République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mentouri Constantine
Faculté Des Sciences de l'Ingénieur
Département d'Informatique

N°: .....

Série: .....

Thèse de Doctorat en Sciences en Informatique

Thème :

Une approche intégrée Mobile-UML/Réseaux de Petri pour l'Analyse des systèmes distribués à base d'agents mobiles

Présentée par : M^r^ Mohamed Rédha BAHRI

Composition du Jury

Benmohammed Mohamed	**Professeur, Université Mentouri Constantine**	**Président**
Chaoui Allaoua	**Maître de Conférences, Université Mentouri Constantine**	**Rapporteur**
Seridi Hamid	**Professeur, Université de Guelma**	**Examinateur**
Kazar Okba	**Maître de Conférences, Université de Biskra**	**Examinateur**
Kholladi Mohamedkhireddine	**Maître de Conférences, Université de constantine**	**Examinateur**

Dédicaces

à mes parents

à ma famille

Remerciements

J’adresse toute ma reconnaissance au Dr Allaoua Chaoui Maître de conférences à l’université Mentouri Constantine qui a dirigé ce travail avec un très grand talent de professionnel. Je le remercie aussi pour sa patience et sa bienveillance durant toutes ces quatre années de thèse.

Je tiens à remercier le Pr Benmohammed Mohamed de m’avoir fait l’honneur d’accepter de présider mon jury de thèse.

Je remercie chaleureusement, Pr Seridi Hamid, Dr Kazar Okba et Dr MohamedKhireddine Kholladi, d’avoir accepté de juger ce travail.

Je tiens à remercier tous mes collègues et amis qui m’ont soutenus durant toutes ses années et en particulier Dr Bouznada Mourad et Dr Raida El Mansouri pour leurs aides et leurs soutiens.

Résumé

L'évolution rapide des réseaux sans fil et la diffusion des dispositifs portables, utilisant le réseau comme infrastructure, ont imposé de nouveaux aspects pour les applications réparties, telles que les applications à base d'agents mobiles.

Un agent mobile est une entité autonome, capable d'interagir avec son environnement ainsi que sur soi même, en se déplaçant d'un site du réseau à un autre pour atteindre ses objectifs.

Cependant, le développement des applications des agents mobiles se heurte à des problèmes inhérents à la sécurité et à l'interopérabilité dans les différentes plateformes d'accueil. Les insuffisances des efforts pour la standardisation de ces plateformes, ont poussé les chercheurs à explorer d'autres paradigmes pour une meilleure prise en charge et mise en œuvre des applications à base d'agents mobiles. D'autre part, peu de travaux de recherche portent sur les méthodes et les outils pour l'analyse et la conception des systèmes d'agents mobiles en s'appuyant sur le standard UML.

Malgré qu'UML possède plusieurs avantages, sa sémantique semi-formelle présente l'insuffisance de l'outil pour la vérification des modèles. Dans cette thèse, nous nous intéressons à l'enrichissement des diagrammes UML2.0 par stéréotypage pour capturer les aspects de la mobilité.

Afin de pallier aux insuffisances formelles d'UML, nous introduisons l'aspect de transformation de graphes issu des MDA (Model Driven Architecture) pour transformer les modèles stéréotypés vers les réseaux de Petri NestedNet. Le but est de pouvoir effectuer des vérifications formelles de ces modèles. Nous avons utilisé AToM^3 comme outil de transformation de graphes (graphe d'états transitions Mobil-UML vers graphe NestedNet).

Mots clés : Agent Mobile, UML, UML-Mobile, Réseaux de Petri, MDA, Transformation de Graphes.

Abstract

The network wireless fast evolution and the mobile device diffusion using the network as infrastructure imposed new aspects for distributed applications as mobile agent applications.
A mobile agent is an autonomous entity able to react on its environment and on it self in moving from one network site to another to reach its objectives.

However, the developments of the mobile agent applications face problems that are inherent to the security as well as to the interoperability in the reception of different platforms. The insufficiency standardization pushed researchers to explore other paradigms for a better holding and practice of applications with mobile agents basis . but few research works deal with methods and tools for the analysis and the conception of the mobile agent systems using UML standard basis .

Though, UML got many advantages its semi-formal semantics presents the disadvantage of the insufficiency of tools for model checking. In this thesis we are interested in the improvement of UML 2.0 diagrams by stereotyping to capture the mobility aspect to solve the insufficiency of UML formals we introduce the transformation graphic aspects coming from MDA model (model driven architecture) to transform the stereotype models toward Petri Nested Nets in order to be able to do the formal checking of this model .
We have used AToM^3 as a tool of graph transformation (transition state chart MOBILE-UML toward Nested Net graph).

Key Words: Mobile Agent, UML, Mobil-UML, Petri Nets, MDA, Graph Transformation.

ملخص

إن التقدم السريع للشبكة اللاسلكية و نشر الوسائل اللاسلكية مستعملين الشبكة كقواعد البناء الأساسية أدت إلى استعمال جوانب جديدة من أجل تطبيقات متفرقة مثل تطبيقات ذات وكيل محمول كأساس .
الوكيل المحمول هو وحدة مستقلة باستطاعتها التأثير على بيئتها ونفسها بانتقالها من شبكة للأخرى لتحقيق أهدافها

لكن تقدم تطبيقات الوكيل المحمول تصادفها مشاكل أمنية و توافقية في مختلف قواعد الإستقبال ,قلة المجهودات من أجل مواصفات القواعد أدى بالباحثين إلى استكشاف نماذج أخرى لأجل دعم تحقيق تطبيقات كأساس للوكيل المحمول و من وجهة أخرى هناك القليل من الأعمال التي اهتمت غير أن UMLبالوسائل و النماذج لتحليل و تصميم نظم الوكيل المحمول مستندين على توافق الوسيلة لمراجعة يملك مميزات كثيرة و دلالته نصف تجريدية تمثل مساوئ عدم قدرة UML النماذج

نمطيا للالتقاء جوانب الحركية UML2.0 في هده الأطروحة نهتم بإثراء المخططات البيانية لتغيير MDA ندخل جانب التعيير للرسوم البيانية من UMLمن أجل تقليص النقائص الرسمية ل والتي من هدفها انجاز تحقيقات رسمية لهده petri nested netالنماذج المنمطة حول الشابكات mobile- كوسيلة لتغيير الرسومات البيانية بيانات حالة الإستقلال Atom3النماذج ولقد استعملنا nested net إلى بيانات uml

الكلمات المفتاحية:تغيير , ,Petri nets محول شبكة , UML, Mobile-UML , وكيل محمول الرسومات البيانية

Table des matières

Liste des figures

CHAPITRE 1 Les agents mobiles

CHAPITRE 2 la modélisation UML

CHAPITRE3 Les réseaux de Pétri

Introduction générale

L'évolution rapide des réseaux sans fils et la diffusion de dispositifs portables utilisant le réseau comme infrastructure ont imposé de nouveaux aspects pour les applications réparties, telles que celles à base d'agents mobiles[Bahri09]. La mobilité en générale concerne le déplacement physique de dispositifs matériels ainsi que la migration des applications logicielles. De ce fait, beaucoup de concepts ont émergé dans le domaine de la mobilité, parmi lesquels nous citons: le *Code Mobile, le Calcul mobile* et *les Agents Mobiles.*

Un agent mobile est une entité autonome capable d'agir sur son environnement ainsi que sur soit même, en se déplaçant d'un site du réseau à un autre, pour atteindre ses objectifs. Bien que les agents mobiles soient introduits récemment, deux grandes normes de standardisation (FIPA [FIPA] et MASIF [MASIF]) et un nombre important de plates-formes [MASIF] ont apparu. Cependant, le développement des applications des agents mobiles se heurte à des problèmes inhérents à la sécurité et à l'interopérabilité dans les différentes plateformes d'accueil. Les insuffisances des efforts pour la standardisation de ces plateformes ont poussé les chercheurs à explorer d'autres paradigmes, pour une meilleure prise en charge et mise en œuvre des applications à base d'agents mobiles. D'autre part, très peu de travaux de recherches portent sur les méthodes et les outils pour l'analyse et la conception des systèmes d'agents mobiles. Les prédominantes approches orientée-agent qui existent jusqu'à maintenant sont celles qui cherchent à adapter les méthodologies d'analyse et de conception *orientées objets* [Bahri09], pour le développement des applications d'agents mobiles.

Il est évidement avantageux d'obtenir une approche qui pourra être utilisée tout au long du processus de développement des systèmes d'agents mobiles en inspirant ses notations de l'analyse et de la conception orientée objet (en particulier du langage UML). Cependant, un problème important s'impose. En effet, il s'agit de la relation des agents en général aux objets en particulier. Dans ce contexte plusieurs travaux [KSaleh04] ont été réalisés et qui ont donné naissance aux profiles UML tel que UML-Mobile.

Malgré le fait qu'UML possède plusieurs avantages, il est à la fois un support de communication, une norme, un langage de modélisation objet. Il offre aussi une bonne commodité pour la modélisation et la compréhension des modèles. La sémantique semi-formelle d'UML présente l'inconvénient de l'insuffisance de l'outil pour la vérification de tels modèles. Il serait idéal de pouvoir transformer ces modèles UML vers des modèles exprimés dans un langage formel (les réseaux de Petri par exemple) équivalent où la vérification est possible. Ceci permettra de palier aux insuffisances d'UML quant à la

vérification des modèles développés. Cette idée cadre avec la philosophie de l'approche IDM (Ingénierie des Modèles).

Ce que propose l'approche de l'ingénierie des modèles (IDM, ou MDE en anglais pour Model Driven Engineering) est de mécaniser le processus que les ingénieurs expérimentés suivent à la main [MDA05]. 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.[MDA05].
Dans le contexte de l'MDA, 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, chaque transformation prenant des modèles en entrée et produisant des modèles en sortie, jusqu'à obtention d'artéfacts exécutables.

Dans cette thèse, nous contribuons en premier lieu par la proposition d'une approche qui consiste en l'extension d'UML2.0 pour supporter la mobilité. Nous ajoutons de nouveaux éléments de modélisation dans les diagrammes d'UML2.0, pour faire face aux trois concepts principaux des agents mobiles à savoir: les *locations*, la *migration* et le *clonage*.

Notre deuxième contribution propose une démarche automatisée basée sur la transformation de modèle en utilisant les grammaires de graphe. Il s'agit en fait de transformer un diagramme UML(qui est un graphe) vers un réseau de Petri(qui est aussi un graphe) et ce, en puisant de l'aspect de la Méta-Modélisation.

Organisation du mémoire

Dans Le premier chapitre de cette thèse, nous abordons le modèle à agents mobiles. Nous nous intéressons particulièrement aux agents mobiles dédiés aux applications reparties, après l'introductions des agents en général. Nous focalisons par la suite sur le paradigme d'agent mobile et ses applications pour les systèmes répartis. Nous présentons enfin quelques plateformes pour l'implémentation des applications des agents mobiles.

Le deuxième chapitre introduit les éléments essentiels du langage de modélisation UML. Après un survol des travaux essentiels que nous avons jugés intéressants pour modéliser les agents mobiles avec UML, nous présentons notre contribution, qui consiste à étendre les différents diagrammes d'UML 2.0 pour la prise en charge de la mobilité. La dernière partie de ce chapitre illustre notre approche par une étude de cas d'un système de bourse électronique.

Afin d'atteindre nos objectifs, nous consacrons le troisième chapitre aux rappels de quelques concepts des réseaux de Petri, où nous mettons l'action sur les réseaux de Petri de haut niveau NestedNet.

Dans le quatrième chapitre, nous présentons le concept de transformation de modèles, en commençant par une présentation du concept de transformation de modèle dans le cadre général. Par la suite, vient l'énumération de quelques types de transformations. Une classification des différentes approches sera suivie par une présentation d'un cadre spécifique de transformation de modèles basée sur la transformation de graphes.

Le dernier chapitre décrit notre seconde contribution qui consiste en la proposition d'une approche intégrée UML-Mobile/NestedNets, pour la modélisation et la vérification des applications des agents mobiles. Nous exposons également dans ce chapitre l'outil **AToM^3** [ATOM3] qui nous a permit par le biais d'une grammaire de graphe, d'automatiser le transfert d'un formalisme source (méta-modèle pour le diagramme d'état transition mobile) vers un formalisme cible (méta-modèle pour le formalisme de RdPs NestedNets).

Enfin, une conclusion générale résume nos contributions à cette thèse et présente des perspectives de notre travail.

Chapitre1

Systèmes réparties et Agents Mobiles

La mobilité en générale concerne le déplacement physique de dispositifs matériels ainsi que la migration des applications logicielles. Ainsi, beaucoup de concepts ont émergé dans le domaine de la mobilité parmi lesquels nous citons: le *Code Mobile, le Calcul mobile et les Agents Mobiles.*

Dans ce chapitre, nous nous intéressons aux agents mobiles dédiés aux applications réparties. Après l'introduction des agents en général et la motivation des agents mobiles en particulier, nous nous focaliserons sur le paradigme d'agent mobile, et nous présenterons enfin quelques plateformes pour l'implémentation des applications d'agents mobiles.

1.1. Systèmes répartis et Agents mobiles

La programmation par agents mobiles a émergé comme un nouveau paradigme pour le développement des applications distribuées. Cependant, il existe d'autres structures organisationnelles pour la construction et la gestion des systèmes répartis, que l'on peut classifier en deux catégories. La première concerne les structures d'organisation sans mobilité, par contre, la deuxième comporte les structures d'organisation avec mobilité.

1.1.1. Structures organisationnelles non mobiles pour les systèmes répartis

La conception d'une application répartie classique, ne nécessitant pas la mobilité, peut faire intervenir plusieurs structures organisationnelles dont les plus répandues seront classifiées comme suit :

1.1.1.1. Client/serveur

Dans le modèle client/serveur, le système distribué est développé à partir d'un ensemble de clients interagissant avec un ensemble de serveurs. Les clients réalisent des appels de services offerts par les serveurs (appels de méthodes synchrones sur les objets des serveurs). Les fonctionnalités d'une application client/serveur sont alors

encapsulées dans les services proposés et exécutés au sein de serveurs (fournisseurs de services).
Le client/serveur est vu comme une technique de dialogue entre deux processus. Le processus client demandeur de service, établit une interaction avec un serveur et orchestre les interactions entre les différents services demandés. Alors que, le processus serveur a pour rôle de répondre aux demandes des processus client, ce qui le rend indépendant des applications qui l'interrogent.(cf. la figure 1.1)

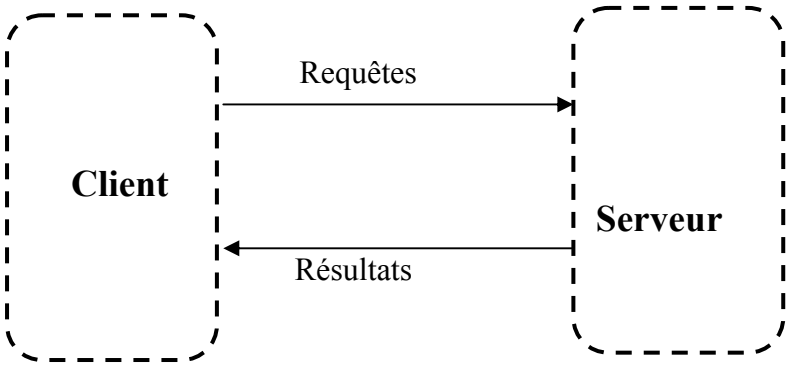


Figure 1.1. Organisation Client/Serveur

Cette structure de base se caractérise par l'exécution synchrone des requêtes. Le processus client émet une requête au processus serveur et se bloque en attente de la réponse. La réponse à la requête du client nécessite l'existence sur un même site des trois éléments suivants [Fpv98]:

- le code du service (les opérations à effectuer) ;
- une unité d'exécution (une machine qui encapsule l'état de l'exécution) ;
- un ensemble de ressources qui représente les données passives ou les unités physiques (processeur, etc …).

