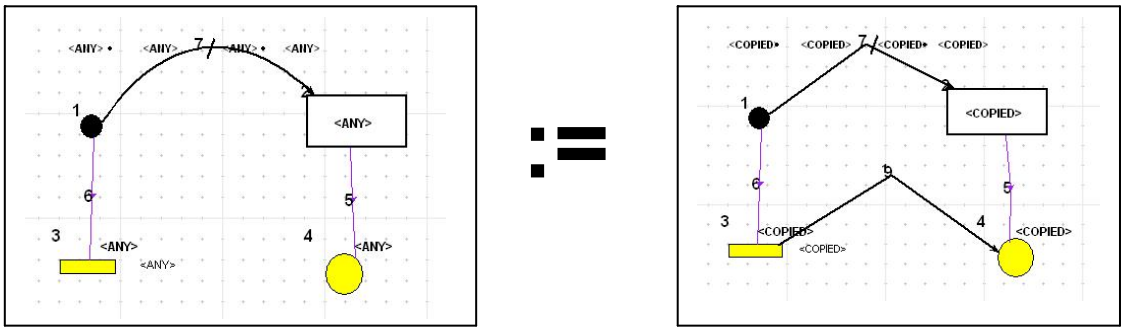

Figure 5.21: Transformation d'une transition entre un état initial et un état simple

Règle 16 : **Transformation d'une transition entre un état mobile et un état final**

- **Nom** : EtatMToEtatFinalR
- **Priorité** : 11
- **Rôle** : cette règle permet de transformer une transition simple entre un état mobile et un état final du diagramme source (diagramme d'Etat Transition Mobile) vers un lient de poids égale à 1 dans le Nested Nets (**Figure 5.22)**.

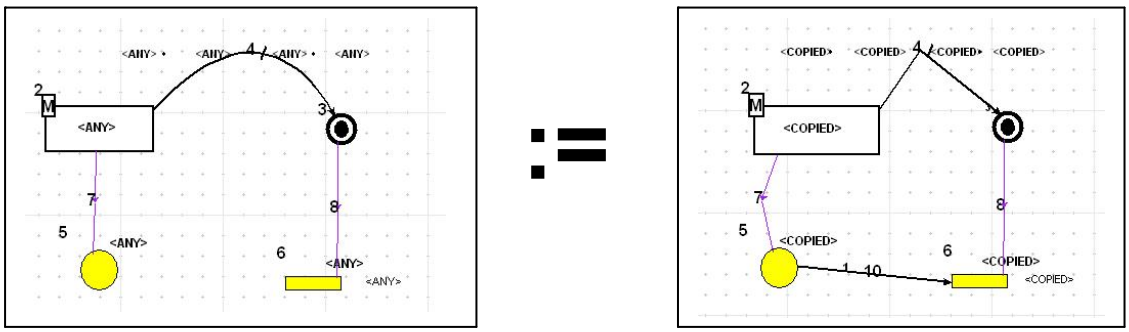

Figure 5.22: Transformation d'une transition entre un état mobile et un état final

Règle 17 : **Transformation d'une transition entre un état simple et un état final**

- **Nom** : EtatSToEtatFinalR
- **Priorité** : 12
- **Rôle** : cette règle permet de transformer une transition simple entre un état simple et un état final du diagramme source (diagramme d'Etat Transition Mobile) vers un lient de poids égale à 1 dans le Nested Nets (**Figure 5.23)**.

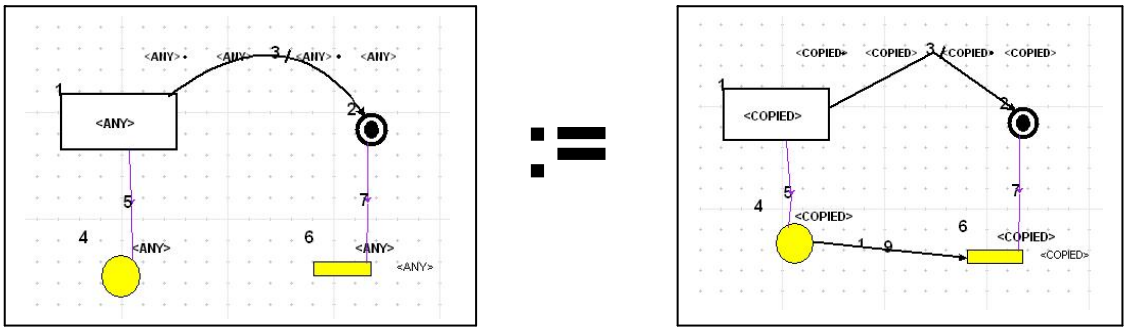

Figure 5.23: Transformation d'une transition entre un état simple et un état final

Règle 18 : **Transformation d'une transition d'un état simple vers lui même.**

- **Nom** : EtatSRelation
- **Priorité** : 12
- **Rôle** : cette règle permet de transformer une transition simple entre un état simple et lui-même du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau N1 et deux arcs de poids 1 dans le Nested Nets (**Figure 5.24)**. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation des attributs suivants: PFS, PFD, Evénements et Activités**.**

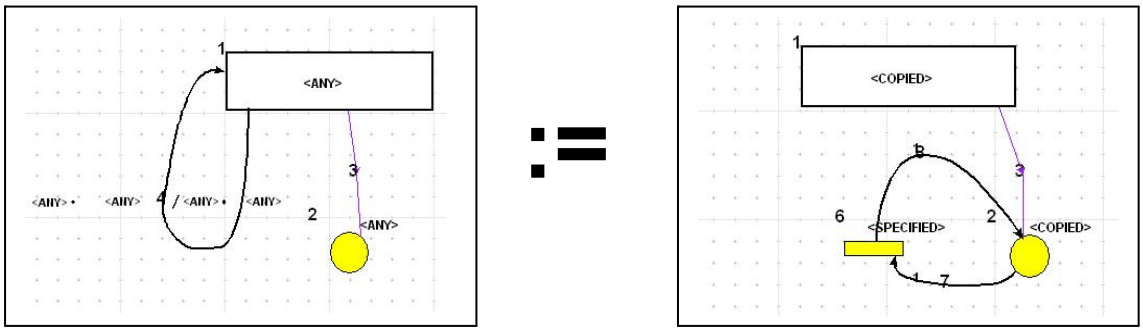

Figure 5.24: Transformation d'une transition d'un état simple vers lui même

Règle 19 : **Transformation d'une transition d'un état mobile vers lui même**

- **Nom** : EtatMRelation

- **Priorité** : 12

- **Rôle** : cette règle permet de transformer une transition simple d'un état mobile vers lui-même du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau N1 et deux arcs de poids 1 dans le Nested Nets (**Figure 5.25)**. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités**.**

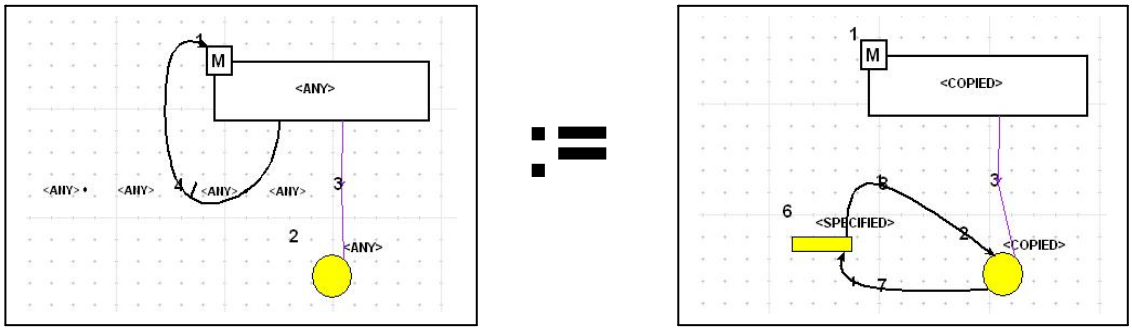

Figure 5.25: Transformation d'une transition d'un état mobile vers lui même

5.5.3 Les règles de transformation de l'ensemble des transitions aux transitions de niveaux 0: cette catégorie regroupe deuze règles qui visent à transformer les transitions du modèle source vers des transitions de niveau 0 dans le model destination.

<u>Règle 20</u> : **Transformation d'une transition mobile «agent return» entre un état mobile et un état simple**

- **Nom** : EtatMToEtatSRN0

- **Priorité** : 13

- **Rôle** : cette règle permet de transformer une transition mobile stéréotypée «agent return» entre un état mobile et un état simple du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 0 dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.26)**. La fonction de prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: Evénements et Activités précède par le mot **"On:".** La première place (nœud 4 dans LHS) pend comme nom l'attribut ***PFS*** de la transition du diagramme

source (nœud 3 dans LHS), et la deuxième place (nœud 5 dans LHS) pend comme nom le mot "**PFB**" pour designer la plateforme de base**.**

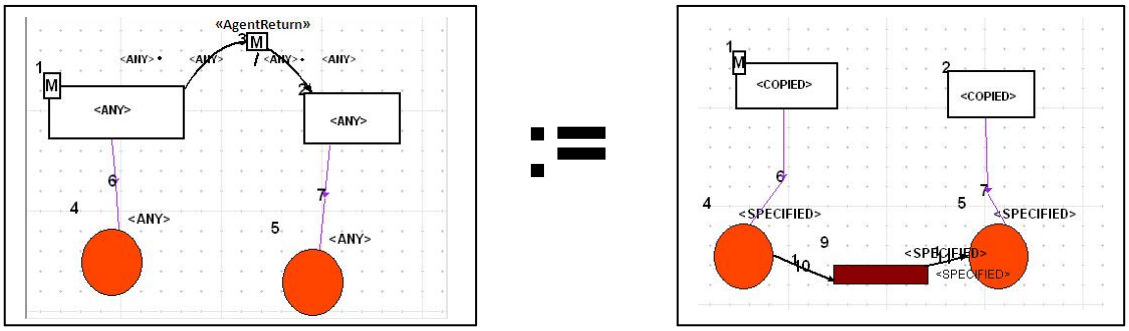

Figure 5.26: Transformation d'une transition entre un état mobile et un état simple

Règle 21 : **Transformation d'une transition mobile entre un état mobile et un état simple**

- **Nom** : EtatMToEtatSMN0
- **Priorité** : 13
- **Rôle** : cette règle permet de transformer une transition mobile entre un état mobile et un état simple du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 0 de type mobile dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.27)**. La fonction de synchronisation prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: Evénements et Activités précède par le mot **"On:".** La première place (nœud 5 dans LHS) pend comme nom l'attribut ***PFS*** de la transition du diagramme source (nœud 3 dans LHS), et la deuxième place (nœud 6 dans LHS) pend comme nom le mot "PFB" pour designer la plateforme de base**.**

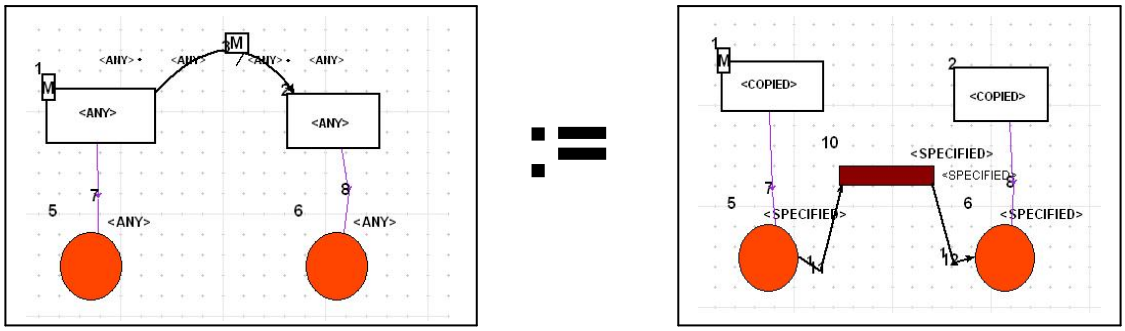

Figure 5.27: Transformation d'une transition mobile entre un état mobile et un état simple

Règle 22 : **Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états simples**

- **Nom** : EtatMToEtToEtRAR
- **Priorité** : 15
- **Rôle** : cette règle permet de remplacer la place de niveau N0 (nœud 7 dans LHS) destinataire d'une transition de niveau N0 (nœud 8 dans LHS) du Nested Nets par une autre place de niveau N0 (nœud 9 dans le LHS) en suppriment la place destinataire dans le LHS (nœud 7) avec la relation simple (nœud 4) dans le même LHS. La place destinataire dans le RHS (nœud 9) prend comme nom le nom de la place supprimé (nœud 7 dans LHS) (**Figure 5.28).** Ce remplacement est fait parce que chaque deux états connectés par une transition simple sont des états d'une même plateforme, sachant que l'une des deux places de niveaux N0 est toujours supprimée.