L'interaction entre le client et le serveur peut suivre plusieurs mécanismes (Protocoles), ce qui permet de différencier plusieurs types de requêtes permettant au client de communiquer avec un serveur, tels que : l'envoi de message, l'appel de procédure à distance et l'appel de méthode à distance.

Néanmoins, on constate que dans le modèle client/serveur, le client invoque des opérations qui seront exécutées sur le serveur. Ce qui centralise l'exécution des services au sein des serveurs, limitant ainsi la dynamique des systèmes distribués.

1.1.1.2. Mémoire distribuée partagée

Le principe de cette structure organisationnelle se base sur la notion d'espace d'échange partagé par un ensemble de processus distribués [BZS93]. L'utilité d'une telle mémoire est d'offrir un grand espace qui facilite les échanges de données cohérentes entre différentes applications.

1.1.1.3. Abonnement/Evénement

Cette structure organisationnelle asynchrone facilite la communication entre différents processus. Lorsqu' un événement précis arrive dans l'environnement partagé, les processus concernés prendront part à l'information. Dans ce contexte, un événement se définit comme une occurrence asynchrone dans un système distribué [Rou02].

1.1.2. Structures organisationnelles avec mobilité pour les systèmes répartis

Avec la croissance rapide des applications utilisant les réseaux comme infrastructures, notamment Internet, les structures organisationnelles non mobiles, entre autre l'approche client/serveur, se heurtent à de très grands défis. Le caractère centralisé de ces approches ralentit énormément la prolifération des nouvelles tendances pour le partage du savoir et des ressources disponibles sur le réseau planétaire. De ce fait, le concept de la mobilité émerge comme une alternative des approches non mobile.
Dans cette section, nous introduisons quelques définitions et concepts de bases de la mobilité, ensuite nous présentons quelques structures d'organisation avec mobilité pour les systèmes répartis.

1.1.2.1. La mobilité

La mobilité en général concerne le déplacement physique ou logiciel dans les réseaux de télécommunication. Dans ce contexte, trois concepts de base sont répandus à savoir : la mobilité physique (*Mobile computing*), le code mobile et les agents mobiles.

- La mobilité physique se définie [Gg96] comme étant un paradigme dans lequel les utilisateurs du réseau partagé se connectent via des machines déplaçables. La motivation de ce paradigme se justifie par : la diffusion des réseaux sans fils, et les réseaux de télécommunication cellulaires. Ces réseaux réclament sans arrêt de grandes capacités de calcul et de traitement de l'information.

- Le code mobile se définit par la capacité dynamique d'échange entre les fragments du code à exécuter et les locations sur lesquels s'exécutera ce code. Faire migrer un code (processus, objet, ou une procédure) permet d'assurer un équilibrage de charge entre les processeurs connectés sur le réseau. Il permet également d'améliorer les performances dans la communication(rassembler les objets qui se communiquent intensivement sur les mêmes nœuds).

• les agents mobiles sont des entités logicielles pouvant se déplacer de façon autonome d'un site du réseau à un autre pour atteindre leurs objectifs.

1.1.2.2. Structures organisationnelles

Parmi les structures organisationnelles, nous allons examiner le code à la demande l'évaluation à distance et le modèle à Agent Mobile.

a-Code à la demande

Dans cette structure, le client dispose de l'unité d'exécution et des ressources mais pas du code nécessaire à la réalisation d'un service précis. Ce code sera récupéré auprès d'un serveur. Ainsi, un client adresse une requête uniquement pour récupérer un code précis afin de l'exécuter localement avec les ressources disponibles. Les trois éléments sont réunis sur le site client comme le montre la figure 1.2 [Rou02]. Cette méthode permet d'étendre les fonctionnalités d'une application directement chez le client sans avoir besoin d'effectuer une nouvelle installation. L'exemple le plus répandu d'utilisation de cette méthode est le téléchargement d'applet Java à partir d'un serveur Web. Ces applets sont des classes Java présentes sur le site serveur, elle seront téléchargées puis interprétées par la machine virtuelle du site client.

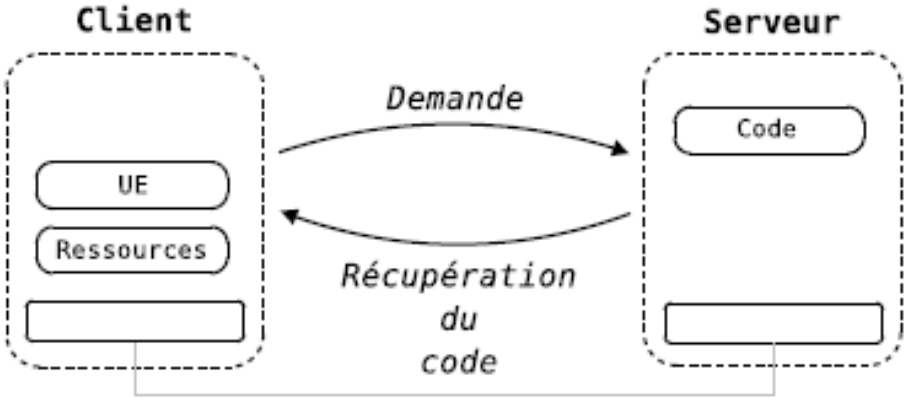


Figure 1.2. Code à la demande [Rou02]

b- Evaluation à distance

L'interaction par évaluation à distance, se réalise par l'envoi du code d'un site client à un autre site serveur. Les ressources et l'unité d'exécution étant initialement sur le serveur. La mise en route du service est réalisée lorsque le serveur reçoit le code à exécuter. Une fois le service réalisé, les résultats seront retournés au client. Le code et

l'unité d'exécution seront par la suite supprimés. Ce mécanisme est illustré dans la figure 1.3 [Rou02] ci-dessous.

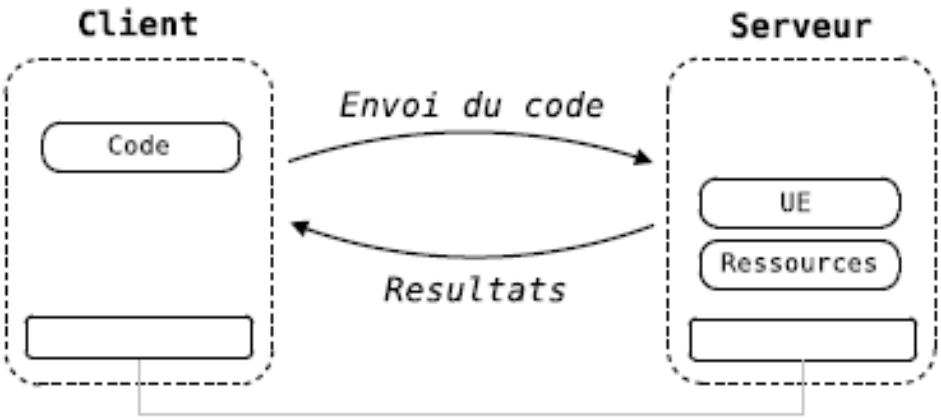


Figure 1.3. Evaluation à distance [Rou02]

Dans cette approche, le client envoie directement le code au serveur. Par exemple, le ***rsh*** (***Remote Shell***) du système **Unix** permet d'exécuter un ***script*** sur une machine distante, où un représentant locale du service est utilisé. Par exemple, l'impression d'un document sur une imprimante *Postscript,* où un client s'adresse à un serveur local en lui communiquant un fichier, qui est ensuite envoyé à l'imprimante où il sera interprète dans une unité d'exécution propre. Nous pouvons aussi mentionner que cette technique peut se réaliser par les requêtes SQL adressées à un serveur de base de données.

On constate que les approches citées ne présentent aucun risques quant à la gestion de la sécurité sur les différents points du réseau.

c- Modèle à Agents Mobiles

Les contraintes imposées par les approches, tels que le code à la demande et l'évaluation à distance, ne répondent pas parfaitement aux améliorations attendues de l'implication de la mobilité pour les systèmes distribués. Les agents mobiles ont été introduits en 1994 par White [Whi94] via la description de l'environnement Téléscript comme une autre alternative de la mobilité. Dans cet environnement, des processus (code et unité d'exécution) peuvent se déplacer d'eux-mêmes d'un site du réseau à un autre (cf. la figure 1.4) pour interagir localement avec des ressources d'autres sites. Cette technologie est alors apparue comme prometteuse pour la conception d'applications distribuées

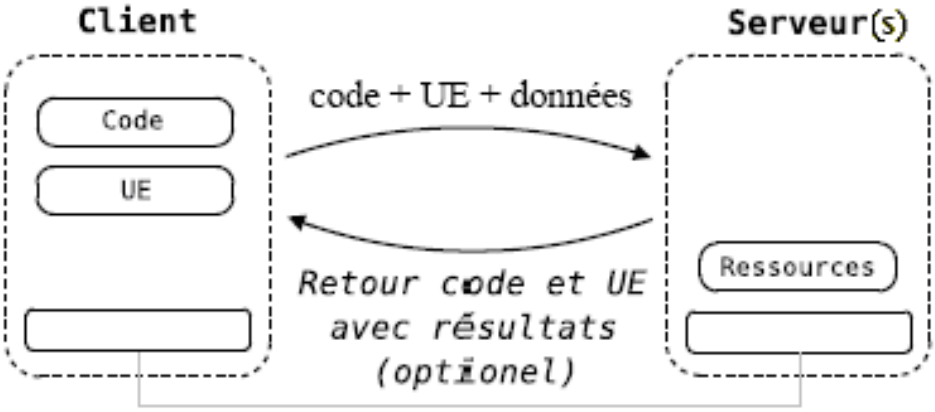


Figure 1.4. Interaction par agents mobiles [Cubat05]

1.2. Motivation des agents mobiles

Un agent mobile peut transporter dans le réseau ou ils se déplace, des données du code ainsi que les unités d'exécution qui lui sont associées. Un agent migre de manière autonome vers les sites qu'il juge intéressants. Cette nouvelle approche a permis d'augmenter la robustesse des applications pour les systèmes répartis, notamment pour :

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

• ***Le Déplacement du 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.

• ***La Fiabilité:*** La vie d'un programme classique est liée à la machine où 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. Les agents Mobiles

1.3.1.Concept d'agents

1.3.1.1. Définitions

Un agent est une entité qui perçoit et agit dans un environnement. Sa fonction agent, spécifie l'action qu'il exécute en réponse à une séquence de percepts donnée [Rous06]. En général, un agent est substitué à toute entité qui peut être considérée comme percevant son environnement grâce à des capteurs et qui agit sur cet environnement via des effecteurs. Nous présentons ici quelques définitions alternatives selon l'approche de l'intelligence artificielle:

• D'après[Ferber95], un agent est une entité physique ou virtuelle qui est capable d'agir dans un environnement et qui peut communiquer directement avec d'autres agents. Cette entité 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). Afin d'atteindre ses objectifs, l'agent tient compte des ressources et des compétences dont il dispose, ainsi que de sa perception et des représentations qu'il dispose de son environnement.

• Selon [Perret97], un agent est une entité active autonome agissant par délégation pour le compte d'un client.

• Dans [Wool03], on apprend qu'un agent est définit comme étant un système informatique situé dans un environnement et capable d'agir de manière autonome, afin d'atteindre ses objectifs de conception.

1.3.1.2. Propriétés

Les agents possèdent en général un nombre important de propriétés et de capacités parmi lesquelles, nous citons ici quelques unes qui nous intéressent:

• ***L'Autonomie:*** Un agent autonome est une entité qui agit dans son environnement indépendamment des commandes externes. L'autonomie d'un agent concerne aussi bien son comportement que ses ressources internes[Drieu]. Le ***«démon »*** est un exemple d'agent autonome.

• ***La Mobilité:*** Un agent mobile est capable de migrer d'une machine vers une autre pour atteindre ses objectifs.

• ***La Communication:*** Les agents peuvent communiquer entre eux ainsi qu'avec autrui. Par échanges de messages ou selon un mode de communication établit par l'architecture dans laquelle évoluent les agents tel que les SMA (Systèmes Multi Agents).

• ***L'Apprentissage:*** Un agent est capable d'augmenter ses capacités par apprentissage en utilisant ses connaissances.

• ***la Réactivité et La Pro-Activité:*** un agent réactif est caractérisé par sa réaction aux événements externes pour l'exécution de nouvelles tâches. Par contre, un agent Pro-Actif se caractérise par la capacité d'anticiper des changements plutôt que de réagir à ces changements.

1.3.1.3. Classes d'agents

Le terme agent prend des sens différents selon les domaines d'application et les perspectives attendues. La classification des différentes architectures d'agents se base sur des critères pertinents et un mode naturaliste (*taxinomie*). Il existe des classifications basées sur les *domaines* ou sur le *points de vue* [Briot01]. Dans cette section, nous présentons une classification comme un tour d'horizon.

• ***Agent cognitif:*** L'intelligence artificielle 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.

• ***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 œuvre. 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.

• ***Agent logiciel:*** cette appellation se justifie par opposition au terme agent physique, tel un robot. Les premiers Agents logiciels sont les démons d'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.

• ***Agent mobile:*** Né au milieu des années 90. Téléscript est la première plateforme d'agents mobiles. Elle a donné lieu par la suite, à de nombreuses autres plateformes.

1.3.2. Systèmes à Agents mobiles

1.3.2.1. Concepts de bases

Un agent qui s'exécute exclusivement dans le système où il a été créé est dit Agent statique. Si l'agent a besoin d'information non disponible dans le système où il s'exécute, il peut interagir avec d'autres agents dans un autre système. Dans ce cas, l'agent utilise un mécanisme de communication tel que RPC (Remote Procedure Calling). Par contre, un agent mobile n'est pas lié au système dans lequel il débute son exécution. Cet Agent 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 à un autre dans le réseau où il poursuit son exécution.

Un agent mobile est capable donc de se déplacer dans son environnement. Il peut être physique (réel ou simulé) ou structurel (niveaux d'exécution par exemple). Un agent mobile dispose donc des dispositifs assurant sa mobilité [Briot01].

Un système d'agent (aussi appelé serveur d'agent) est l'espace de vie des agents. Il est associé à une autorité qui identifie l'organisation ou la partie pour laquelle les agents agissent. La diversité des systèmes ainsi développés, rend les passerelles entre de tels systèmes souvent impossibles. Afin de permettre l'interopérabilité entre les systèmes à agents mobiles, l'OMG (Object Management Groupe) a montré son intérêt pour cette classe d'application par la définition des spécifications de MASIF (Mobile Agent System Interoperability) [MASIF]. Par ailleurs, la FIPA (Foundation For Intelligent Physical Agents) [FIPA] a lancé un autre effort pour la spécification de l'architecture des systèmes d'agents mobiles.

1.3.2.1.1. Mobilité des Agents

La mobilité d'un agent se mesure par rapport à sa migration. La migration d'un agent caractérise le déplacement d'un agent en cours d'exécution d'une machine

d'origine vers une autre machine. Lors de sa migration, l'agent transporte son code, son état d'exécution ainsi que ses ressources selon la légende suivante:

- L'agent ainsi que ses canaux de communication seront suspendus sur la machine d'origine.
- L'état de l'agent sera extrait et transféré sur le réseau, en tant que données, vers la machine de destination où il sera restitué.
- L'agent repend son exécution sur la machine de destination après avoir été réactivé. Ses canaux de communication seront ensuite mis à jours.

Les détails de la légende citées ci-dessus déterminent la classe de la mobilité de l'agent. Généralement, on distingues deux grandes classes pour la mobilité: la mobilité forte et la mobilité faible.

a- La mobilité forte : Dans cette classe, l'agent se déplace entièrement avec tous ses composants vers la machine de destination où il reprendra son exécution à partir de la dernière instruction interrompue. Ce déplacement, peut être proactif ou réactif. Dans cette classe, on distingue deux types de mobilités :

Le clonage : Dans ce type de mobilité, une copie conforme de l'agent mobile est transféré sur le réseau et l'autre copie reste au niveau de la machine d'origine. Les stratégies de communication de l'agent doivent être clairement spécifier pour ce type de mobilité.

La Migration : Ce type de mobilité définit la migration proprement dit. L'agent mobile sera transféré vers la machine de destination sans laisser de copie sur la machine d'origine.

b- La mobilité faible : Dans cette classe, l'agent mobile est dépourvu de son contexte d'exécution lors de sa migration. Il doit redémarrer son exécution sur le site de destination depuis le début. Plusieurs stratégies de récupération peuvent être intégrées dans les applications utilisant cette classe de mobilité. De la même manière que pour la mobilité forte, dans cette classe aussi le déplacement des agents mobiles peut être proactif ou réactif.

1.3.2.1.2. La place

Dans l'environnement d'accueil de l'agent migrateur, la place peut être vue comme étant un agent fixe qui peut recevoir l'agent mobile en lui offrant des services (figure 1.5). Par exemple, lorsqu' un agent mobile se déplace vers un serveur d'agents, il rentre dans la place que lui offre le service demandé. Un serveur d'agents peut être considéré comme étant une place dans un système multi agents. La place de départ et la place de destination de l'agent migrateur peuvent se situer au sein d'un même système d'agents

ou sur des systèmes différents. Les places sont identifiées de manière unique dans le système considéré.

1.3.2.1.3. La Région

Une région modélise un ensemble de systèmes à agents mobiles pas nécessairement de même type (cf. Figure 1.5). Elle représente l'ensemble de toutes les places et tous les agents qui appartiennent à une même autorité(organisation).

1.3.2.1.4. Le nom d'un agent

L'identité d'un agent doit être unique dans sa région, afin de pouvoir l'identifier. Le nom attribué à un agent peut être combiné en utilisant son identité et d'autres informations utiles.

1.3.2.1.5. La localisation

La localisation d'un agent mobile est définie par la combinaison de sa place d'exécution et l'adresse réseau du système d'agent où réside la place.

1.3.2.1.6. Autorité d'Agent et de Place

L'autorité d'un agent permet d'identifier son organisation d'affiliation, afin de lui autoriser l'accès à des ressources demandés. Pareillement, une place possède l'autorité de la région de sa création.

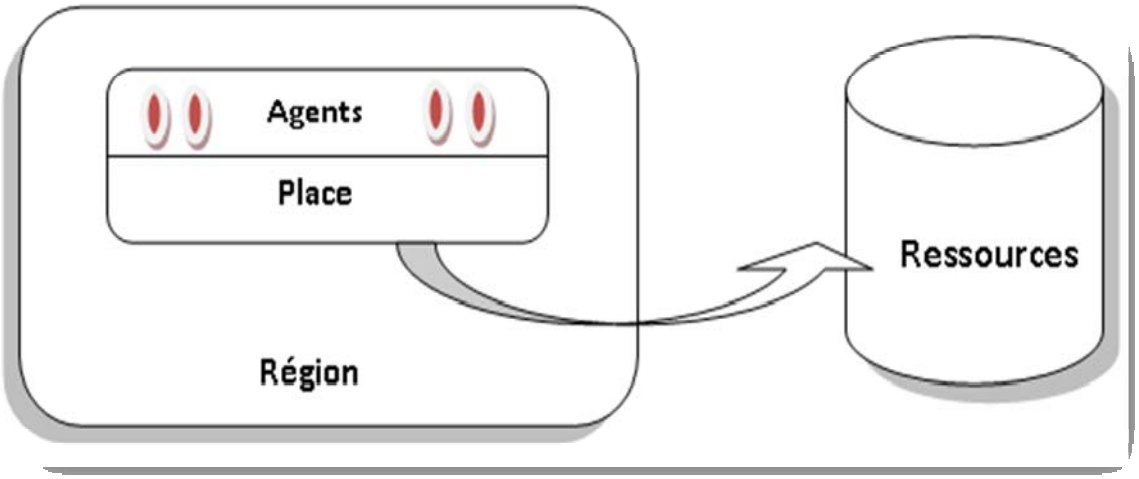


Figure 1.5. Place et Région

1.3.2.2. Services système

Nous décrivons dans cette section les éléments utiles d'un système à agents mobiles pour la construction des applications.

La création d'un agent se fait généralement au sein d'une place spécifiée. Le créateur (le système ou autres) doit s'authentifier dans la place et établir les autorisations et les références que va posséder l'agent. La création de l'agent passe par les étapes suivantes:

• ***Instanciation et nommage de l'agent :*** Au début le système charge et exécute le code de la classe agent. L'objet agent est ensuite instancié. A la fin, un identificateur unique est attribué pour l'agent ainsi créé.

• ***Initialisation de l'agent :*** Le créateur de l'agent fournit les arguments nécessaires à l'agent pour permettre son initialisation. Après son initialisation, l'agent est considéré complètement installé dans sa place.

• ***Migration d'un agent mobile :*** Un système à agent mobile offre pour chaque agent créé tous les mécanismes nécessaires qui assurent sa mobilité. Le processus de migration peut être initié par l'agent lui-même ou par autrui. L'agent migre selon ses objectifs de sa place courante vers la place de destination spécifiée. (cf. Figure 1.6)

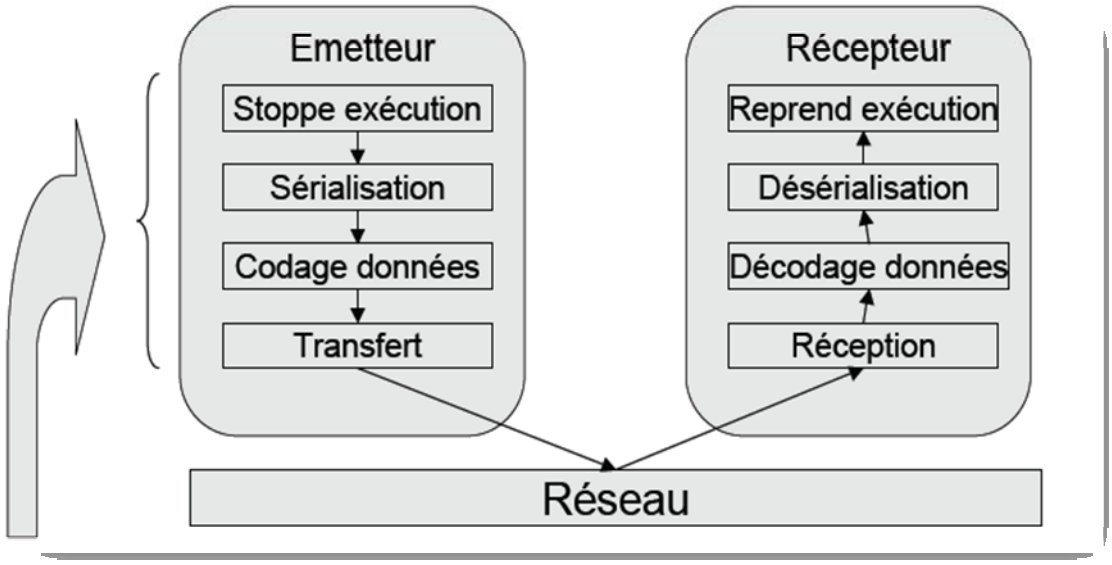


Figure 1.6. Le transfert de l'Agent Mobile

• ***La communication des agents :*** La communication entre les agents peut avoir lieu entre des agents résidents ou non dans la même place. Selon l'autorisation attribuée à un agent, celui-ci peut évoquer une méthode d'un autre agent ou lui envoyer un message. Cette communication peut suivre différents schémas:

a-*Now-type messaging:* Ce type de messagerie synchrone et le plus utilisé. L'exécution de l'émetteur est bloquée jusqu'à ce que le récepteur aura complètement téléchargé et envoyé la réponse du message reçu. (cf. la figure 1.7)

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. (cf. la figure 1.7)

c-*One-way-type messaging*: C'est un type de messagerie asynchrone qui ne bloque pas l'exécution courante de l'émetteur qui ne retiendra pas une variable pour le message envoyé sachant que le receveur ne va pas répondre. Ce type de messagerie 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.* (cf. la figure 1.7).

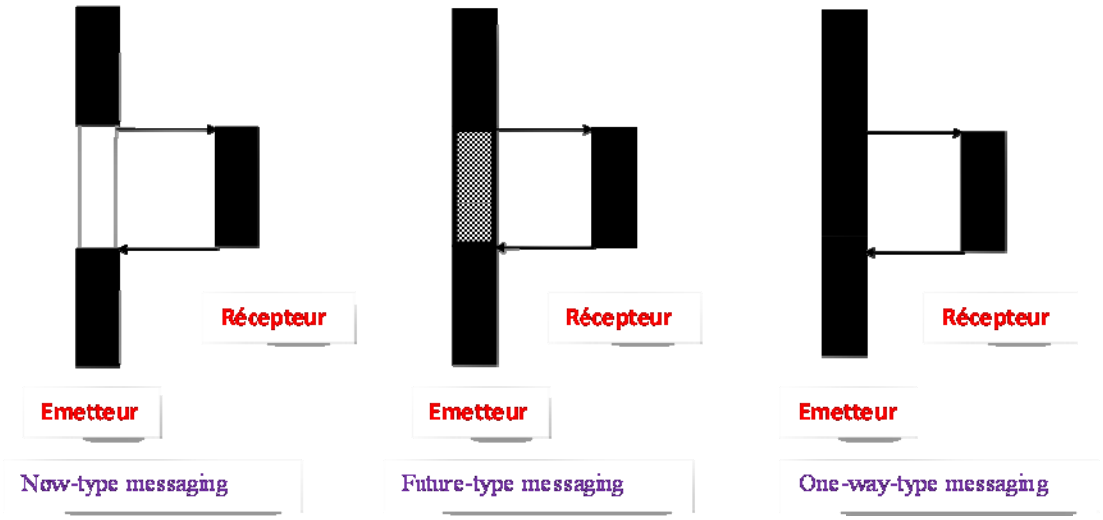


Figure 1.7 La communication entre les agents mobiles

• ***La Sécurité*** **:** De part la nature répartie et hétérogène des systèmes distribués, la sécurité joue un rôle prépondérant pour le développement de tels systèmes. Pour mettre en place une politique de sécurité visée pour le développement des systèmes à base d'agents mobiles, les concepteurs utilisent généralement des mécanismes d'authentification (mot de passe, certificat,...), de cryptographie (SSH, SSL,...) et de contrôle d'accès (droit utilisateur, pare-feu). 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 du 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, le problème de sécurité n'est pas entièrement résolu. C'est d'ailleurs l'argument de base expliquant 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.

1.3.3. Domaine d'application des agents mobiles

Bien que les agents mobiles soient introduits récemment, plusieurs domaines d'application impliquant les agents mobiles comme nouvelle technologie se sont manifestés, parmi lesquels nous citons :

• ***Le 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.

• ***La Compression des documents :*** La récupération des documents est l'un des grand chantier du Web. Par exemple, si un client veut obtenir une copie d'un document, il doit formuler une demande auprès d'un 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 de données. Pour cela, le serveur doit être étendu pour pouvoir assurer cette fonction de compression. Différents clients peuvent utiliser différents outils de compression, cela nécessite des extensions particulières pour chaque client.

• ***Services de télécommunication :*** La fonctionnalité de mobilité prend toute son ampleur dans le domaine des télécommunication. Les agents mobiles peuvent être utilisés pour couvrir les couches de protocoles de communication, de la maintenance des réseaux 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, quel 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 message vocal. La mobilité permet à ces agents de suivre l'utilisateur même quand il change de machine.

• ***Robotique :*** Les agents peuvent aussi contribuer à l'amélioration du rendement des robots physiques. À l'aide de capteurs, les robots se basent dans leur 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.

1.3.4. Présentation des quelques plateformes d'agents mobiles

Dans cette partie du chapitre nous présentons selon[Bl-MO98] quelques plateformes d'agents mobiles :

• **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. Il a été basé sur le système TCL (Tool Command Language) et plus tard, a été étendu pour supporter Java, Scheme et C/C++. Il est aussi 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 durant 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).

• **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'. Il 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.

• **Grasshopper** : La plateforme 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 plate-forme 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 et RMISSL).

• **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]**.

• **TACOMA :** Le projet TACOMA (Tromsô And COrnell Moving Agents) fournit un support dans les systèmes d'exploitation pour la programmation d'agents mobiles. Le modèle mélange client, serveur et agent. Il propose la notion d'agents et des mécanismes de migration et de communication. Un agent TACOMA est un processus exécutant un script TCL 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.

• **Voyager :** Voyager est un middleware développé par ObjectSpace en 1997. Il est implémenté en Java une plateforme 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 plateforme 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. En plus Voyager est capable d'exploiter des composants JavaBeans.

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

Selon les 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.