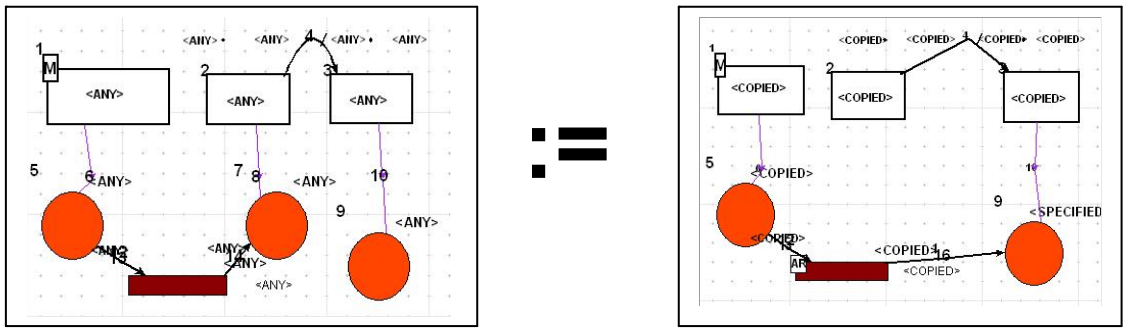

Figure 5.28: Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états simples

Règle 23 : **Transformation d'une transition mobile entre un état simple et un état mobile**

- **Nom** : EtatSToEtatMRMN0

- **Priorité** : 14

- **Rôle** : cette règle permet de transformer une transition mobile entre un état simple et un état mobile du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 0 de type mobile dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.29)**. La fonction de synchronisation prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: Evénements et Activités précède par le mot **"On:".** la première place (nœud 4 dans LHS) pend comme nom le mot "PFB" pour designer la plateforme de base, et la deuxième place (nœud 5 dans LHS) pend comme nom l'attribut ***PFD*** de la transition du diagramme source (nœud 3 dans LHS)**.**

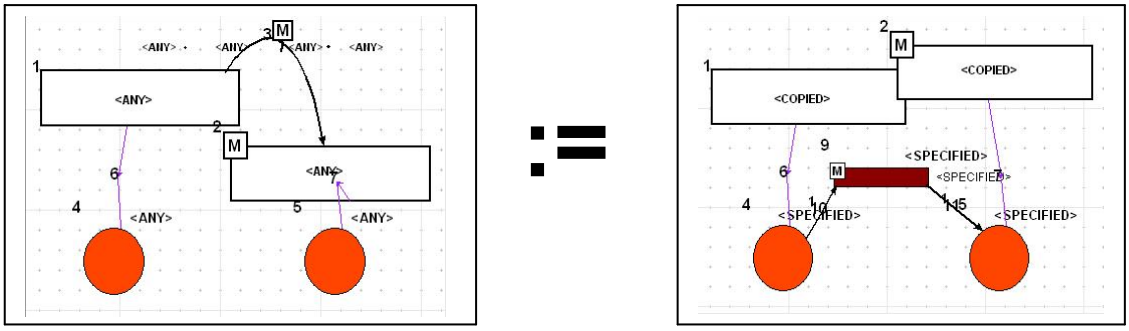

Figure 5.29: Transformation d'une transition mobile entre un état simple et un état mobile

<u>Règle 24</u> : **Transformation d'une transition mobile entre un deux états mobiles**

- **Nom** : EtatMToEtatMRMN0

- **Priorité** : 14

- **Rôle** : cette règle permet de transformer une transition mobile entre deux états mobiles du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 0 de type mobile dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS,

PFD, Evénements et Activités (**Figure 5.30)**. La fonction de synchronisation prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: Evénements et Activités précède par le mot **"On:".** La première place (nœud 5 dans LHS) pend comme nom l'attribut ***PFS*** de la transition du diagramme source (nœud 3 dans LHS), et la deuxième place (nœud 6 dans LHS) pend comme nom l'attribut ***PFD*** de la transition du diagramme source (nœud 3 dans LHS)**.**

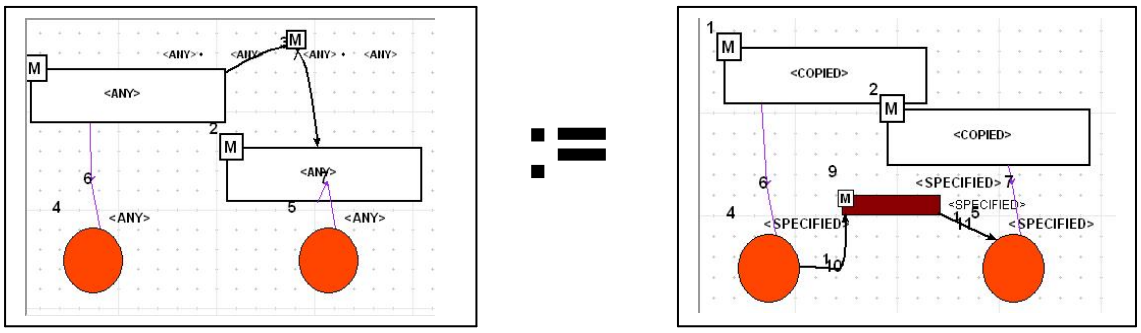

Figure 5.30: Transformation d'une transition mobile entre un deux états mobiles

<u>Règle 25</u> : Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états mobiles

- **Nom** : EtatMToEtToEtRAR
- **Priorité** : 15
- **Rôle** : cette règle permet de remplacer la place de niveau N0 (nœud 5 dans LHS) destinataire d'une transition de niveau N0 (nœud 8 dans LHS) du Nested Nets par une autre place de niveau N0 (nœud 6 dans le LHS) en suppriment la place destinataire dans le LHS (nœud 5) avec la relation simple (nœud 7) dans le même LHS. La place destinataire dans le RHS (nœud 6) prend comme nom le nom de la place supprimé (nœud 5 dans LHS) (**Figure 5.31).** Ce remplacement est fait parce que chaque deux états connectées par une transition simple sont des états d'une même plateforme, sachant que l'une des deux places de niveaux N0 est toujours supprimée.

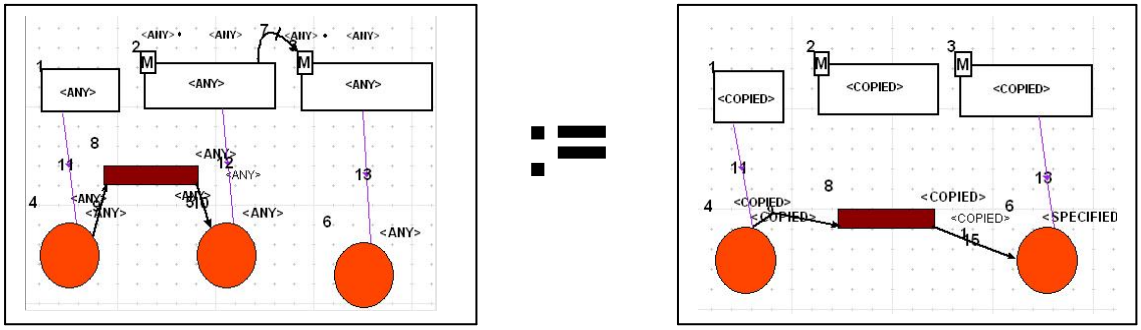

Figure 5.31: Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états mobiles

Règle 26 : **Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états mobiles**

- **Nom** : EtatMToEtatMRN0R
- **Priorité** : 15
- **Rôle** : cette règle permet de remplacer la place de niveau N0 (nœud 6 dans LHS) destinataire d'une transition de niveau N0 (nœud 11 dans LHS) du Nested Nets par une autre place de niveau N0 (nœud 7 dans le LHS) en suppriment la place destinataire dans le LHS (nœud 6) avec la relation simple (nœud 4) dans le même LHS, la place destinataire dans le RHS (nœud 7) prend comme nom le nom de la place supprimé (nœud 6 dans LHS) (**Figure 5.32).** Ce remplacement est fait parce que chaque deux états connectées par une transition simple sont des états d'une même plateforme, sachant que l'une des deux places de niveaux N0 est toujours supprimée.

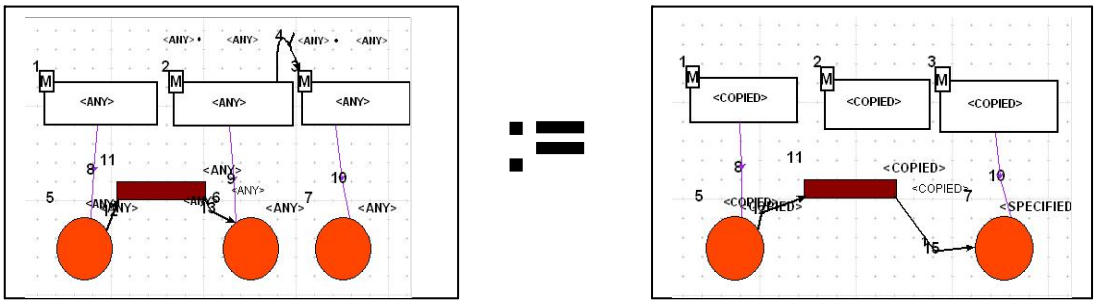

Figure 5.32: Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états mobiles

Règle 27 : **Transformation d'une transition à distance entre deux états simples**

- **Nom** : EtatSToEtatSRRN0
- **Priorité** : 16
- **Rôle** : cette règle permet de transformer une transition à distance entre deux états simples du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 0 de type à distance (R) dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.33)**. La fonction de synchronisation prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: Evénements et Activités précède par le mot **"On:".** La première et la deuxième place (nœud 4 et 5 dans LHS) pend comme nom le mot "PFB" pour dire que ces plateformes sont des plateformes de base**.**