1.3.5. Apports et limites du paradigme des agents mobiles

Malgré leur jeune apparition, les agents mobiles se sont rapidement imposés en particulier dans le domaine de la recherche portant sur les applications réparties. Ce paradigme a été aussitôt évalué afin de vérifier ce qu'il pouvait apporter comme avantages par rapport aux méthodes de programmation classiques. Dans la suite de cette section, nous présentons quelques aspects pour l'évaluation des agents mobiles :

• ***Les performances :*** Les premières améliorations apportées par les agents mobiles portent sur le gain de performances dues à une meilleure utilisation des ressources physiques employées.

• **La *Réduction du trafic réseau :*** Le déplacement des agents mobiles permet de réduire significativement, les communications distantes entre les clients des réseaux et les serveurs. Cette réduction apporte les avantages suivants:

-La diminution de la consommation de bande passante. En effet, des études montrent que 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, en comparaissant avec l'envoi à distance de requête (procédures et méthodes). Cette diminution est constatée dans différents types d'applications nécessitant d'intenses échanges d'informations entre le client et le serveur. A titre d'exemple, la collecte d'informations dans des bases de données réparties et l'exploration d'Internet.

-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 propre 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, voir sur un même site, on les place dans un environnement où le temps de réponse des interactions sera limité, ce qui permet de réduire d'autant les temps de latence.

-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 minimiser les ruptures de liens physiques qui peuvent intervenir fréquemment dans les environnements sans fil.

• ***Indépendance des calculs:*** Dans le modèle d'évaluation à distance, le client et le serveur doivent rester connectés tant que le service est en cours d'exécution. Par ailleurs, certains services, nécessitant de longues phases de traitement, ne supportent pas toujours les ruptures de connexion avec les clients. D'autre part, le maintien des lien des communications dans les réseaux à large échelle ou sans fil, s'avère très difficile. Les agents mobiles offrent aux clients la possibilité de déléguer les interactions avec le service sans maintenir une connexion de bout en bout.

• ***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 tel que la disparition d'un nœud, ou d'ordre fonctionnel comme l'arrêt d'un service. Si dans un site par exemple un service tombe en panne, l'agent utilisateur de ce service pourra alors choisir de se déplacer vers un autre site contenant la fonctionnalité désirée. Ceci permet une meilleure tolérance aux fautes.

• ***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.

En plus, 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 s'adaptent aux caractéristiques des services afin d'exprimer leur requêtes. 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.

D'autre part, la capacité de raisonnement des agents permettra de concevoir des agents 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.

• ***Le développement et l'insuffisance de la standardisation:*** Le domaine des agents mobiles étant encore relativement jeune, il se heurte 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 dont chacun possède ses propres défauts et qualités [Cubat05]. Avec cette offre pléthorique, il est difficile de parler de standardisation.

Jusqu'à l'heure, il n'existe pas de langage standard partagé par toutes les plateformes de développements des applications d'agents mobiles. 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 en place des langages précis en s'appuyant sur un ensemble d'ontologies et de protocoles caractérisant les interactions inter-agents.

4. Conclusion

Nous avons présenté dans ce chapitre une introduction générale au paradigme d'agents mobiles. Au début, nous avons introduit le concept d'agents en général pour pouvoir ensuite faire une brève présentation du paradigme des agents mobiles. Nous avons enchainé ensuite sur quelques exemples de plateformes d'implémentation des applications d'agents mobiles. pour conclure le chapitre par des critiques du paradigme.

Néanmoins, le développement des applications des agents mobiles se heurte à des problèmes inhérents à la sécurité, et à l'interopérabilité dans les différentes plateformes d'accueil hétérogènes. Les insuffisances des efforts pour la standardisation de ces plateformes, ont poussé les chercheurs à explorer d'autres horizons pour une meilleure prise en charge et mise en œuvre des applications à base d'agents mobiles.
D'autre part, très peu de travaux de recherches portent sur les méthodes et les outils pour l'analyse et la conception des systèmes d'agents mobiles.

Dans Le chapitre suivant, nous présenterons une approche de modélisation des applications d'agents mobiles, basés essentiellement sur le standard UML.

Chapitre2

Les Agents Mobiles et la Modélisation UML

2.1. Introduction

Un agent mobile est une entité logicielle pouvant se déplacer de façon autonome d'un site du réseau à un autre pour atteindre ses objectifs. Bien que les agents mobiles soient introduits récemment, deux grandes normes de standardisation (FIPA et MASIF [MASIF]) et un grand nombre de plateformes [MASIF] sont apparus. Cependant, le domaine d'agents mobiles se heurte à des contraintes lors des phases de développement. Les prédominantes approches orientées-agents existantes cherchent à adapter les méthodologies d'analyse et de conception *orientées objets* [Bahri09] pour le développement des applications d'agents mobiles.

Il est avantageux d'obtenir une approche qui pourra être utilisée tout au long des phases de développement des systèmes à agents mobiles et qui inspire ses notations de l'analyse et de la conception orientée objet (en particulier du langage UML). Toutefois, un sérieux problème se pose. Il s'agit en fait, de la relation entre les agents et les objets.

Dans ce chapitre nous allons introduire les éléments essentiels du langage de modélisation UML. Après un survol des travaux essentiels que nous avons jugés intéressants pour modéliser les agents mobiles avec UML, nous présentons notre contribution qui consiste en l'extension des différents diagrammes d'UML 2.0 pour la prise en charge de la mobilité. Nous terminons par présenter une illustration de notre approche à travers une étude de cas d'un système de bourse électronique.

2.2. Agents Mobiles et modélisation UML

2.2.1. La modélisation

Un modèle est une représentation abstraite d'un système réel, construit pour un objectif donné. Il contient un ensemble restreint d'informations sur le système modélisé. Le modèle construit contient toujours des informations pertinentes vis-à-vis de son utilisation future.
Par ailleurs, la modélisation d'un système aide à la compréhension de son fonctionnement, ainsi qu'à la maitrise de sa complexité bien avant sa réalisation. Ceci offre des avantages considérables aux concepteurs de tels systèmes.

La modélisation en informatique peut-être vue comme la séparation des différents besoins fonctionnels et préoccupations extra-fonctionnelle [MDA05] (telles que: la sécurité, la fiabilité, l'efficacité, la performance, la ponctualité, la flexibilité, etc.)

En générale, la réalisation d'un système fait l'objet d'un travail d'équipe. La communication entre les différents membres de l'équipe de travail détermine la réussite de tout projet. Le modèle se définit comme étant un langage commun aux membres de l'équipe, favorisant ainsi la communication et la compréhension entre les différentes parties concernées par le projet. Par exemple, dans l'ingénierie du logiciel le modèle permet une meilleure répartition des tâches entre les différents intervenants. Ceci joue un rôle important quant à la réduction des coûts et des délais de réalisation des projets.

2.2.2. Types de Modélisations

La classification de la modélisation peut se faire selon le degré du formalisme des langages ou des méthodes impliquées dans le processus de la modélisation. Ainsi, la modélisation peut être considérée comme étant formelle, semi-formelle ou informelle [José]. La table de la figure 2.1 ci-dessous présente une définition des catégories de langages ainsi que des exemples de langages ou de méthodes qui l'utilisent.

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

Figure 2.1. Classification et Utilisation de Langages ou de Méthodes **[José]**

• ***Modélisation Informelle :*** Le processus de modélisation informelle à base de langages informels, se justifie selon [José] pour plusieurs raisons:
- La facilité de compréhension du langage permet des consensus entre les personnes qui spécifient et celles qui commandent un logiciel;
- elle représente une manière familière de communication entre personnes.

Par ailleurs, l'utilisation d'un langage informel rend la modélisation imprécise et parfois ambigüe. Le caractère informel de cette approche rend difficile toute tentative de standardisation.

• ***Modélisation Semi-Formelle***

Le processus de modélisation semi-formelle est basé sur un langage textuel ou graphique pour lequel une syntaxe précise est définie [José]. La sémantique d'un tel langage et souvent assez faible. Néanmoins, ce type de modélisation permet d'effectuer des contrôles et de réaliser des automatisations pour certaines tâches.

La plupart des méthodes de modélisation semi-formelles s'appuient fortement sur des langages graphiques. Ceci se justifie par la puissance expressive du modèle graphique. Par ailleurs, l'appui de la modélisation semi-formelle (tels que : UML) sur des langages graphiques, permet la production de modèles assez faciles à interpréter.
Cependant, cette modélisation soufre de la déficience des aspects sémantiques impliqués dans l'approche. Afin de palier aux insuffisances de cette approche, l'utilisation de contraintes a été introduite. Dans ce cadre, nous citons à titre d'exemple les travaux de S. Cook et J. Daniels [Cook94].

• ***Modélisation Formelle***

Une méthode formelle est un processus de développement rigoureux basé sur des notations formelles avec une sémantique définie. Le principal avantage des aspects formelles est leur capacités à exprimer une signification précise, permettant ainsi des vérifications de la cohérence et de la complétude d'un système. Par exemple J. P. Bowen et M. C. Hinchey [Hin95] montrent qu'avec une traduction appropriée, les méthodes formelles peuvent aider à la compréhension d'un système par un utilisateur.

2.2.3. Modélisation Orientée Objet

L'approche orientée objet se classe dans la modélisation semi-formelle. Elle considère le logiciel comme une collection d'objets dissociés. La fonctionnalité du logiciel émerge de l'interaction entre les différents objets qui le constitut. [Aud07].

Cette modélisation a connu une large utilisation. Toutefois, l'absence de standards uniformisant l'approche au début des années 90 a donné lieu à une génération multi variée de logiciels hétérogènes et souvent dupliqués, dispersant ainsi l'effort de la communauté informatique. Au milieu des années 90 des prémices de standardisation apparaissent donnant naissance à un langage unifié sous l'appellation abréviée ***UML***.

2.2.4. UML*(The* Unified Modeling Language for object-oriented developement *)*

UML Selon l'OMG[OMG03] est un langage de modélisation qui permet de spécifier, visualiser, construire et documenter les artefacts des systèmes logiciels, ainsi que la modélisation des systèmes non logiciels**.**

2.2.4.1. Historique

La naissance d'UML est dû à la fusion des trois méthodes de références dans le domaine de la modélisation objet durant les années 1990. En effet, il s'agit de la fusion en 1994 des deux méthodes, celle de Grady Booch Booch-93 adaptée à la construction et OMT-2 (*Object Modelling Technique*) celle de James Rumbaugh qui se concentre sur l'analyse et l'abstraction. Cette fusion a donné naissance à une nouvelle méthode : la méthode unifiée(*The Unified Methode*). Plus tard, en 1995 Ivar Jacobson, le créateur des cas d'utilisation (*Use Cases*), rejoint le groupe en fusionnant avec Booch-93 et OMT-2 la méthode OOSE (*Object Oriented Software Engineering*) dont l'objectif essentiel est la détermination des besoins du système. En 1997 en sa version 1.1 UML a été adopté par l'OMG (Object Management Group). A la fin de 2006 la version UML 2.0 devient un langage de modélisation de logiciels standardisés (cf. la figure 2.2).

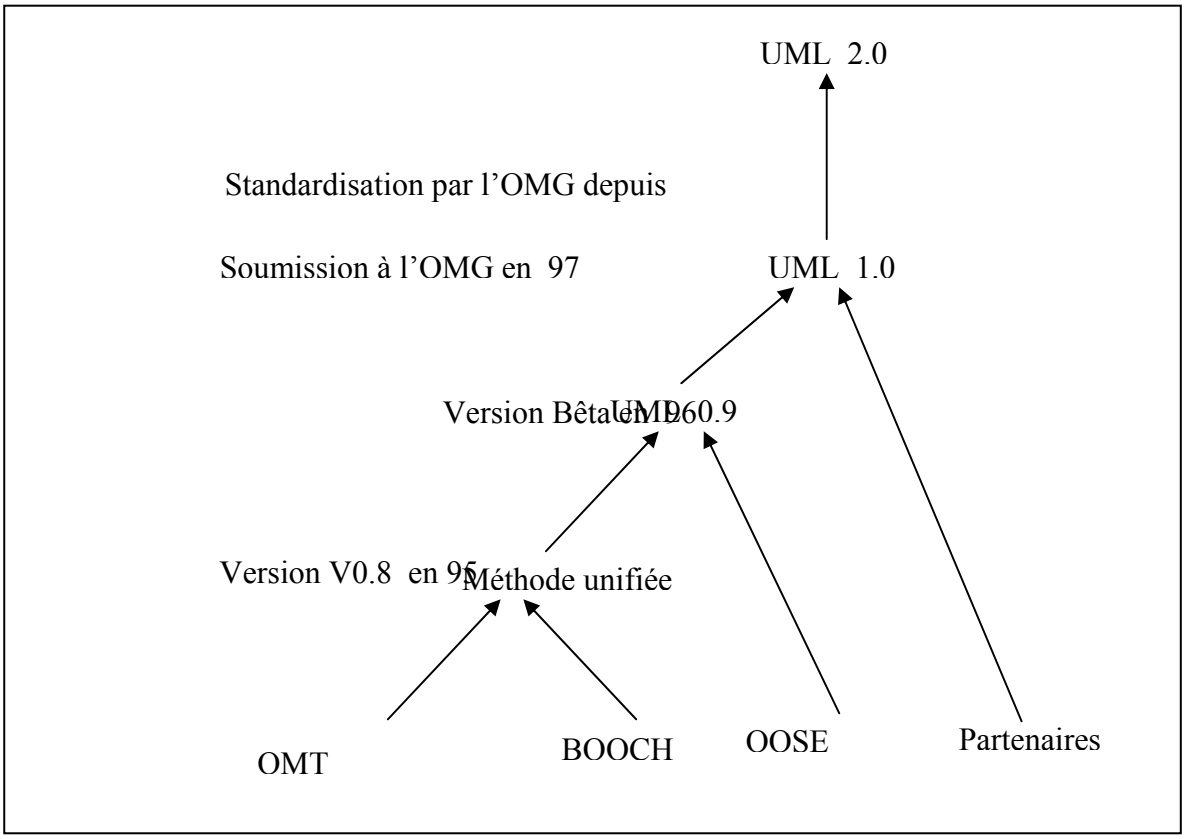


Figure 2.2. Evolution de UML.

2.2.4.2. Les diagrammes UML

UML puise des avantages de la modélisation semi-formelle en offrant à l'utilisateur un moyen de visualiser et de manipuler des éléments de modélisation. Les éléments de représentation sont le plus souvent des graphes connexes où les sommets correspondent aux éléments et les arcs aux relations. Ces graphes servent à visualiser un système sous différentes vues.

UML propose un certain nombre de diagrammes qui servent à visualiser un système sous différentes perspectives. Pour les systèmes complexes, un diagramme ne représente qu'une vue partielle des éléments qui composent ces systèmes.

UML comporte un ensemble de diagrammes représentant des vues distinctes pour modéliser des concepts particuliers du système. Ils se répartissent en deux grands groupes [Aud07]:

- **Diagrammes structurels ou statiques du système**:

- *Les diagrammes de cas d'utilisation* représentent les fonctions du système du point de vue utilisateur.
- *Les diagrammes de classes* expriment la structure statique d'un système, en termes de classes et de relations entre ces classes.
- *Les diagrammes d'objets* sont des graphes d'instances des diagrammes de classes.
- *Les diagrammes de composants* présentent les dépendances entre les composants logiciels du système.
- *Les diagrammes de déploiements* illustrent le déploiement des composants du système sur les dispositifs matériels.

- **Diagrammes comportementaux ou dynamiques du système :**

- *Les diagrammes de séquence* donnent une représentation temporelle des objets du système ainsi que leurs interactions.
- *Les diagrammes de collaborations* donnent une représentation spatiale des objets, de leur liens, ainsi que de leur interaction.
- *Les diagrammes d'états-transitions* présentent les automates d'états finis du point de vue des états et des transitions des objets dans le système.
- *Les diagrammes d'activités* représentent les comportements des opérations en terme d'action.

Ces diagrammes servent à visualiser un système sous différentes vues. Dans cette optique, une modélisation UML reproduit uniquement les diagrammes utiles dans la perspective des vues adoptées pour l'étude d'un système.

2.2.4.3. Les différentes vues d'un système

Les différentes vues d'un système représentent sa formulation. Elles sont indépendantes et complémentaires. Ces vues (cf. la figure 2.3) permettent de définir l'architecture du système modélisé [BRJ01]. Chaque vue est une projection, selon un aspect particulier, dans l'organisation et la structure du système.

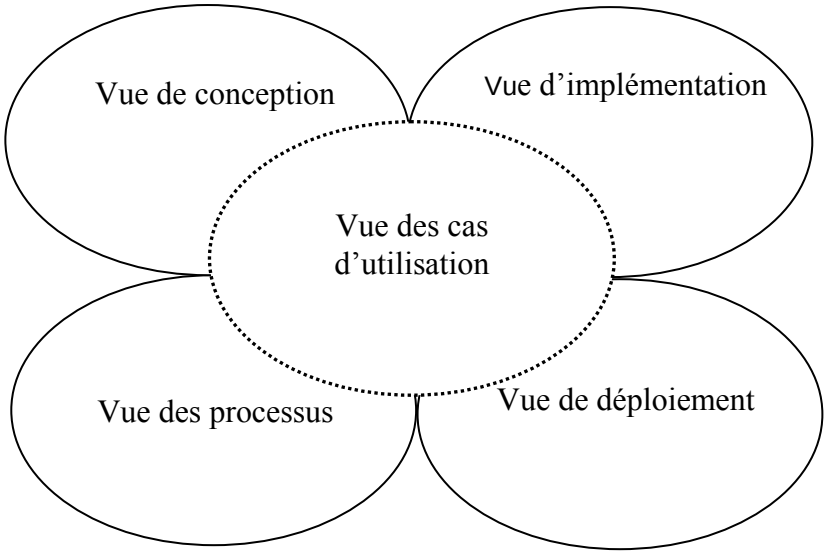


Figure 2.3. Modélisation de l'architecture d'un système **[BRJ01]**

• La *Vue des cas d'utilisation* permet de décrire le système tel qu'il est vu par les acteurs du système lui-même.
• *La Vue logique* est d'ordre conceptuel, son objectif est de modéliser les éléments et les mécanismes principaux du système. Dans ce cadre, UML implique par exemple les classes et les interfaces comme éléments de modélisation.
• *La Vue d'implémentation* indique les différentes ressources nécessaires à la réalisation du projet, tels que : les bases de données, et autres. Cette vue permet de créer des liens entres les composants du système ainsi que leur regroupement en modules.
• *La Vue de déploiement* est d'un très grand intérêt pour les systèmes distribués. Elle définit une vue spatiale du projet.
•*La Vue des processus* représente la vue temporelle et technique en manipulant des notions tels que : les tâches concurrentes, la synchronisation, les processus, etc.

2.2.4.5. La méthodologie UML

Plusieurs processus de modélisation tirent avantage d'UML en utilisant les différentes vues mentionnées précédemment, pour modéliser les systèmes tels que : le processus d'ingénierie de logiciel RUP (**Rational Unified Process**[Kim07]), etc.
Néanmoins, certains aspects des systèmes à modéliser ne sont pas capturés directement par UML. Pour palier aux insuffisances du langage de modélisation UML, les concepteurs d'UML proposent des mécanismes d'extension pour l'enrichissement d'UML, entre autres les stéréotypes, les étiquettes, les contraintes, etc.

2.2.5. UML et les Agents Mobiles

2.2.5.1. Genèse

La nature diversifiée des systèmes réels et leur nombres considérable soulèvent un réel défi pour la modélisation UML. Toutefois, l'OMG [OMG04] propose des profiles UML pour le spécialiser dans des domaines spécifiques, tel que : MobileUML. les spécifications de ces profiles se trouvent au niveau des nouvelles classes de méta-modèles UML.

La modélisation des applications d'agents mobiles considère les concepts de la mobilité à savoir : la migration, le clonage, les locations, la sécurité, etc. Cette modélisation peut se faire selon la littérature exclusivement par trois approches **[Adl]:** l'approche pattern, l'approche formelle et l'approche semi formelle. Dans l'approche semi formelle on trouve deux classes.

- Une classe d'extension de la méthodologie orientée agent (agent-oriented methodologies), tel que l'on en trouve par exemple dans les travaux de MaSE [Mase03] et m-Gaia [m-Gaia04].
- La deuxième classe propose des extensions du langage UML pour modéliser les concepts propres à la mobilité. Cette classe présente l'avantage d'être plus universelle car elle s'adresse à une large communauté, celle d'UML.

Nous nous intéressons particulièrement à cette deuxième classe de modélisation des applications d'agents mobiles. Nous présentons en premier lieu des travaux relevant du domaine considéré, par la suite, nous exposons notre propre contribution dans ce domaine.

2.2.5.2. Exemples d'extension d'UML pour les applications à base d'agents mobiles

Dans cette section, nous allons exposer quelques travaux ayant trait à l'extension d'UML pour modéliser les applications à base d'agents mobiles.

- **AUML** : Les auteurs Mouratidis et al [Har] ont introduit une extension pour les deux diagrammes UML : le diagramme de déploiement et le diagramme d'activité. Les auteurs spécifient que le diagramme de déploiement ***(AUML deployment diagram***) permet de capturer les raisons de déplacement d'un agent mobile d'un nœud du réseau à un autre, ainsi que la vérification de la location actuelle d'un agent mobile. Par contre, le diagramme d'activité (***AUML activity diagram)*** permet de capturer le moment de déplacement d'un agent mobile d'un nœud du réseau à un autre.

- **Mario K et al [Kus]** proposent une extension pour le diagramme de séquence d'UML, afin de pouvoir modéliser le chemin de migration d'un agent mobile en prenant en considération la création de l'agent mobile, sa location actuelle ainsi que le chemin qui sera exploré par cet agent.

- **MA-UML:** Les auteurs H.HACHICHA, A.Loukil, et K.Ghedira [Hach], **[Adl]** proposent une approche pour modéliser et implémenter des applications à base d'agents mobiles. Les

auteurs matérialisent leur contribution par une notation UML appelée ***MA-UML*** pour modéliser les agents mobiles, et un outil ***CASE*** qui s'appelle **MAMT** pour le mapping entre les diagrammes conceptuels et les classes d'implémentation java. Les nouveaux diagrammes introduits par les auteurs se basent sur les diagrammes d'UML standard stéréotypés pour capturer les concepts de la mobilité.

• **M-UML:** K.Saleh et C.El-Morr [Kas01] proposent ***M-UML*** une extension du langage UML 1.4. Les auteurs ont ajouté pour chaque diagramme d'UML des signes graphiques et des stéréotypes pour spécifier et exprimer la mobilité.

Malgré la panoplie des applications réalisées pour la prise en charge des systèmes à base d'agents mobiles, ce domaine de recherche reste encore immature en l'absence d'une normalisation. Néanmoins, l'orientation des chercheurs vers UML comme norme permet à chaque fois d'améliorer les contributions réalisées, et ce, en puisant toujours des nouveautés du standard UML.
Dans la section suivante, nous présenterons notre contribution qui consiste à étendre UML 2.0 par la notion de mobilité.

2.3. Contribution1: Extension d'UML 2.0 par la Mobilité[Bahri09]

Le mécanisme de stéréotypage prévu dans UML permet aux utilisateurs (concepteurs, analystes, et autres…) d'ajouter de nouvelles classes d'éléments en plus des éléments prédéfinis. Un élément spécialisé par un stéréotype et sémantiquement équivalent à une nouvelle classe du méta modèle qui portera le même nom que le stéréotype.
Dans cette section, nous introduisons de nouveaux éléments enrichissant les diagrammes d'UML 2.0 par stéréotypage [Bahri09] à savoir : *diagramme de cas d'utilisation*, *diagrammes de séquence*s, *diagrammes de classes*, *diagrammes d'objets*, *diagrammes d'état transition*, *diagrammes d'activités*, et les *diagrammes de déploiements*. Ces éléments permettront de modéliser les concepts principaux caractérisant la notion de mobilité dans les agents mobiles: les *locations*, la *migration* et le *clonage*.
Par la suite, nous illustrons notre contribution par une étude de cas qui consiste en un système de bourse électronique à base d'agents mobiles.

2.3.1. Les diagrammes

Afin de supporter la mobilité, nous avons ajouté des concepts dans chaque diagramme d'UML 2.0.

2.3.1.1 Diagrammes de cas d'utilisation

2.3.1.1.1 Acteur mobile :

Un agent mobile est représenté dans un diagramme de cas d'utilisation comme un acteur mobile, il est stéréotypé par : **<<ma: nom de l'agent mobile>>** (cf. la figure 2.4). Un acteur mobile participe au cas d'utilisation au même titre que les autres acteurs d'UML standard.

<<ma: acteur mobile>>

Figure 2.4. L'acteur mobile

2.3.1.1.2. Association de migration <<send>> :

Une nouvelle relation est créée entre un acteur quelconque et un autre mobile à travers une association stéréotypée par : **<<send>>** (cf. la figure 2.5). Cette association indique (point de départ de la flèche) la première location (location originale : plateforme ou autres) depuis laquelle l'agent mobile pourra se déplacer.

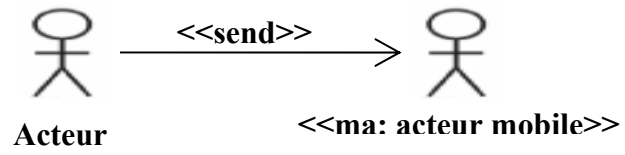


Figure 2.5. Association de migration

2.3.1.2. Diagrammes de séquences :

Les actions de clonage et de déplacement d'un agent mobile par rapport aux interactions des objets du système dans les diagrammes de séquences, ainsi que les locations vers lesquelles se déplacera cet agent, sont introduits par l'application des éléments suivants (cf la figure 2.6) :

1- <<**moveTo:** ***id-location***>>: l'application de l'action **moveTo** permet à l'agent mobile de se déplacer vers la location identifiée par ***id-location***

2- <<**clone:** ***nbr***>>: cette action permet à l'agent mobile de se cloner à un nombre entier noté :***nbr.***

3- <<**return**>>: cet élément permet à l'agent mobile de retourner vers sa location d'origine.

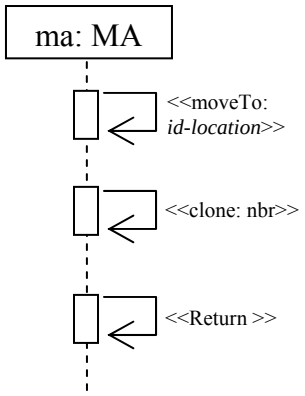


Figure 2.6. Nouveaux éléments introduits aux diagrammes de séquences.

2.3.1.3. Diagrammes de classes :

La classe ***MA*** *(Mobile Agent)* est une classe abstraite stéréotypée par : **<<mobile>>**. Elle doit être toujours présente lors de la modélisation d'un système à agents mobiles. Les classes mobiles du système sont alors toutes héritées à partir de cette classe. Cette classe **<<mobile>>** regroupe trois méthodes publiques(cf. la figure 2.7) indispensables à la réalisation des agents mobiles:

- **clone(int)**: Par cette méthode, l'objet agent mobile peut se cloner à un nombre entier déterminé.
- **moveTo()**: Cette méthode permet à l'objet agent mobile de se déplacer vers une location déterminée.
- **return()**: Enfin, cette méthode donne la possibilité à l'objet agent mobile de retourner vers sa plate-forme de base.

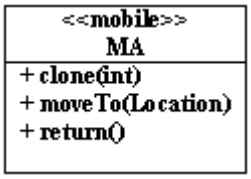


Figure 2.7. La classe mobile

2.3.1.4. Diagrammes d'objets :

Un objet mobile est créé par instanciation d'une classe mobile (cf. la figure 2.8). Nous introduisons dans les diagrammes d'objets les éléments ci-dessous, afin de montrer les différences entre les échanges des messages des objets du système et les actions de **clonage,** de **déplacement** ainsi que les **locations** vers lesquelles se déplaceront les agents mobiles :

1- **i**:<<moveTo: *id-location*>>: l'agent mobile doit se déplacer vers la location identifiée par *id-location.*
2- **j**:<<clone: *nbr*>>: l'agent dans ce cas doit se cloner à un nombre *nbr*.
3- **k**:<<return>>: dans cette situation l'agent mobile retourne vers sa location d'origine.

(Où: i, j, k…. indiquent l'ordre des actions par rapport aux autres messages échangés, comme sur la figure 2.9.