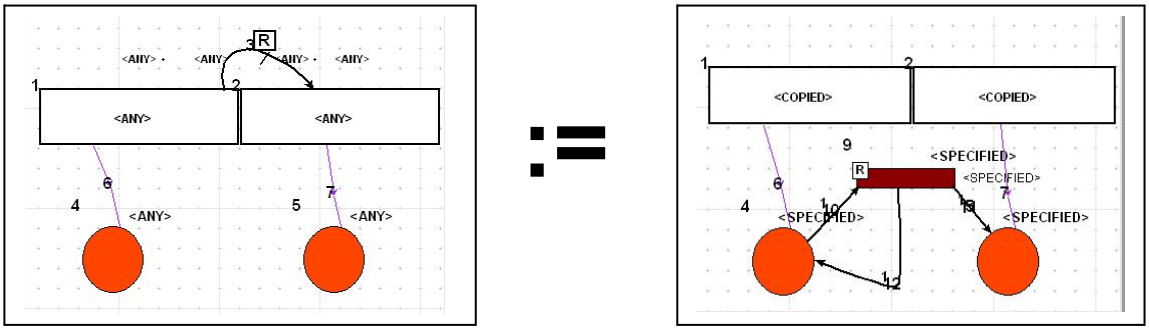

Figure 5.33: Transformation d'une transition à distance entre deux états simples

<u>Règle 28</u> : **Transformation d'une transition à distance entre deux états mobiles**

- **Nom** : EtatMToEtatMRRN0
- **Priorité** : 16
- **Rôle** : cette règle permet de transformer une transition à distance entre deux états mobiles du diagramme source (diagramme d'Etat Transition Mobile) vers une transition de niveau 0 de type à distance (R) dans le Nested Nets. Cette transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs suivants: PFS, PFD, Evénements et Activités (**Figure 5.34)**. La fonction de synchronisation prend comme nom, à partir de la transition du dia-

gramme source, la concaténation entre les attributs suivants: Evénements et Activités précède par le mot **"On:".** La première place (nœud 5 dans LHS) pend comme nom l'attribut ***PFS*** de la transition du diagramme source (nœud 3 dans LHS), et la deuxième place (nœud 6 dans LHS) pend comme nom l'attribut ***PFD*** de la transition du diagramme source (nœud 3 dans LHS)**.**

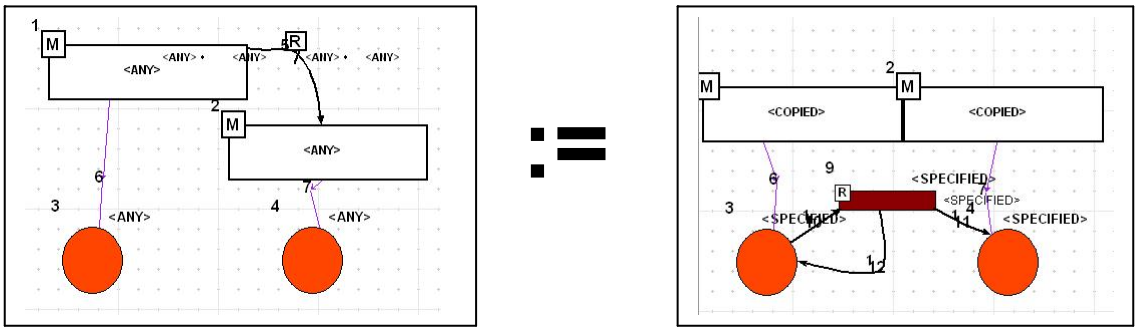

Figure 5.34: Transformation d'une transition à distance entre deux états mobiles

<u>Règle 29</u> : **élimination des places de niveau 0 inutiles**

- **Nom** : EtatSToEtatSRSN0
- **Priorité** : 17
- **Rôle** : Chaque deux états connectés par une transition simple sont de la même plateforme, par conséquent l'une des deux places de niveaux N0 est toujours supprimée (**Figure 5.35).**

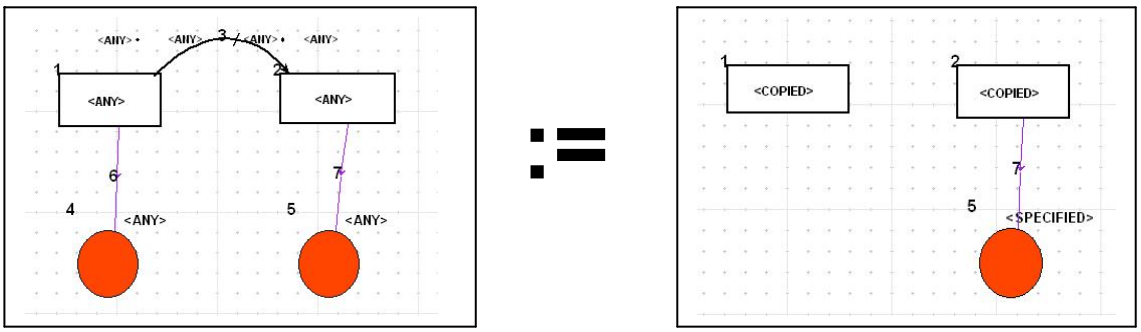

Figure 5.35: élimination des places de niveau 0 inutiles

<u>Règle 30</u> : **élimination des places de niveau 0 inutiles**

- **Nom** : EtatMToEtatMRSN0
- **Priorité** : 17

- **Rôle** : Chaque deux états connectés par une transition simple sont de la même plateforme, par conséquent l'une des deux places de niveaux N0 est toujours supprimée (**Figure 5.36).**

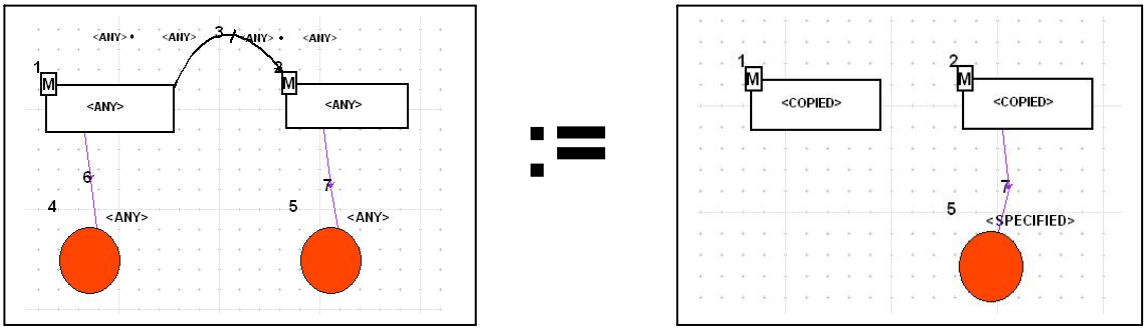

Figure 5.36: élimination des places de niveau 0 inutiles

Règle 31 : **Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états simples**

- **Nom** : EtatSToEtSToEtSRN0
- **Priorité** : 15
- **Rôle** : cette règle permet de remplacer la place de niveau N0 (nœud 6 dans LHS) destinataire d'une transition de niveau N0 (nœud 11 dans LHS) du Nested Nets par une autre place de niveau N0 (nœud 7 dans le LHS) en suppriment la place destinataire dans le LHS (nœud 6) avec la relation simple (nœud 4) dans le même LHS, la place destinataire dans le RHS (nœud 7) prend comme nom le nom de la place supprimé (nœud 6 dans LHS) (**Figure 5.37).** Ce remplacement est fait parce que chaque deux états connectées par une transition simple sont des états de la même plateforme, sachant que l'une des deux places de niveaux N0 est toujours supprimée.

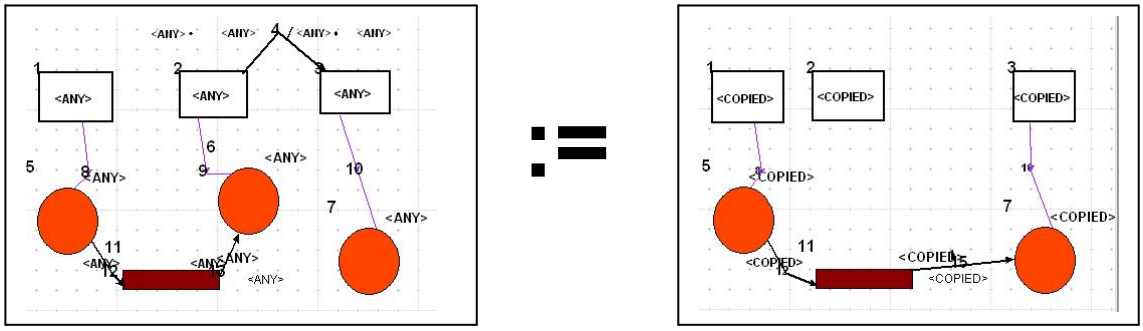

Figure 5.37: Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états simples

5.5.4 Les règles de changement de la structure des RDPs de deux niveaux du Nested Nets à cause des relations à distance: cette catégorie regroupe trois règles qui visent à changer la structure des RDPs de deux niveaux du Nested Nets à cause des relations à distance.

Règle 32 : **Changement de la structure des RDPs de deux niveaux du Nested Nets**

- **Nom** : TransitionMMM
- **Priorité** : 18
- **Rôle** : S'il existe une transition N0 de type 'R' qui précède une transition de type 'M', on ajoute une transition de retour de type 'R' après la première transition de type 'R' et par conséquent on ajoute une transition N1 et une place N1 dans le RDP du niveau N1. Ceci est justifié par le fait que l'agent qui envois un message à distance attend une repense (Transition de retour dans le RDP N0) **(Figure 5.38).** Le nom de la transition de retour N0 de type 'R' est la concaténation du mot 'Retour' et du nom de la transition N0 de type 'R' dans le LHS (nœud 7), la fonction de synchronisation de cette nouvelle transition de type N0 prend la valeur de la fonction transition N0 de type 'R' dans le LHS (nœud 7) concaténer avec le mot 'On:Retour'. On met le mot 'RetourAD' pour les noms de la place et la transition du RDP du niveau 1 les noms et on met le mot 'Retour' concaténé avec la fonction de la transition N0 de type 'R' dans le LHS (nœud 7).

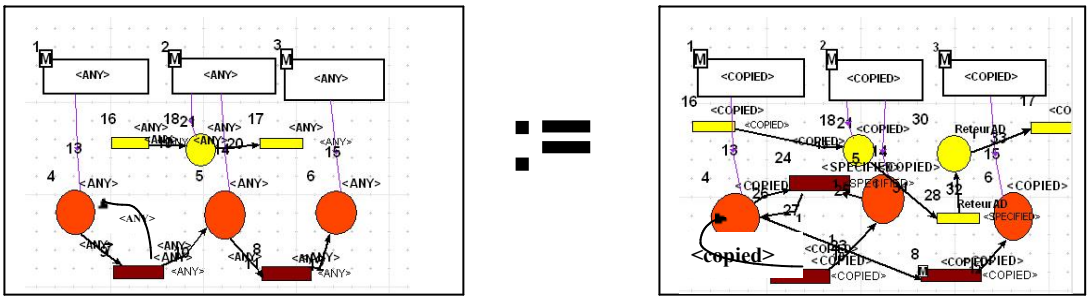

Figure 5.38: Changement de la structure des RDPs de deux niveaux du Nested Nets

Règle 33 : **Changement de la structure des RDPs de deux niveaux du Nested Nets**

- **Nom** : TransitionADRS
- **Priorité** : 18

- **Rôle** : S'il existe une transition N0 de type 'R' qui précède une transition de type 'M' on ajoute une transition de retour de type 'R' après la première transition de type 'R' et par conséquent on ajoute une transition N1 et une place N1 dans le RDP de niveau N1. Ceci est justifié par le fait que l'agent qui envois un message à distance attend une repense (Transition de retour dans le RDP N0) **(Figure 5.39).** Le nom de la transtion de retour N0 de type 'R' est la concaténation du mot 'Retour' et du nom de la transition N0 de type 'R' dans le LHS (nœud 7), et la fonction de synchronisation de cette nouvelle transition de type N0 prend pour valeur la fonction de la transition N0 de type 'R' dans le LHS (nœud 7) concaténer avec le mot 'On:Retour'. On met le mot 'RetourAD' pour les noms de la place et la transition du RDP de niveau 1 les noms et on met le mot 'Retour' concaténé avec la fonction de la transition N0 de type 'R' dans le LHS (nœud 7).

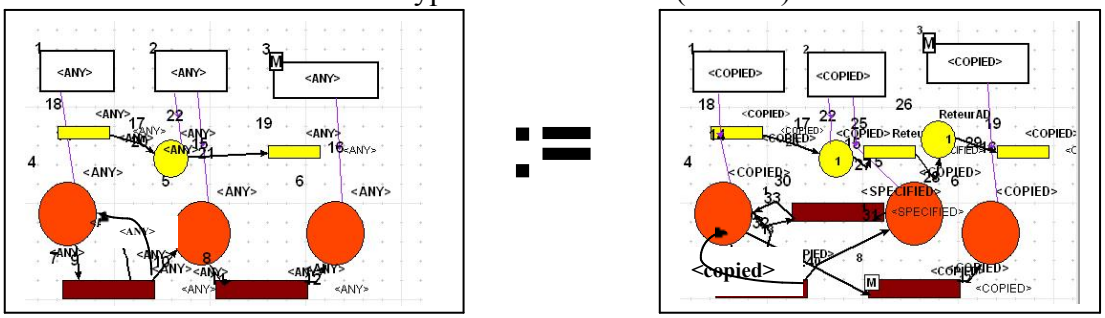

Figure 5.39: Changement de la structure des RDPs de deux niveaux du Nested Nets

<u>Règle 34</u> : **Changement de la structure des RDPs de deux niveaux du Nested Nets**

- **Nom** : TransitionADMS
- **Priorité** : 18
- **Rôle** : S'il existe une transition N0 de type 'R' qui précède une transition de type 'M' on ajoute une transition de retour de type 'R' après la première transition de type 'R' et par conséquent on ajoute une transition N1 et une place N1 dans le RDP de niveau N1. Ceci est justifié par le fait que l'agent qui envois un message à distance attend une repense (Transition de retour dans le RDP N0) **(Figure 5.40).** Le nom de la transition de retour N0 de type 'R' est la concaténation du mot 'Retour' et du nom de la transition N0 de type 'R' dans le LHS (nœud 7), la

fonction de synchronisation de cette nouvelle transition de type N0 est pris pour valeur la fonction de la transition N0 de type 'R' dans le LHS (nœud 7) concaténer avec le mot 'On:Retour'. On met le mot 'RetourAD' pour les noms de la place et la transition du RDP de niveau 1 les noms et on met le mot 'Retour' concaténé avec la fonction de la transition N0 de type 'R' dans le LHS (nœud 7).

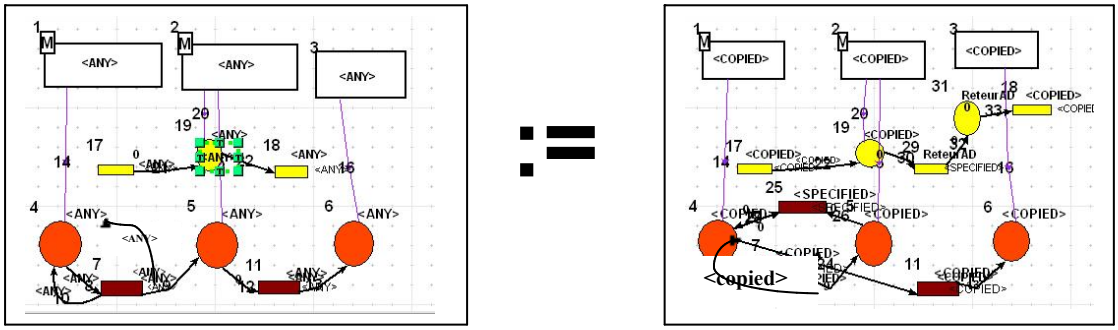

Figure 5.40: Changement de la structure des RDPs de deux niveaux du Nested Nets

5.5.5 Les règles qui éliminent les états du digramme d'état transition mobile: cette catégorie regroupe trois règles qui visent à éliminer les états du digramme d'état transition mobile.

Règle 35: **Elimination des états simples**

- **Nom** : EtatSDelete
- **Priorité** : 19
- **Rôle** : Cette règle permet de supprimer les états simples du diagramme d'état transition mobile (**Figure 5.41**).

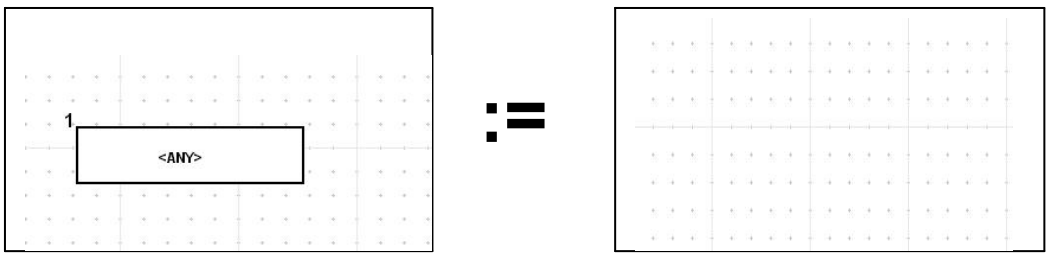

Figure 5.41: Elimination des états simples

Règle 36 : **Elimination des états mobiles**

- **Nom** : EtatMDelete
- **Priorité** : 19

- **Rôle** : Cette règle permet de supprimer les états mobiles du diagramme d'état transition mobile (**Figure 5.42**).

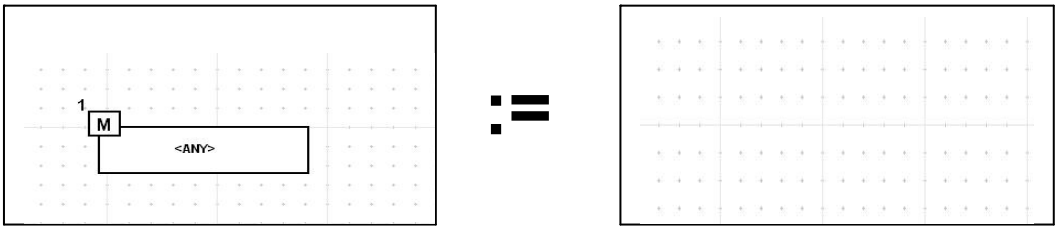

Figure 5.42: Elimination des états simples

Règle 37 : **Elimination des états initiaux**

- **Nom** : EtatIDelete
- **Priorité** : 19
- **Rôle** : Cette règle permet de supprimer les états initiaux du diagramme d'état transition mobile (**Figure 5.43**).

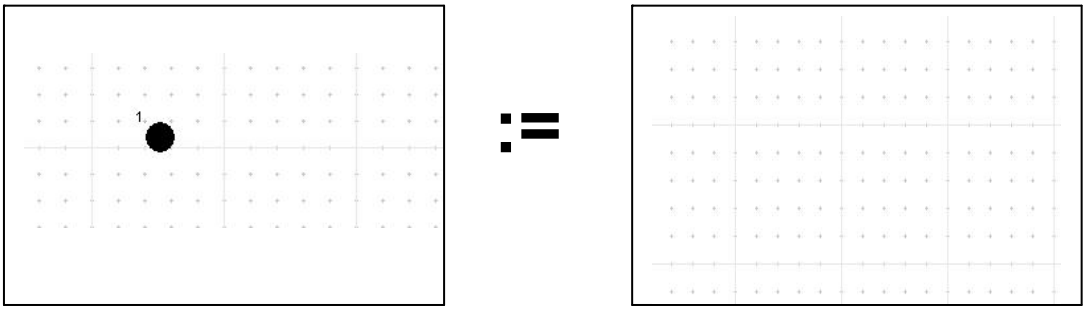

Figure 5.43: Elimination des états initiaux

Règle 38 : **Elimination des états finaux**

- **Nom** : EtatFDelete
- **Priorité** : 19
- **Rôle** : Cette règle permet de supprimer les états finaux du diagramme d'état transition mobile (**Figure 5.44**).

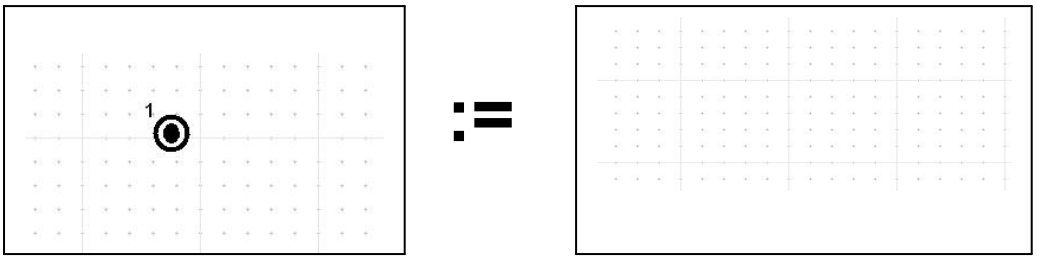

Figure 5.44: Elimination des états finaux

5.5.6 Les règles relient entre les deux niveaux d'un RdPs Nested Nets:

cette catégorie regroupe deux règles de liaison des RDPs de deux niveaux.

Règle 39 : **Un lien entre une place de niveau N0 et une place de niveau N1**

- **Nom** : ConnectionN0ToPN1
- **Priorité** : 20
- **Rôle** : Cette règle permet de faire un lien entre une place de niveau N0 et une place de niveau N1 si le nombre de jeton dans la place N0 égale à 1 (**Figure 5.45**).

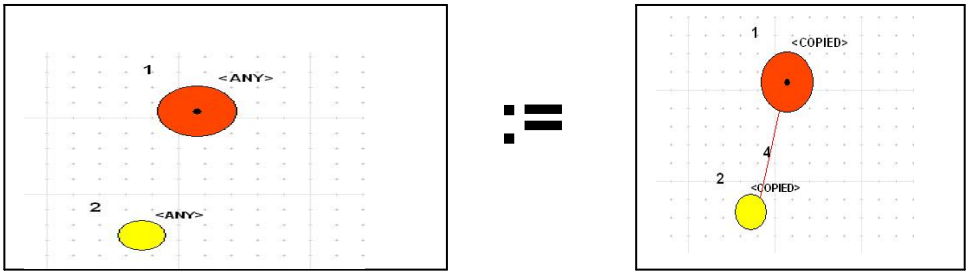

Figure 5.45: Un lien entre une place de niveau N0 et une place de niveau N1

Règle 40 : **Un lient entre une place de niveau N0 et une transition de niveau N1**

- **Nom** : ConnectionN0ToTN1
- **Priorité** : 20
- **Rôle** : Cette règle permet de faire un lien entre une place de niveau N0 et une transition de niveau N1 si le nombre de jeton dans la place N0 égale à 1 (**Figure 5.46**).