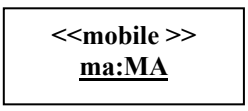


Figure 2.8. Objet mobile

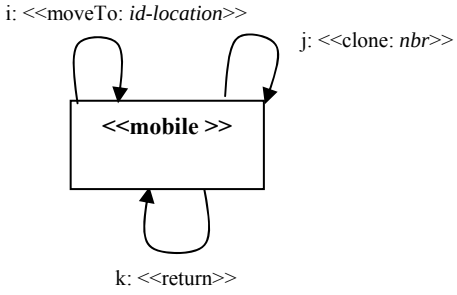


Figure 2.9. Les actions de clonage et de déplacement par rapport aux échanges des messages

2.3.1.5. Diagrammes d'état transition

Dans les partitions hiérarchiques multidimensionnelles (ou encore les travées hiérarchiques multidimensionnelles), une dimension représente une location et l'autre représente un objet. Dans notre diagramme d'état transition, une dimension représente une *location* stéréotypée par: <<**platform:** *nom d'un agent* ***X*** >> qui désigne la plateforme de base de l'agent (ou l'objet) ***X***. Les autres dimensions orthogonales représentent les objets, ainsi que les agents mobiles du système. Ces agents mobiles se distinguent des autres objets par l'utilisations du stéréotype: <<**mobile**>>.
L'événement provocant le déplacement d'un agent mobile d'une plate-forme quelconque vers une autre différente de celle de base est stéréotypé par <<**move**>>. Dans la figure 2.10 ci-dessous, l'événement ***Event1*** provoque l'agent mobile ***ma*** à se déplacer vers la location ***location2*** pour atteindre l'état ***State2***. En fin, L'événement permettant à un agent mobile de rejoindre sa plateforme de base est stéréotypé par <<**return**>>, comme le montre l'événement ***Eventi*** dans la figure suivante :

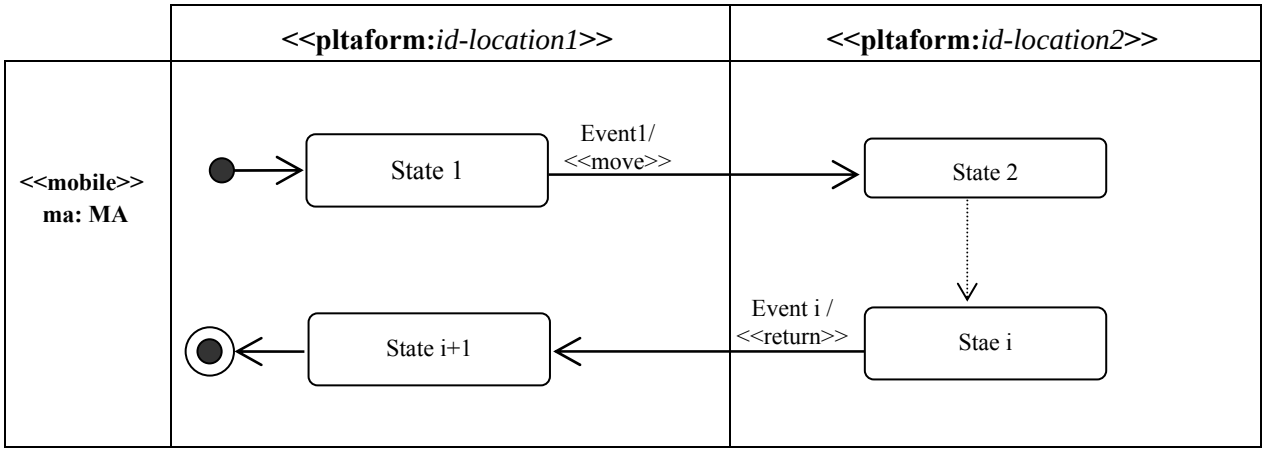


Figure 2.10. Diagrammes d'état transition

2.3.1.6. Diagrammes d'activités

Un diagramme d'activités contient généralement trois types de nœuds appelés « ***ActivityNodes*** » regroupant les nœuds : objet *(Object node)*, action *(Action node)*, et contrôle *(Action node)*.

UML 2.0 introduit le nœud paramètre *(ActivityParameterNode)* comme un nouveau type de nœud objet pouvant être placé aux entrées/sorties d'une activité, afin de contrôler le flot des objets au niveau des entrées/sorties de cette activité.
Nous proposons dans notre extension deux types de nœuds :

1-*MobileActivityAgentNode*: Un nœud agent mobile pour modéliser le flot d'un agent mobile dans les diagrammes d'activité, il est représenté par un nœud objet stéréotypé par <<mobile>>. Dans la figure 2.11, *ma2:MA* il représente un nœud agent mobile.

2-*MobileActivityParameterNode*: Un nœud paramètre mobile représenté par un nœud paramètre stéréotypé aussi par <<mobile>>, est utilisé pour modéliser le flot d'un agent mobile au niveau des entrées/sorties d'une activité. Dans la figure2.11, *ma1:MA*, *ma3:MA* et *ma4:MA* représentent des nœud paramètres mobiles

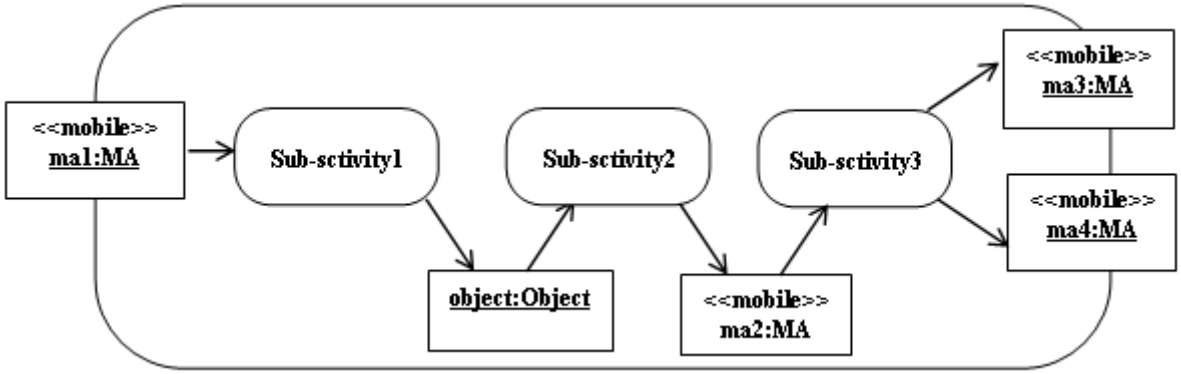


Figure 2.11. Les nœuds objets et paramètres mobiles dans les diagrammes d'activités

2.3.1.7. Diagrammes de déploiement

Le diagramme de déploiement est utilisé pour modéliser les systèmes répartis tels que le modèle client/serveur. Ce diagramme montre la distribution physique des unités de traitement des nœuds, ainsi que leurs composants internes. Un nœud dans un système mobile peut envoyer et recevoir des agents mobiles.

Dans notre diagramme de déploiement étendu, un nœud qui envoie des agents mobiles seront étiquetés par le stéréotype <<send>> suivi des noms de ces agents mobiles. Aussi, un nœud mobile qui reçoit des agents mobiles sera étiqueté par le stéréotype <<receive>> suivi des noms des agents mobiles.

Dans la figure 2.12, le nœud ***Node 1*** peut envoyer l'agent mobile ***MA1*** et recevoir l'agent mobile ***MA2.*** Le nœud ***Node 2*** peut recevoir les deux agents mobiles ***MA1*** et ***MA2.*** Alors que, le nœud ***Node i*** peut envoyer ***MA2*** et recevoir ***MA2.***

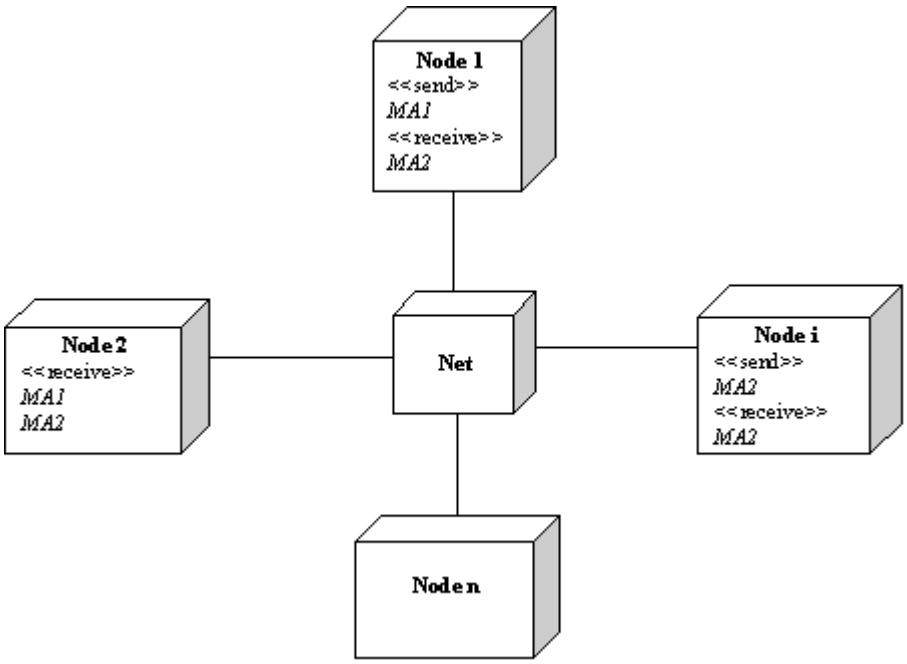


Figure 2.12. Diagramme de déploiement

2.3.2 Etude de cas : Un système de bourse électronique mobile

Pour réaliser un système de bourse électronique à base d'agents mobiles, nous supposons l'existence de plusieurs actionnaires (acheteurs et vendeurs). Chaque actionnaire possède un compte chèque au niveau d'une banque du système, et un compte d'actions au niveau de la bourse. Dès qu'une session boursière est ouverte, les actionnaires commencent à passer leurs ordres d'achat et de vente.
Le système regroupe deux types d'agents (mobiles et stationnaires):

-Les agents mobiles s'organisent en trois classes: les Acheteurs *(BMA: Buyer Mobile Agent),* les Vendeurs *(SMA: Seller Mobile Agent),* et les Boursiers *(PMA: Purse Mobile Agent*.

-Les agents stationnaires, par contre, sont représentés par une classe d'agents gestionnaires de comptes chèque *(CAM: Cheque Accounts Manager)*. Chaque agent *CAM**i*** réside dans une plateforme d'une banque appartenant au système. En plus, on trouve sur la plateforme de la bourse, deux autres agents stationnaires, un agent gestionnaire de comptes d'actions *(SAM: Share Accounts Manager)* et un agent gestionnaire de transactions *(TM: Transactions Manager).*

Lors d'une session boursière, dés qu'un actionnaire acheteur (i) passe un ordre d'achat d'une action particulière, un agent *BMAi* se déplacera vers la plateforme de la bourse où se trouve l'agent *TM,* en portant un ordre contenant les informations suivantes:
- l'identité de l'actionnaire acheteur,
- le nom de l'action souhaitée,
- la quantité demandée,
- le prix proposé.

Après l'arrivée de l'agent mobile *BMAi*, l'agent *TM* classe l'ordre d'achat s' il est accepté. Par ailleurs, si un actionnaire vendeur (j) passe un ordre de vente, il provoque le déplacement d'un agent mobile *SMAj* vers la plateforme de l'agent *TM* en portant l'ordre de vente qui contient les informations suivantes:
- l'identité de l'actionnaire vendeur,
- le nom de l'action proposée,
- la quantité et le prix demandés.

Enfin, cet ordre de vente sera classé par le *TM* de la même manière que celui d'achat.

Dans un temps déterminé, l'agent *TM* calcule le cours de chaque action mise en vente dans la session ouverte. Ensuite, il classe les meilleurs ordres d'achats et de ventes, selon une stratégie adoptée. L'agent *TM* réenvoie par la suite les agents mobiles *BMAs* et *SMAs* en provenance des actionnaires sélectionnés, afin de confirmer leurs ordres respectifs; selon le cas une transaction peut être affirmée ou annulée.
Si une transaction est affirmée, l'agent *TM* crée un agent mobile *PMA* qui se clonera à un nombre égal à celui de banques contenant les comptes chèques des actionnaires concernés par cette transaction. Chaque *PMAi* se déplacera vers la plate-forme où se trouve l'agent *CAMi*. Dés son arrivée le *PMAi* invoque le *CAMi* correspondant à mettre à jour tous les comptes visés, de telle sorte qu'il crédite les comptes des vendeurs et débite les comptes des acheteurs. Ainsi, le *TM* doit invoquer l'agent *SAM* à mettre à jour les comptes d'actions de ces actionnaires. Cet agent *TM* se chargera en plus de calculer régulièrement l'indice de la bourse.
Enfin, après avoir mis à jour les comptes chèque au niveau des plateformes des agents *CAMi* chaque agent mobile *PMA*i doit retourner vers sa plateforme d'origine (la plateforme de l'agent *TM*).

L'architecture générale de notre système boursier électronique ainsi que les agents mobiles sont montrés dans la figure 2.13.

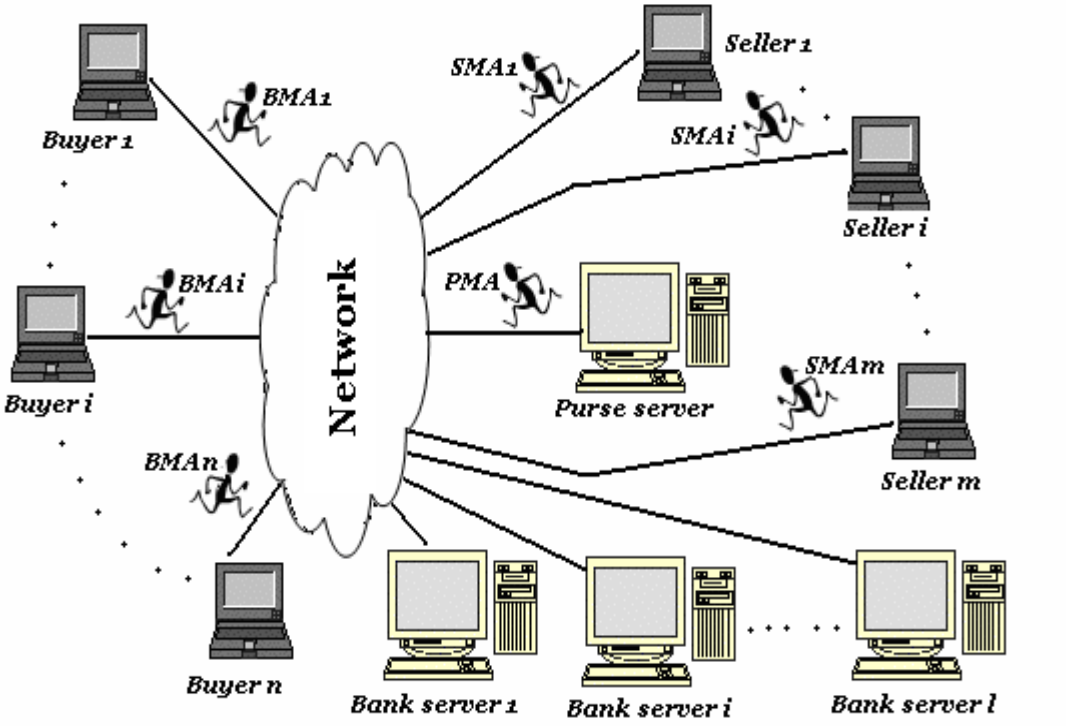


Figure 2.13. Un système de bourse électronique à base d'agents mobiles

Par projection des extensions d'UML2.0 proposées dans notre Approche. Nous présentons ci-dessous les différents diagrammes d'UML stéréotypés, pour le système de bourse électronique mobile.

2.3.2.1. Diagramme de cas d'utilisation

La figure 2.14 ci-dessous représente le diagramme de cas d'utilisation du système boursier, notre système contient les deux types d'acteurs mobiles et non-mobiles.