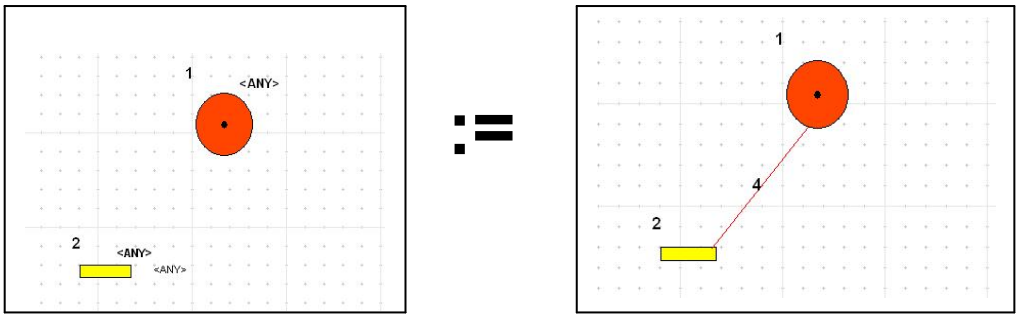

Figure 5.46: Un lien entre une place de niveau N0 et une transition de niveau N1

5.5.7 Exemple1 de transformation d'un diagramme d'état transition mobile vers un Nested Nets

Pour pouvoir concrétiser l'utilité de la grammaire définie, on a essayé de l'appliquer sur l'exemple présenté dans **[[Kas01](Fig2)].** La **figure 5.47** présente cet exemple sous forme d'un diagramme d'état transition mobile édité à l'aide de l'outil généré préalablement pour ce but qui est la modélisation des diagrammes d'état transition.

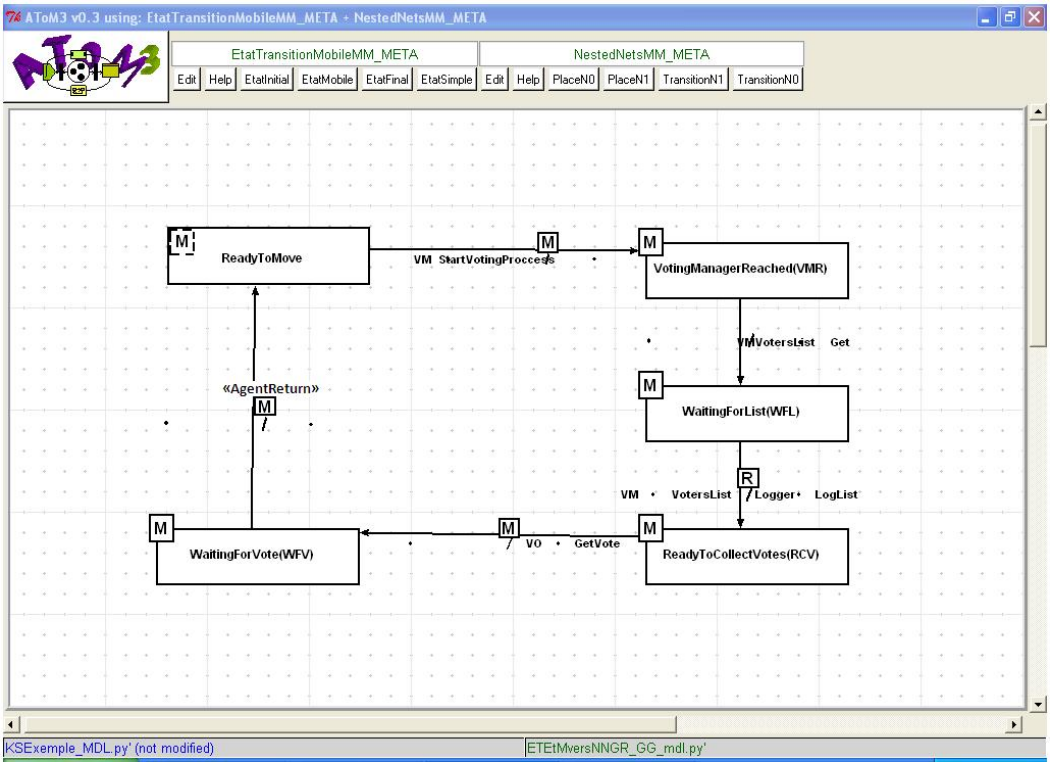

Figure 5.47 : Exemple d'un diagramme d'état transition mobile

Afin d'appliquer l'ensemble de règles sur l'exemple précédent, on doit exécuter la grammaire selon la figure suivante.

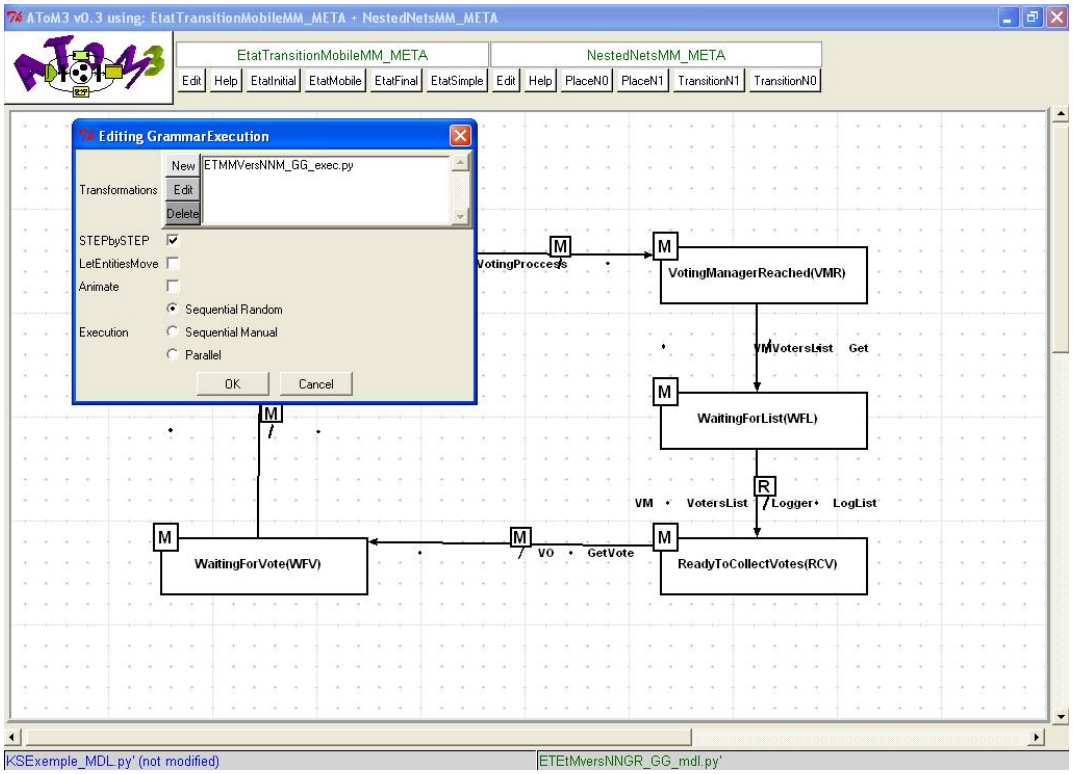

Figure 5.48: Lancement de l'exécution de la grammaire

L'exécution des règles est faite en respectant la priorité de chacune d'entre elles, de telle sorte que la règle qui possède la priorité numéro 1 soit la plus prioritaire, donc elle est exécutée en premier lieu, et la règle possédant la priorité numéro 20 est exécutée en dernier lieu.

La **figure 5.49** présente un graphe intermédiaire qui représente une réunion entre les éléments des deux formalismes source (*Diagramme d'état transition mobile*) et cible (Nested Nets). Il est obtenu en appliquant la règle numéro 8 (priorité 3) EtatMToEtatMR sur toutes les instances d'un message existant dans le graphe.

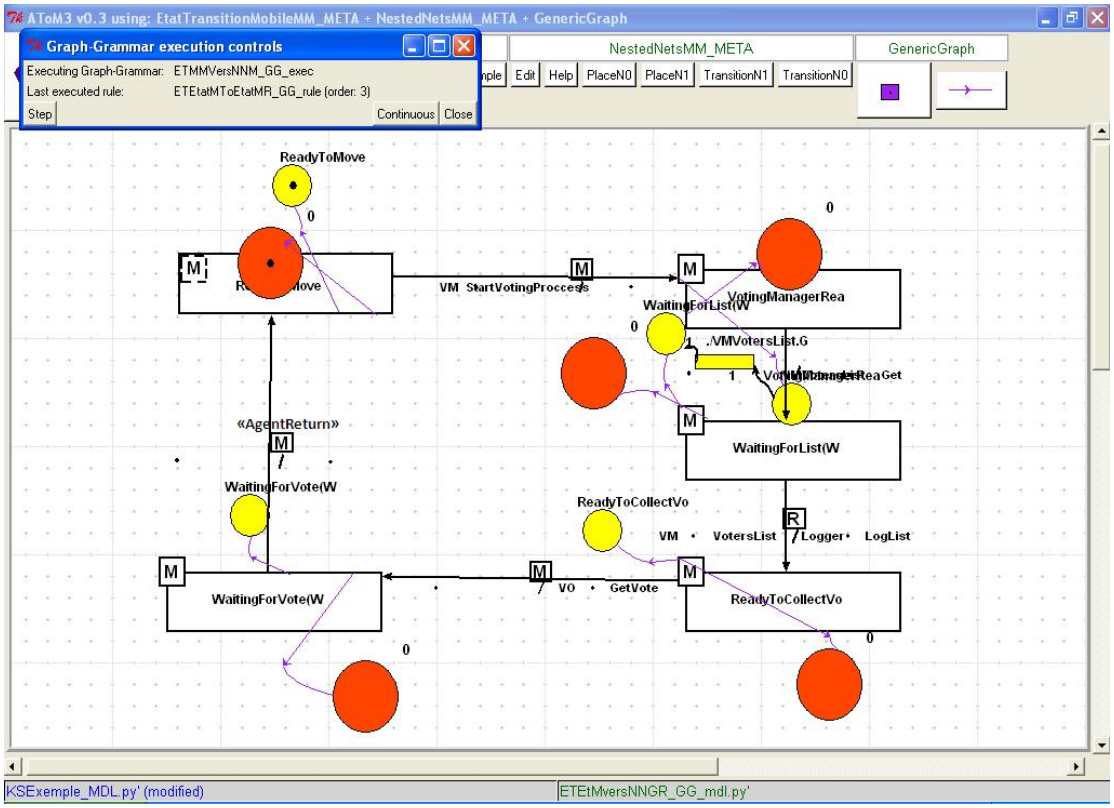

Figure 5.49 : graphe intermédiaire obtenu après l'exécution de la règle 8

L'exécution des différentes règles va permettre d'aboutir à un graphe final qui représente le modèle Nested Nets équivalent au diagramme d'état transition mobile de la **figure 5.50.**