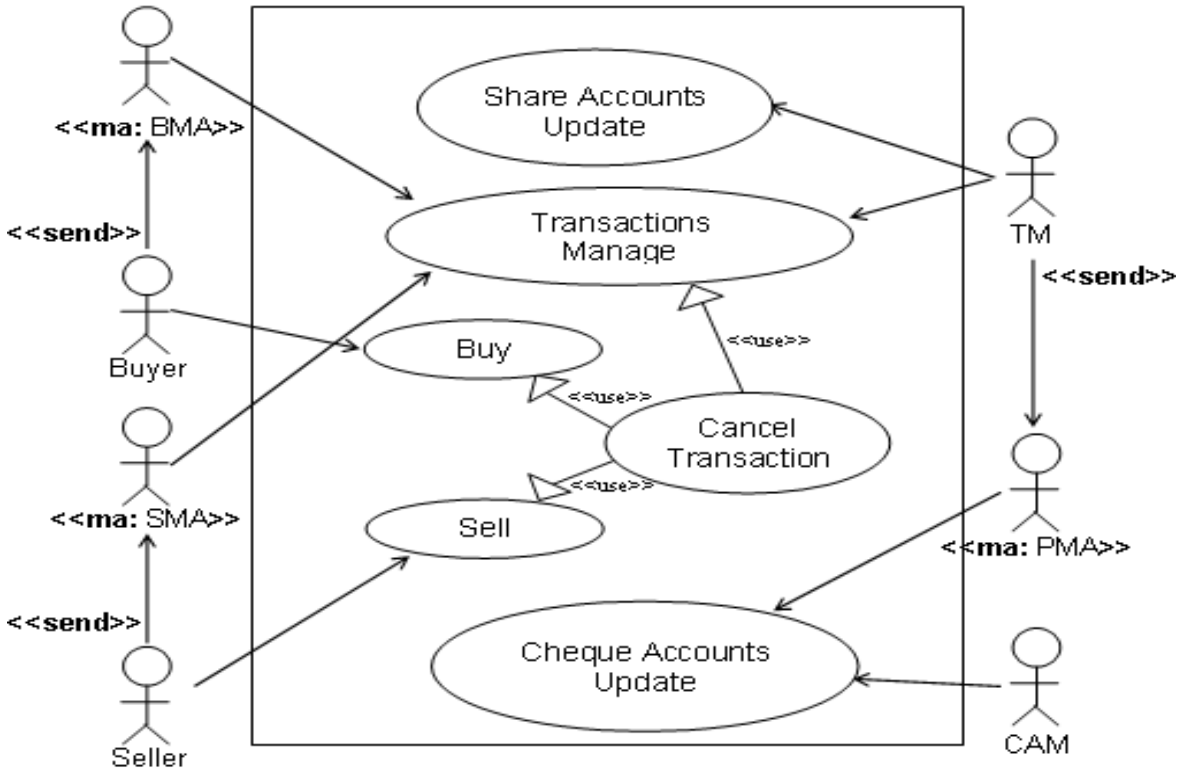


Figure 2.14. Diagramme de cas d'utilisation du système boursier

-Un agent mobile acheteur *BMA* pourra être envoyé d'une plateforme d'un acheteur afin de participer avec l'acteur *TM* au cas d'utilisation *Transactions Manage*.

-Un agent mobile vendeur *SMA* pourra être envoyé d'une plateforme d'un vendeur afin de participer aussi au cas d'utilisation *Transactions Manage*..

-De la même manière, un agent mobile boursier *PMA* pourra être envoyé de la plateforme de l'agent *TM* à une plate-forme d'un agent *CAM* particulier afin de participer avec lui au cas d'utilisation *Cheque Accounts Update.*

2.3.2.2 Diagramme de séquences

Le diagramme de séquences suivant détaille le cas d'utilisation *Buy* (*acheter*).

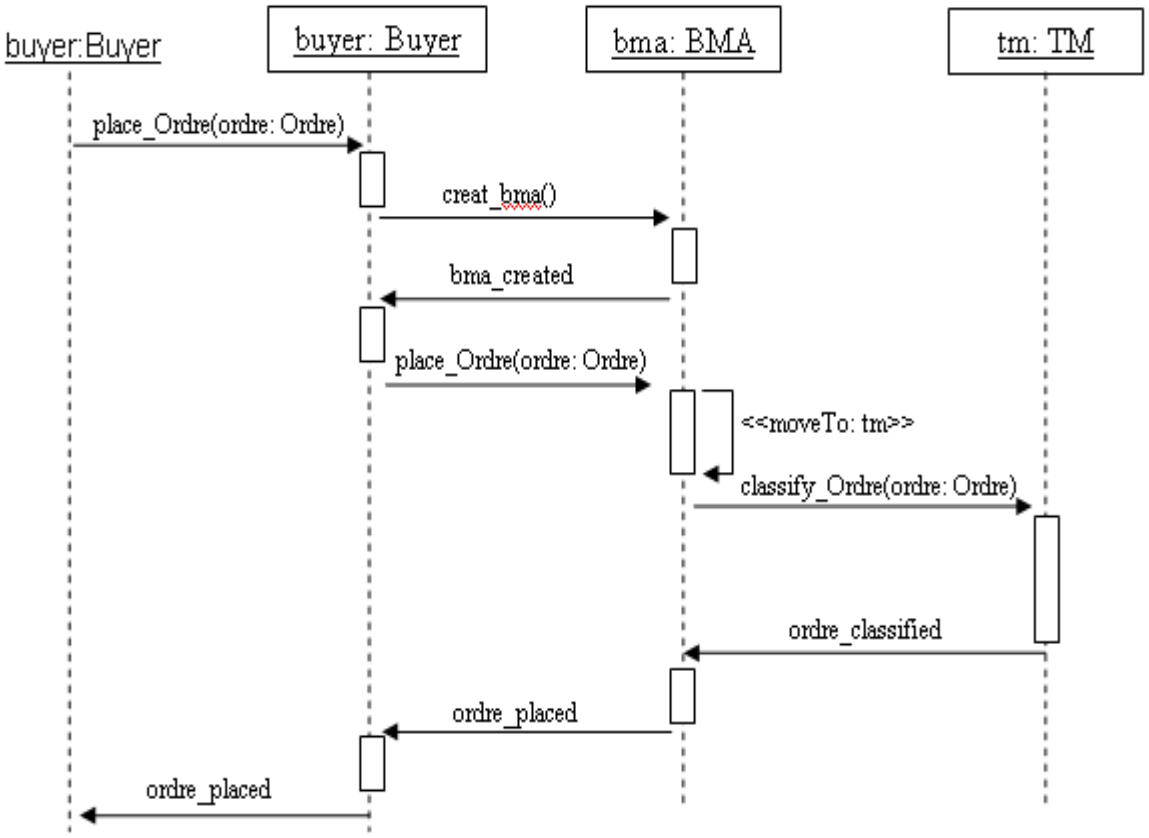


Figure 2.15: Diagramme de séquence de passation d'un ordre d'achat

Le nouvel élément stéréotypé par <<moveTo: *identifiant d'une plate-forme*>> nous permet de capturer les déplacements d'un agent mobile sur son axe de temps. Par conséquent, nous pouvons connaître les locations à partir desquelles l'agent mobile interagit avec les autres objets du système. Par exemple, après avoir achevé l'action de déplacement présentée par <<moveTo: tm>> l'agent mobile *bma:BMA* peut interagir localement avec l'agent *tm:TM*.

2.3.2.3. Diagramme de classes

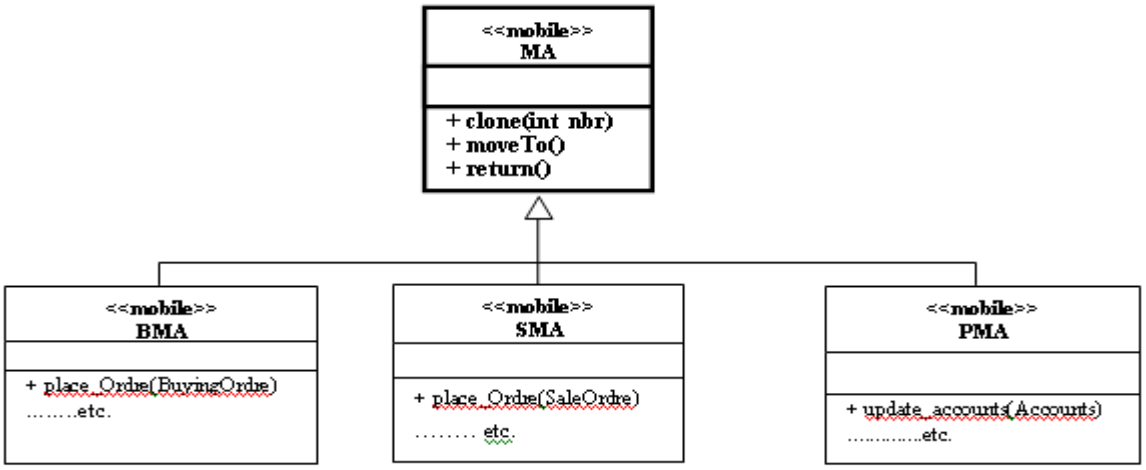


Figure 2.16. Diagramme de classes du système boursier et "la classe mobile"

Ce qui nous intéresse dans le diagramme de classe de notre système boursier est la classe mobile abstraite **MA**, à partir de laquelle les trois autres classes mobiles du système boursier (*BMA: Buyer Mobile Agent, SMA: Seller Mobile Agent et PMA: Purse Mobile Agent*) sont héritées, toutes ces classes sont stéréotypées par <<mobile>> (cf. la figure 2.16).

2.3.2.4 Diagramme d'objet

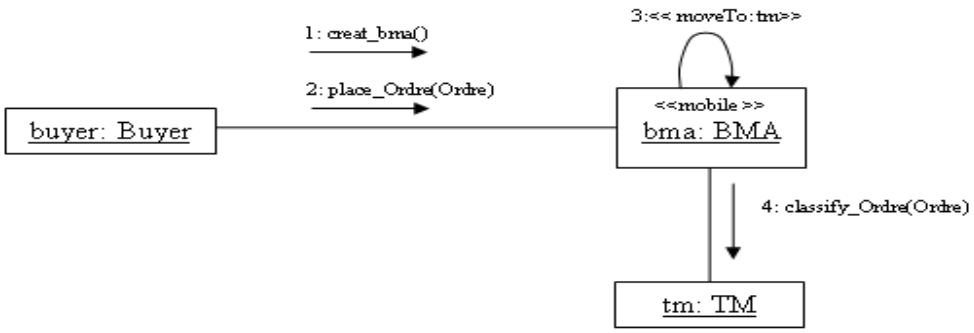


Figure 2.17. Diagramme d'objets (passation d'un ordre d'achat)

Dans la figure 2.17 nous décrivons le diagramme de séquence de passation d'ordre. Pour passer un ordre d'achat:
- Un agent *buyer* crée un agent mobile vendeur *bma* à travers le message (**1: creat_bma()**);
- Par la suite, le *buyer* invoque le *bma* pour passer l'ordre via le message (**2: place_ordre(Ordre)**);

- Pour cela, le *bma* doit se déplacer vers la plate-forme de base de l'agent *tm*, cette action est accomplie par le message (**3: <<moveTo:tm>>**);
- Enfin, l'agent *bma* interagit localement avec l'agent *tm* pour terminer sa tache suite au message (**4:classify_Ordre(Ordre)**).

2.3.2.5. Diagramme d'état transition

L'exemple suivant montre le diagramme d'état transition des deux agents mobiles *BMA* et *PMA* de notre système boursier.

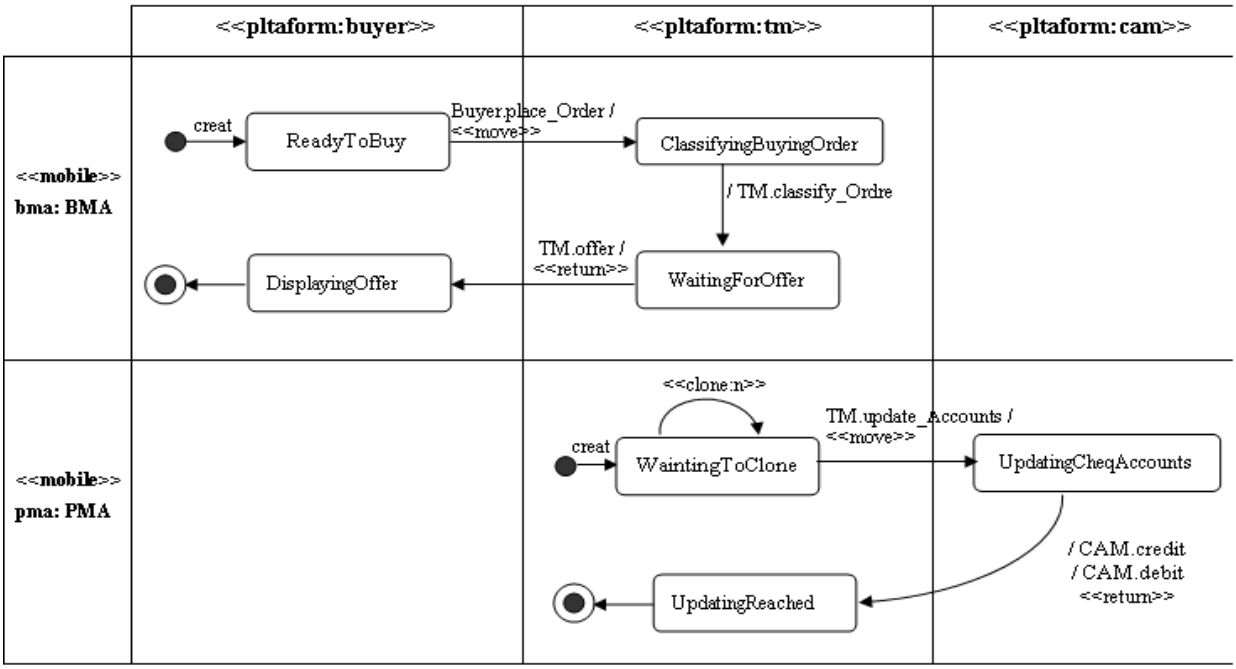


Figure 2.18. Diagramme d'état transition des agents mobiles BMA et PMA

Dans la figure 2.18, la première ligne orthogonale représente les états de l'agent mobile *BMA* qui sont connectés par des transitions. Le premier état qui peut être pris par cet agent mobile est *ReadyToBuy* (c.-à-d. prêt à acheter), après avoir reçu un message (*Buyer.place_ordre*) depuis l'objet *Buyer*, cet agent se déplace vers la plate-forme de base de l'agent *TM* afin de passer à l'état *ClassifyingBuyingOrdre* où il tente de classer son ordre d'achat. Pour cela, il envoie un message *TM.classify_Ordre* à l'agent gestionnaire de transactions *TM* et il change ensuite son état à *WaitingForOffer* dans lequel il se met en attente jusqu'à ce qu'il reçoive un message TM.*offer* de l'agent *TM*, une *offre* porte le prix calculé de l'action demandée et la quantité proposée, qui déclenche une action provocant *BMA* à se déplacer pour changer son état à *DisplayingOffer* sur la plateforme de base de son acheteur (propriétaire), afin de lui présenter cette offre.