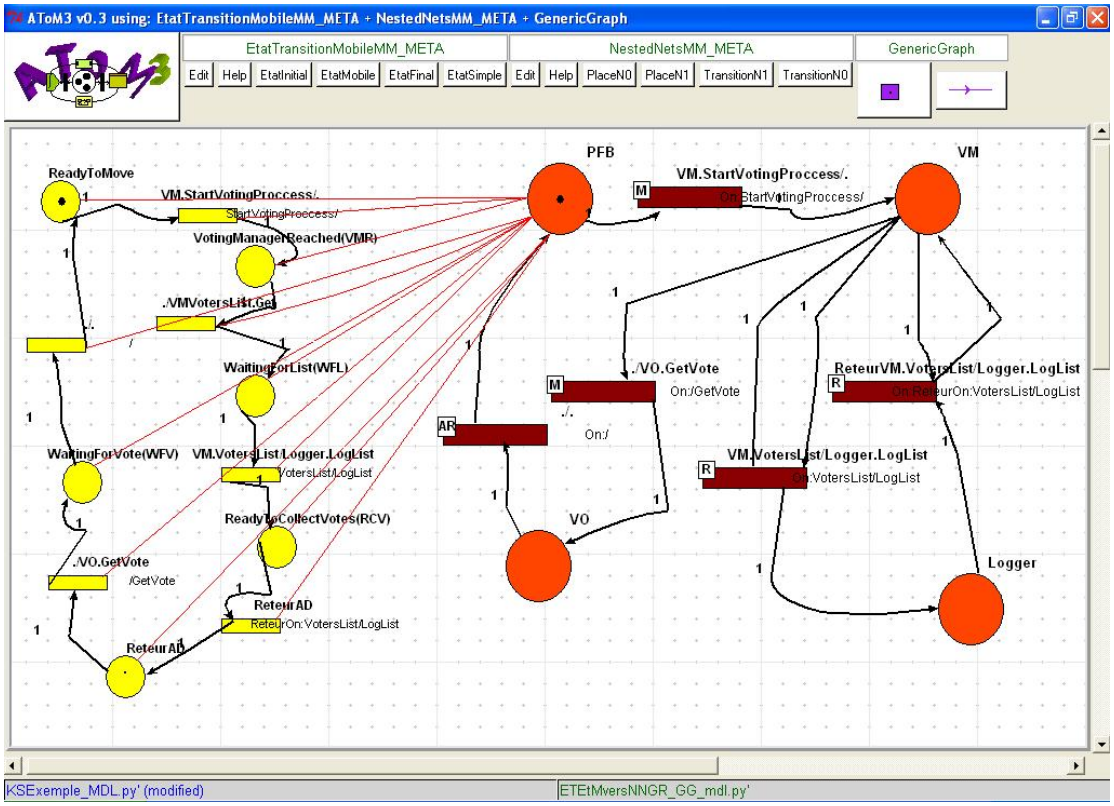

Figure 5.50 Nested Nets résultat de la transformation.

Pour vérifier le Nested Nets résultat de la transformation, on peut utiliser un outil de vérification des réseaux de Petri ordinaire. A titre d'exemple, on site l'outil **INA** pour vérifier chaque un des deux niveaux des réseaux de Petri séparément, pour voir plus sur la transformation d'un graphe de RdP vers la notation INA voir **[REA]**.

5.5.8 Exemple2 de transformation d'un diagramme d'état transition mobile vers un Nested Nets

On utilise l'exemple présenté dans **[[Kas02] (Fig12)].**

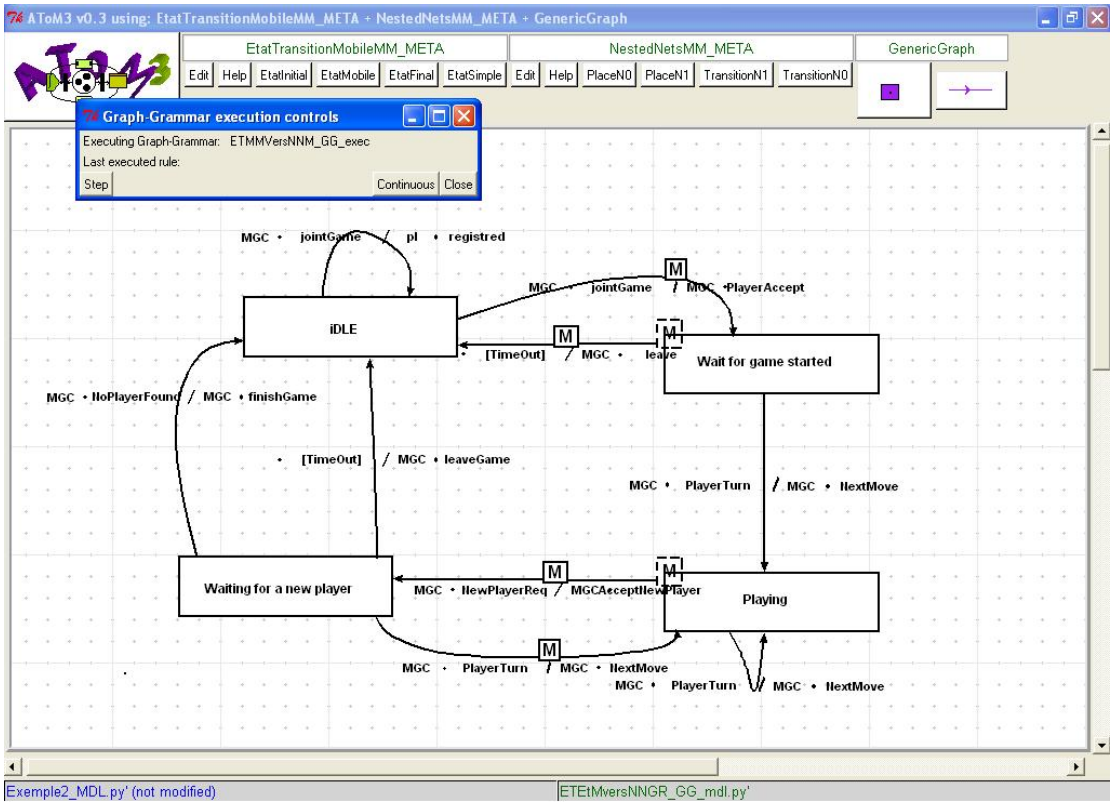

Figure 5.51 : Exemple d'un diagramme d'état transition mobile

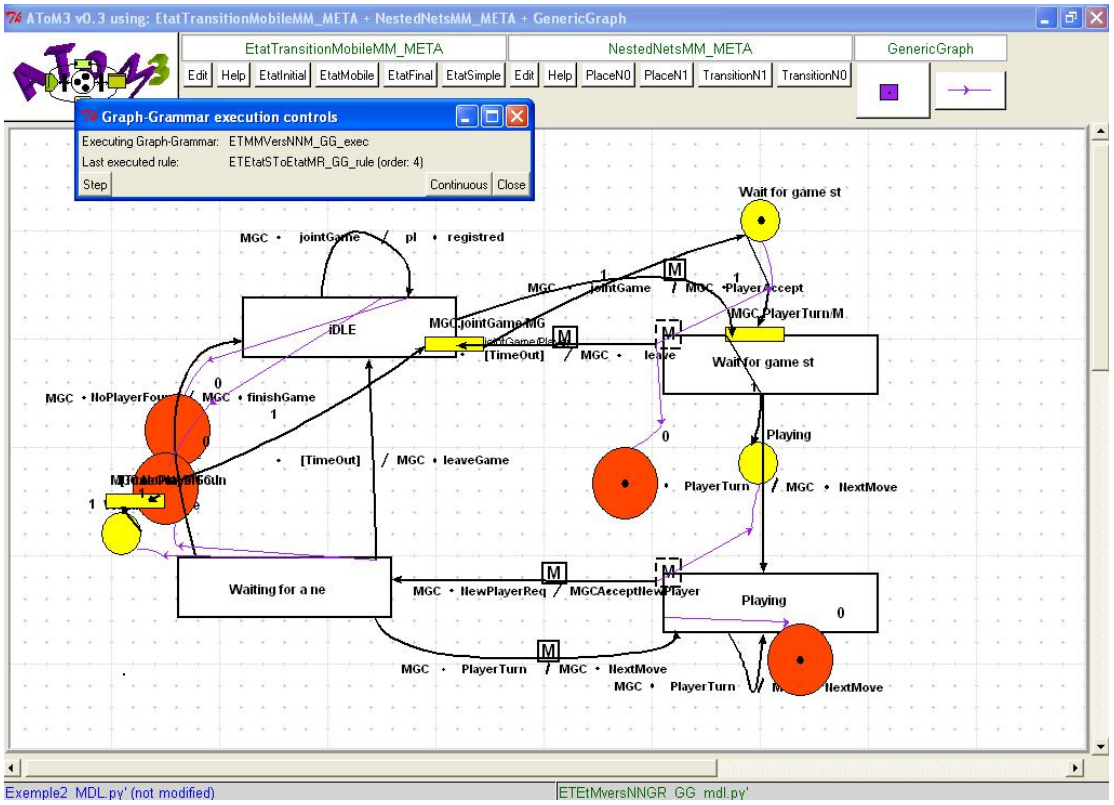

Figure 5.52 : graphe intermédiaire obtenu après l'exécution de la règle 9

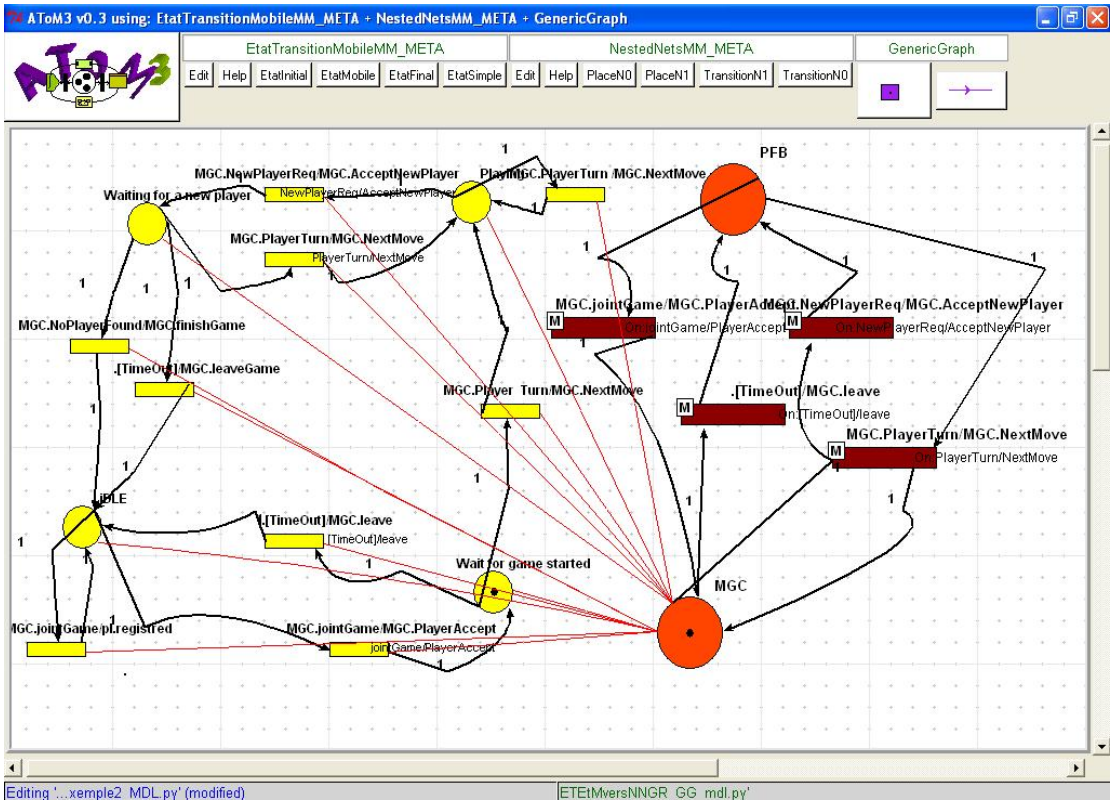

Figure 5.53 Nested Nets résultat de la transformation.

5.6 Conclusion

Dans ce chapitre en utilisant Atom3 **[Atom]** comme un outil de transformation de graphe on a essayé de proposé une grammaire de graphe pour transformer automatiquement n'importe quel diagramme d'état transition mobile vers les réseaux de Petri Nested Nets a deux niveaux d'abstraction pour des raisons de vérification. En commençant par une proposition d'un métamodèle pour le diagramme d'état transition mobile, suivie par une autre proposition d'un métamodèle pour le formalisme de RdP Nested Nets, ensuit nous avons proposé une grammaire de graphe pour transformer le formalisme source vers le formalisme cible, et finalement nous avons argumenté notre proposition avec deux exemples réels et avec des bons résultats.

Comme perspective et pour compléter notre travail pour couvrir l'aspect fonctionnel des applications mobile nous aurons proposé une grammaire de graphe pour la transformation automatique des diagrammes de séquence mobile vers les réseaux de Petri Nested Nets.

Conclusion générale

Conclusion général

Dans ce mémoire, on a essayé de présenter le concept de transformation de modèles qui est l'une des pièces centrales de l'approche MDA, par une proposition d'une grammaire de graphe qui permet de transformer les diagrammes d'UML mobile vers les RdP Nested Nets.

On a choisi l'approche de transformation de graphe 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 lisibles; parmi ces outils on a choisi l'outil **ATOM3** comme un bon outil de modélisation, multi formalismes, métamodélisation et de transformation de graphe. La transformation de graphe avec **ATOM3** est une opération simple et efficace grasse à leur sémantique.

Le formalisme source dans notre proposition est UML mobile (M-UML), cette dernière est une extension du langage UML destiné pour modéliser les applications d'agent mobile. On a choisi M-UML parmi toutes les approches et méthodes de modélisation des applications d'agent mobile car sa simplicité et sa couverture de tous les concepts de la mobilité telle que la location, le clonage et le chemin qui suit l'agent mobile durant son cycle de vue, ansai que tous les diagrammes et les vues d'UML standard.

Le formalisme destination est une extension des réseaux de Petri coloré qui s'appelle ***Nested Nets***, ce formalisme couvre tous les concepts de la mobilité grasse sa représentation par ***n*** niveaux d'abstractions. Dans ce travail on a utilisé deux niveaux d'abstraction et on peut utiliser plus de deux niveaux d'abstraction au fur et au mesure avec l'augmentation de la complexité des applications mobiles.

Nous avons argumenté notre proposition avec deux exemples réels et les résultats est très satisfaisantes.

Comme perspective et pour compléter notre travail pour couvrir l'aspect fonctionnel des applications mobile nous aurons proposé une grammaire de graphe pour la transformation automatique des diagrammes de séquence mobile vers les réseaux de Petri Nested Nets.

Bibliographie et Webgraphie

1. Bibliographie

- **[Adl]:** Adlèn Loukil, Héla Hachicha and Khaled Ghedira A proposed Approach to Model and to Implement Mobile Agents SOIE, Institut Supérieur de Gestion de Tunis, Université de Tunis.
- **[Anu97]** Anu Maria: **Introduction to Modelling and Simulation,** Proceedings of the 29th conference on Winter simulation, Atlanta, Georgia, United States, Pages: 7 – 13, 1997
- **[Arido]:** Arido Y. and Lange D.B.: **Agent Design Patterns: Elements of Agent Application Design**. *In* Proceeding ofAutonomous Agent '98, ACM Press.
- **[Aud07]** Laurent Audibert : **UML 2.0**, Institut Universitaire de Technologie de Villetaneuse, Département Informatique, Adresse du document : **http://www-lipn.univ-paris13.fr/audibert/pages/enseignement/cours.htm,** novembre 2007.
- **[BR05]** Michael Balaha, James Rumbaugh : **Modélisation et conception orientées objet avec UML 2**, deuxième édition, Pearson Education, France, 2005.
- **[BL-MO]:** Danny B.Lange and Mitsuru Oshima Programming And Deploying Java Mobile Agents with Aglets, 1998.
- [Briot-D]**: Jean-Pierre Briot et Yves Demazeau** Principes et architecture des systèmes multi-agents **11 octobre 2001 .Paris.**
- **[BRJ01]** Grady Booch, James Rambaugh et Ivar Jacobson : **Le Guide de l'utilisateur UML**, deuxième édition, Eyrolles, 2001.
- **[Car]**: Eric Cariou **Ingénierie des Modèles Transformation de modèles** Université de Pau et des Pays de l'Adour Département Informatique Eric.Cariou@univ-pau.fr.
- **[C.Cubat]**: Christophe CUBAT DIT CROS, thèse de doctorat titre « Agents Mobiles Coopérants pour les Environnements Dynamiques», Institut National Polytechnique de Toulouse 2005.
- **[Chr]**: Christian Kroiß Gefei Zhang_Ludwig-Maximilians **TOOL SUPPORTED MODELING OF MOBILE SYSTEMS** -Universit¨at M¨unchen, Germany.

- 🕮 **[Cld-K]**: Claude Kaiser **ANNEXE 2 LES RÉSEAUX DE PETRI** Reproduit avec la permission de **Francis Cottet**, ENSMA décembre 2001.
- 🕮 **[Cook]** Cook (S.) et Daniels (J.). **– Designing Object Systems - Object-Oriented Modelling with Syntropy.** – Prentice-Hall, 1994.
- 🕮 **[Cser]:** György Csertàn Gborà Huszerl Istvàn Màjzik Zsigmond Pap Andras Pataricza Dàniel Varró **VIATRA - Visual Automated Transformations for Formal Verification and Validation of UML Models.** Budapest University of Technology and Economics Department of Measurement and Information Systems.
- 🕮 **[Cza]**: Krzysztof Czarnecki and Simon Helsen **Classification of Model Transformation Approaches** Krzysztof Czarnecki and Simon Helsen University of Waterloo, Canada. czarnecki@acm.org, shelsen@computer.org.
- 🕮 **[DRIEU]:** Benjamin DRIEU **L'intelligence artificielle distribuée appliquée aux jeux d'équipe situés dans un milieu dynamique:** l'exemple de la RoboCup.
- 🕮 **[Edg01]:** Edgardo A. Belloni and Claudia A. Marcos: **Modeling of Mobile-Agent Applications with UML** ISISTAN Research Institute - Facultad de Ciencias Exactas - UNICEN Paraje Arroyo Seco - Tandil (B7001BBO) - Buenos Aires, Argentina.
- 🕮 **[Edg02]:** Edgardo Belloni and Claudia Marcos: **MAM-UML: An UML Profile for the Modeling of Mobile-Agent Applications** *ISISTAN Research Institute Facultad de Ciencias Exactas – UNICEN*
- 🕮 **[Favre]**: Jean-Marie Favre, Jacky Estublier et Mireille Blay-Fornarino **L'ingénierie dirigée par les modèles *au-delà du MDA*** Hermes Science publications Lavoisier.
- 🕮 **[Ferber 95]**: Jacques FERBER **- Les systèmes multi-agents Vers une intelligence collective** - Masson, 1995.
- 🕮 **[FrC]:** Franck Cassez et Olivier H. Roux**; Traduction structurelle des réseaux de Petri temporels en automates temporisés** IRCCyN/CNRS UMR 6597 BP 92101, 1 rue de la Noë F-44321 Nantes Cedex 3 Prénom.Nom@irccyn.ec-nantes.fr

📖 **[Gaia]** Wooldridge, M., Jennings N. R., and Kinny, D.**: The Gaia Methodology for Agent-Oriented Analysis and Design. 2000,** Journal of Autonomous Agents and Multi-Agent Systems, Vol.3, No. 3, pp. 285-312.

📖 **[G-Kar]**G. Karsai, A. Agrawal, **Graph Transformations in OMG's Model-Driven Architecture,** Lecture Notes in Computer Science, Vol 3062, 243-259, Springer Berlin / Heidelberg, juillet 2004.

📖 **[Guerra]:** Esther Guerra and Juan de Lara **A Framework for the Verification of UML Models. Examples using Petri Nets.** Escuela Politécnica Superior, Ingeniería Informática Universidad Autónoma de Madrid**.**

📖 **[Hach]**: Héla HACHICHA[1], Adlèn LOUKIL[2], Khaled GHEDIRA[3] **MAMT: an environment for modeling and implementing mobile agents** 1: ISIMS, Institut Supérieur d'Informatique et de Multimédia, Sfax hela.hachicha@fsegs.rnu.tn 2: INSAT, Institut National des Sciences Appliquées et de Technologie, Tunis adlen.loukil@insat.rnu.tn 3: ENSI, Ecole Nationale des Sciences de l'informatique, Tunis khaled.ghedira@isg.rnu.tn

📖 **[Hall]** Hall (A.). **– Seven myths of formal methods.** IEEE Software, September 1990.

📖 **[Har]:** Haralambos Mouratidis1, James Odell2, Gordon Manson1**: Extending the Unified Modeling Language to Model Mobile Agents** *1Department of Computer Science, University of Sheffield, England {h.mouratidis, g.manson}@dcs.shef.ac.uk 2James Odell Associates, Ann Arbor, MI USA email@jamesodell.com*

📖 **[Hin]** Bowen (J. P.) et Hinchey (M. G.). **– Seven more myths of formal methods**. IEEE Software, july 1995, pp. 34–41.

📖 **[José]:** Jos´e-Celso Freire Junior_ - Jean-Pierre Giraudin **– Agnès Front Atelier MODSI: Un Outil de M´eta-Mod´elisation et de Multi-Modélisation.** Laboratoire Logiciels Systèmes Réseaux – IMAG B.P. 72 - 38402 - Saint Martin d'H`eres Cedex – France.

📖 **[jade]:** Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood **Developing Multi-*Agent* Systems With *Jade*:** Amazon.fr:.

📖 **[Kang]:** Miao Kang*, Lan Wang and Kenji Taguchi: **Modelling Mobile Agent Applications in UML2.0 Activity Diagrams** *School of Computing,*

Leeds Metropolitan University Department of Computing, School of Informatics,*

📖 **[Kas01]:** Kassem Saleh*, Christo El-Morr **M-UML: an extension to UML for the modeling of mobile agent-based software systems** Department of Computer Science, American University of Sharjah, P.O. Box 26666, Sharjah, United Arab Emirates Received 30 December 2002; revised 29 June 2003; accepted 22 July 2003

📖 **[Kas02]:** Kassem Saleh; Christo El Morr; Aref Mourtada; Yahya Morad **A mobile-agent platform and a game application specifications using M-UML** *The Electronic Library;* 2004; 22, 1; Research Library pg. 32

📖 **[Kee]:** Kees van Hee1[1]; Irina A. Lomazova[2]*;* Olivia Oanea[1]*;* Alexander Serebrenik[1]; Natalia Sidorova[1]; Marc Voorhoeve[1] **Nested nets for adaptive systems** 1 Department of Mathematics and Computer Science Eindhoven University of Technology P.O. Box 513, 5600 MB Eindhoven, The Netherlands *f*k.m.v.hee, o.i.oanea, a.serebrenik, n.sidorova, m.voorhoeve*g*@tue.nl 2 Program Systems Institute of Russian Academy of Science, Pereslavl-Zalessky, 152020, Russia irina@lomazova.polnet.botik.ru

📖 **[Kim]:** Dr Mouhamed Tahar Kimour **Le processus unifiés (UP, RUP et TUP)** cours de l'école doctoral pôle est 2007.

📖 **[Klein**] Klein C., Rausch A., Sihling M. and Wen Z.**: Extension of the Unified Modeling Language for mobile agents.** In Siau K. and Halpin T. (Eds.): Unified Modeling Language. Systems Analysis, Design and Development Issues, chapter VIII. Idea Group Publishing, 2001.

📖 **[Koh 2007]:** Michael Köhler, Roman Langer, Rolf von Lüde, Daniel Moldt, Heiko Rölke and Rüdiger Valk **Socionic Multi-Agent Systems Based on Reflexive Petri Nets and Theories of Social Self-Organisation;** *Journal of Artificial Societies and Social Simulation* vol. 10, no. 1 http://jasss.soc.surrey.ac.uk/10/1/3.html 31-Jan-2007.

📖 **[Koh 02]**: Michael Köhler and Berndt Farwer **Object Nets for Mobility** Department for Informatics University of Hamburg, Application and Theory of Petri Nets.

📖 **[Kurt1 97]:** Jensen Kurt. **Coloured petri nets: Basic concepts, analysis methods and pratical use**, volume 1. Springer, 1997.

📖 **[Kurt2 97]**: Kurt Jensen **A Brief Introduction to Coloured Petri Nets** Computer Science Department, University of Aarhus Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark, E-mail: kjensen@daimi.aau.dk, WWW: http://www.daimi.aau.dk/~kjensen/

📖 **[Kus]:** Mario Kusek and Gordan Jezic**: Modeling Agent Mobility with UML Sequence Diagram** Department of Telecommunications Faculty of Electrical Engineering and Computing University of Zagreb Unska 3, Croatia, HR-10000 *University of Bradford*

📖 **[Laf]:** Pierre Laforcade, Vincent Barré, Boubekeur Zendagui **Scénarisation Pédagogique et Ingénierie Dirigé par les Modèles** LIUM / IUT de Laval 52 rue des Docteurs Calmette et Guérin 53020 Laval Cedex 9, France prénom.nom@lium.univ-lemans.fr.

📖 **[Lara 01]**: Juan de Lara[1] and Hans Vangheluwe[2] **Computer Aided Multi-Paradigm Modelling to Process Petri-Nets and Statecharts** [1]ETS Inform_atica Universidad Aut_onoma de Madrid Madrid Spain, Juan.Lara@ii.uam.es; [2]School of Computer Science McGill University, Montr2al Quèbec, Canada hv@cs.mcgill.ca.

📖 **[L-Bar]**L.Baresi, R.Hekel : **Tutorial Introduction to graph transformation. A software Engineering perspective**, Lecture Notes in Computer Science, Volume 3256/2004, 431-433, Springer Berlin, novembre 2004.

📖 **[Lokos]:** Charles Lakos: **A Petri Net View of Mobility (FORTE 2005)** © 2006, The University of Adelaide CNAM - Mars 2006.

📖 **[M-And]** M. Andries, G. Engels, A. Habel, B. Hoffmann, h.-J. Kreowski, s. Kuske, D. Pump, A. Schürr et G. Taentzer, **Graph transformation for specification and programming**, *Science of Computer programming*, vol 34, NO°1, pages 1-54, Avril 1999.

📖 **[Marc]**: Jean-Marc Jézéquel **L'ingénierie des modèles** professeur à l'université de Rennes 1 responsable du projet Triskell à l'INRIA/Irisa.

- **[MaSE]:** Self, A., DeLoach, S. A. 2003**. Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology.** Special Track on Agents, Interactions, Mobility, and Systems (AIMS) at The 18th ACM Symposium on Applied Computing (SAC 2003).
- **[MER 74]**: MERLIN P. M., **A study of the recoverability of computing systems**, PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1974.
- **[MG00]** Pierre-Alain Muller, Nathalie Gaertner : **Modélisation objet avec UML**, Deuxième édition, Eyrolles, 2000.
- **[m-Gaia]:** Sutandiyo, W., Chhetri, M. B., Loke, S.W., Krishnaswamy, S. 2004. **MGaia: Extending the Gaia Methodology to Model Mobile Agent Systems.** In the Sixth International Conference on Enterprise Information Systems (ICEIS 2004), Porto,Portugal, April 14-17.
- **[Mil]:** Pierre-Alain Muller **De la modélisation objet des logiciels à la meta-modélisation des langages informatiques,** Thèse de HDR présentée devant L'Université de Rennes.
- **[Morge] :** Maxime Morge **Interaction dans les systèmes multi-agents : Vers les systèmes multi-agents dialogiques.**
- **[Muscutariu]:** Muscutariu F. and Gervais M-P**.: On the modeling of mobile agent-based systems.** In 3rd International Workshop on Mobile Agents for Telecommunication Applications (MATA'01), LNCS Vol. 2164, pp. 219-234. Springer-Verlag, August 2001.
- **[Nav]**: Naveen Prakash[1], Sangeeta Srivastava[2] et Sangeeta Sabharwal[3] **The Classification Framework for Model Transformation** [1]JIIT, A-10, Sector 62, NOIDA 201307, India, [2]BCAS, Dwarka, Sector 2, New Delhi, India. [3]NSIT, Dwarka, Sector 3, New Delhi, India.
- **[Odel]** Odell (J. J). **– Specifying structural constraints.** *Journal of Object Oriented Programming*, 1993, pp. 12–16.
- **[OMG 03]:** Object Management Group (OMG): **MDA Guide Version 1.0.1**, copyright 2003.
- **[OMG03a]** Object Management Group: **OMG Unified Modeling Language Specification**, Version 1.5, mars 2003.

- **[OMG 04]**: Object Management Group (OMG), Model Driven Architecture (MDA), site Internet, http://www.omg.org/mda,2004.
- **[Pablo]:** Juan Pablo LópezGrao, José Merseguer, Javier Campos**. From UML Activity Diagrams To Stochastic Petri Nets: Application To Software Performance Engineering.** Departamento de Informàtica e Ingeniería de Sistemas Universidad de Zaragoza, Spain.
- **[PIC]:** G.P. Picco, A.L. Murphy, and G.-C. Roman: LIME**: Linda Meets Mobility.** In proc. Of 21th Int. Conf. On Software Engineering (ICSE), 1999.
- **[PERRET]:** Stéphane PERRET**, Agents mobiles pour l'accès nomade à l'information répartie dans les réseaux de grande envergure**, thèse doctoral, Université Joseph Fourier - Grenoble I, 1997.
- **[Picoo]:** G.P. Picco, A.L. Murphy, and G.-C. Roman: LIME: **Linda Meets Mobility. In** proc. Of 21th Int. Conf. On Software Engineering (ICSE), 1999.
- **[RAM 74]:** RAMCHANDANI C., **Analysis of asynchronous concurrent systems by timed Petri nets,** PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974, Project MAC Report MAC-TR-120.
- **[REA]:** Raida El Mansouri, Elhillali Kerkouche, and Allaoua Chaoui **A Graphical Environment for Petri Nets INA Tool Based on Meta-Modelling and Graph Grammars** WASET.ORG 2008.
- **[Rum]** Rumbaugh (J.), Blaha (M.), Premerlani (W.), Eddy (F.) et Lorensen (W.). **– Object Oriented Modeling and Design. –** Prentice Hall, 1991.
- **[SAD]:Wafa Saadi la transformation du diagramme de séquence vers les ECATNets.** Mémoire de Magistère.
- **[Sald]:** John Anil Saldhana and Sol M. Shatz **UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis.** Department of Electrical Engineering and Computer Science University of Illinois at Chicago.
- **[Scor]:** G. Scorletti et G. Binet **Réseaux de Petri** Maîtres de conférences à l'Université de Caen, France 20 juin 2006. Page web: http: //www.greyc.ensicaen.fr/EquipeAuto/Gerard S/mait_Petri.html.
- **[Simo]:** [1]Simona Bernardi, [1]Susanna Donatelli et [2]josé Merseguer**, From UML Sequence Diagrams and Statecharts to analysable Petri Net models**.

[1]Dipartimento di Informatica Università di Torino, Italy. [2]Dpto de Informática e Ingeniería de Sistemas University of Zaragoza, Spain.

📖 **[Sopena]:** Éric Sopena **Éléments de théorie des graphes,** Université de Bordeaux 1.

📖 **[Spy]** Spivey (J.M.). **– The Z Notation: A reference Manual. –** Prentice Hall, 1989.

📖 **[TRAN]:** TRAN Vinh Duc, Victor MORARU **Réseau de Petri Institut de la Francophonie pour l'Informatique -** Promotion 10 15 juillet 2005.

📖 **[Varro]: Dàniel Varro' and Andràs Pataricza Automated Formal Verification of Model Tranformations Budapest University of Technology and Economics Department of Measurement and Information Systems H-1521 Budapest, Magyar tudósok kórtúja 2.**

📖 **[Wool]:** Jeennings N. R. and Wooldridge M. (2000), **«Agent-Oriented Software Engineering» in** Handbook of Technology (ed. J.Bradshaw)AAAI/MIT Press**.**

📖 **[Yann]:** Yann Dantal et Christophe Haug **Théorie des graphes Principes et programmation** Soluscience 2003**.**

📖 **[Zhao]:** Yu Zhao[1], Yushun Fan[1], Xinxin Bai[1], Yuan Wang[1], Hong Cai[2], Wei Ding[2]**. Towards Formal Verification of UML Diagrams Based on Graph Transformation.** [1]CIM Research Center, Department of Automation, Tsinghua University, Beijing China 100084. [2]IBM China Research Lab, 4F, Haohai Building, 5th Shangdi Street, Beijing China 100085.

📖 **[Zhax]:** Zhaoxia Hu and Sol M. Shatz **Mapping UML Diagrams to a Petri Net Notation for System Simulation,** Concurrent Software Systems Laboratory University of Illinois at Chicago.

2. Webgraphie

📖 **[And]**: http://www.andromda.org

📖 **[Atom]:** AToM3 home page**: http://moncs.cs.mcgill.ca/MSDL/research/projects/ATOM3.html.**

📖 **[jak]**: http://jakarta.apache.org/velocity/.

🕮 **[jam]**: http://jamda.sourceforge.net.

🕮 **[Ker]**: www.kermeta.org.

🕮 **[mdsd]**: http://www.mdsd.info/.

🕮 **[OMG] www.omg.org :** Site web du groupe OMG**.**