2.3.2.6. Diagramme d'activité

Dans la figure 2.19 nous représentons le diagramme d'activité de l'achat d'une action. Dans l'activité *''Place Order''* un acheteur tente de passer un ordre. Pour cela, un agent mobile *BMA* doit se déplacer vers la plate-forme de l'agent *TM* en appliquant l'action ''*Go*''. Au niveau de l'activité *''Receive Orders''* l'agent *TM* reçoit les ordres d'achat et de vente depuis les agents mobiles *BMA* et *SMA* arrivant depuis leurs platesformes de base. L'activité ''*Receive Orders*'' a deux entrées où les deux nœuds paramètres mobiles *BMA* et *SMA* sont placés.

L'ordre d'achat passé sera traité au niveau de l'activité *''Manage Order''* : si cet ordre est accepté [**Order accepted**] l'agent mobile doit retourner vers sa plate-forme de base en appliquant l'action *''Go''*. En conséquence, dans l'activité *''Make Decision''* l'acheteur peut affirmer [**OK**] ou annuler [**Cancel**] son ordre. Si l'ordre est affirmé, l'activité *''Update Accounts*'' est invoquée, dans cette activité tous les comptes (comptes chèque et comptes d'action) des actionnaires concernés par cette transaction sont mis à jour; le symbole petit râteau (rake-style) attaché à l'intérieur ressemble à une hiérarchie miniature indiquant que cette invocation lance plusieurs activités.

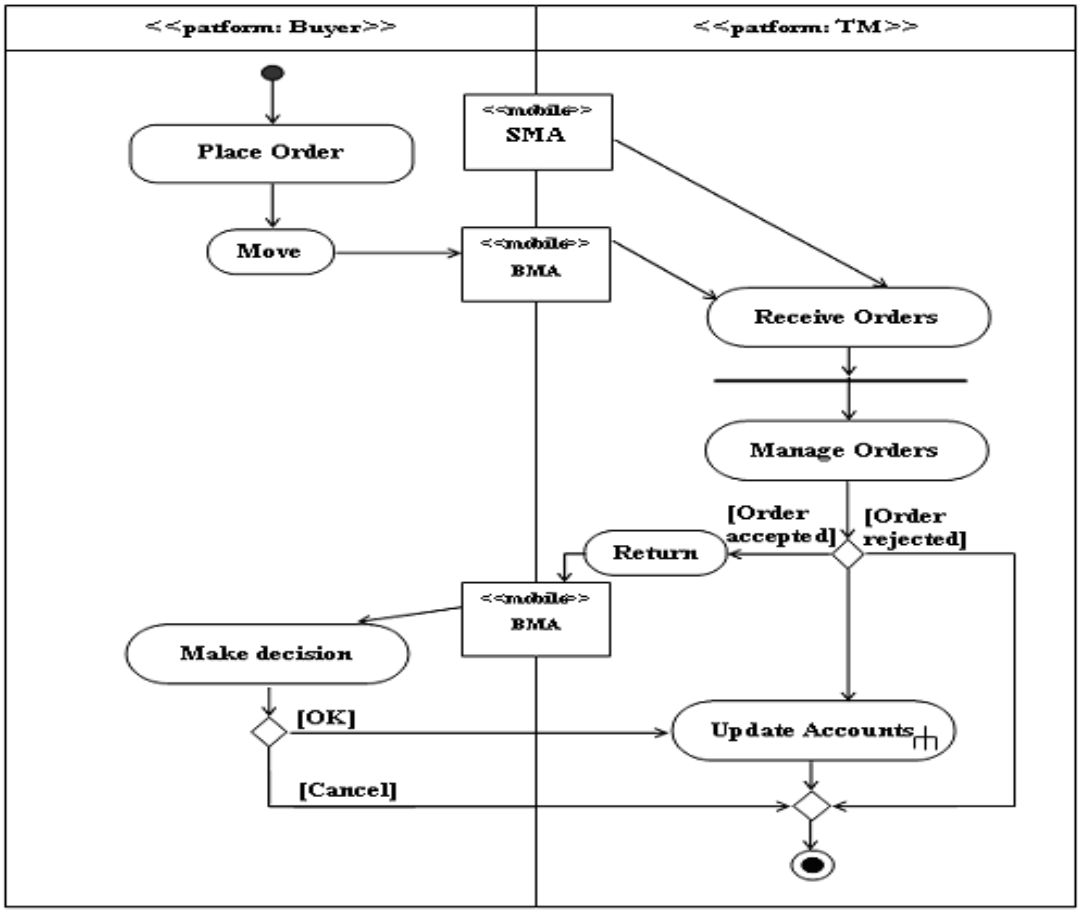


Figure 2.19. Diagramme d'activité d'achat d'une action

2.3.2.7. Diagramme de déploiement

Dans notre diagramme de déploiement étendu, comme le montre la figure 2.20, les nœuds ***BuyerServer*** peuvent envoyer les agents mobiles ***BMA***, et le nœud ***SelerServer*** peuvent envoyer les agents mobiles ***SMA***. Alors que les nœuds ***BancServer*** peuvent recevoir les agents mobiles ***PMA.*** Enfin le nœud ***PurseServer*** peut envoyer les agents mobiles ***PMA*** et recevoir les agents mobiles ***BMA*** et ***SMA.***

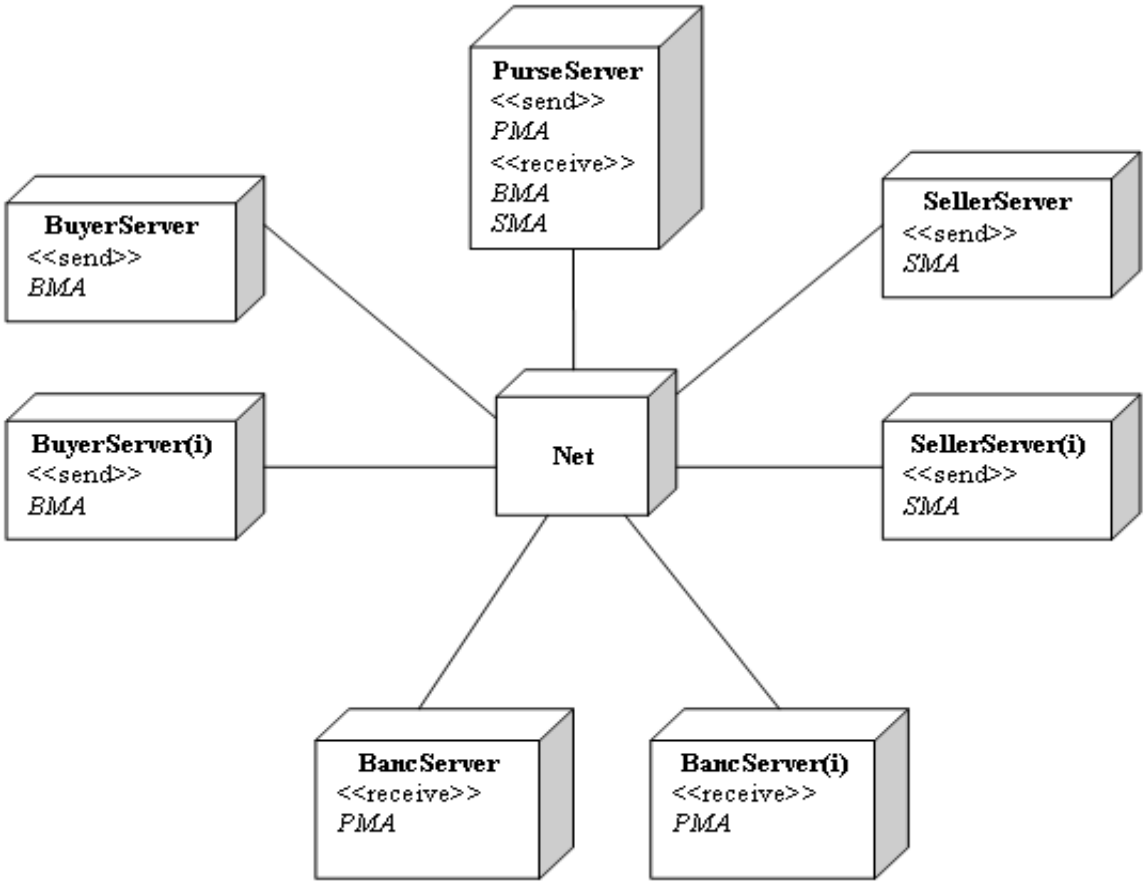


Figure 2.20. Diagramme de déploiement du système boursier

2.3.3. Perspectives

UML2.0 offre de nouveaux éléments de modélisation flexibles nous permettant de bien capturer les concepts de mobilité. Dans cet optique, nous avons proposé dans ce mémoire une approche qui consiste en l'extension de cette version (UML2.0). Notre objectif était d'introduire de nouveaux éléments de modélisation afin de capturer les trois principaux concepts de mobilité: la location, la migration, et le clonage. Afin de montrer l'applicabilité

de notre approche, nous avons modélisé un système de bourse électronique comme une étude de cas.

Le comportement d'un agent mobile est défini à partir de ses plans et ses actions. A tout moment, l'agent mobile choisit un ensemble de plans à exécuter pour atteindre son but. Notre futur travail consiste en l'introduction de nouveaux éléments de modélisation, afin de pouvoir modéliser les plans et les actions d'un agent mobile.

2.4. Conclusion

Dans ce chapitre, nous avons présenté une introduction au langage de modélisation UML, en débutant par la présention de la modélisation en général. Nous avons enchaîné ensuite sur quelques aspects d'UML et une brève présentation de ses extensions. Nous avons présenté également notre première contribution d'extension d'UML 2.0 par mobilité. Cette contribution à fait l'objet de quelques publications notamment un journal (IJCSNS novembre 2009) .

En conséquence, UML possède plusieurs facettes. C'est une norme, un langage de modélisation objet, un support de communication et 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 dûe à sa 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 équivalents où la vérification est abordable.

Dans le chapitre suivant, nous introduisons les réseaux de pétri afin de définir le modèle cible de la transformation du modèle source UML. L'approche de transformation de modèle ou, plus précisément la transformation des graphes sera abordée en chapitre quatre.

Chapitre 3

Les réseaux de Petri NestedNets

3.1. Introduction

Nous avons présenté dans le chapitre précédant, une contribution qui consiste en l'extension des diagrammes d'UML 2.0 pour supporter la mobilité. L'intérêt d'une telle contribution est de renforcer la robustesse des applications dans les systèmes distribués et ce, en les modélisant avec des outils de modélisation standards tel que UML. Le revers de la médaille est que UML est un langage de modélisation semi-formel, cet aspect réduit énormément la possibilité de l'analyse et de la vérification des systèmes modélisés.

La vérification et la validation des systèmes informatiques complexes constituent un enjeu important. Un système ouvert évolue continuellement, sa validité est toujours remise en cause. C'est en particulier le cas pour les systèmes critiques, dont les pannes peuvent avoir des conséquences irréversibles et dramatiques sur leurs environnements tel que les systèmes mobiles. Dans ce cadre, nous introduisons dans ce chapitre les réseaux de Petri de haut niveaux NestedNets, pour la modélisation de la mobilité. Ces réseaux formels permettent d'analyser et de vérifier les systèmes mobiles et palier ainsi aux insuffisances de l'approche semi-formelle d'UML.

Les réseaux de Petri offrent un outil formel et une bonne représentation graphique qui permettent de modéliser et d'analyser les systèmes discrets, particulièrement les systèmes concurrents et parallèles.
La facette graphique des réseaux de Petri, nous aide à comprendre aisément le système modélisé. Par ailleurs, leur puissance d'expression mathématique permet de simuler des activités dynamiques et concurrentes. L'intérêt majeur de ces réseaux réside dans leurs possibilité d'analyser les systèmes modélisés, grâce aux modèles de graphes et aux équations algébriques.

3.2. concepts de bases et définitions des réseaux de Petri (RdPs)

3.2.1. Historique

Les réseaux de Petri [Tran05], ont été introduits par Carl Adam Petri dans sa thèse "Communication avec des Automates" en Allemagne, en 1962. Ce travail a été développé par la suite par Anatol W. Holt, F. Commoner, M. Hack et leurs collègues dans le groupe de recherche de Massachussetts Institute Of Technology (MIT), au début des années 70s. En 1975 MIT organise la première conférence sur les réseaux de Petri et les méthodes relationnels. Plus tard, En 1981 le premier ouvrage sur les réseaux de Petri a été publié par J. Peterson.

3.2.2. Définitions

- ***Réseaux de Petri simples*** : On appelle Réseau de Petri simple places-transitions 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) représente l'ensemble de places d'entrée de la transition Ti (transitions d'entrée) ;
- O(Ti) représente l'ensemble de places qui sont en sortie de la transition Ti (sotie des places).

- ***Réseaux de Petri marqués*** : On appelle Réseaux de Petri marqués, le couple R=<Q;M0> , où M0 est le marquage initial du réseau et tel que la fonction de marquage M est définie de **P** dans l'ensemble des entiers naturels

3.2.3. Représentation graphique de RdPs

Le réseau de Petri est représenté par deux type de sommets alternés, les places Pi et les transitions Ti. Ces sommets sont reliées par des arcs orientés. Tout arc doit relier une place à une transition ou bien une transition à une place (cf. la figure 3.1)

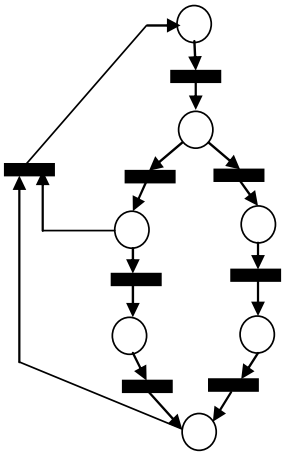


Figure 3.1. Un réseau de Petri comportant
7 places, 6 transitions et 15 arcs orientés

3.2.4. Le Marquage d'un réseau de Petri

Le marquage d'un réseau de Petri permet de définir l'état d'un système modélisé par ce réseau. Le marquage consiste à placer initialement un nombre **mi** entier (positif ou nul) de jetons dans chaque place **Pi** du réseau. Le marquage du réseau sera défini par un vecteur **M = {mi}** (cf. la figure 3.2).

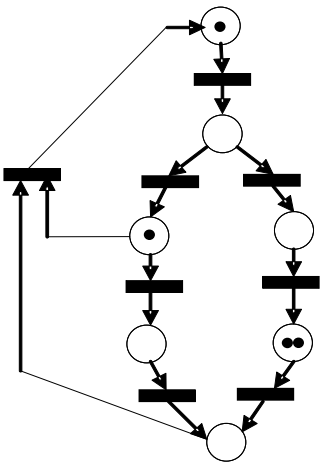


Figure 3.2. Un réseau de Petri marqué avec un vecteur
de marquage M : **M = (1,0,1,0,0,2,0)**.

3.2.5. Evolution d'états d'un réseau de Petri

L'évolution d'état du réseau de Petri correspond à une évolution de marquage. Les jetons qui indiquent l'état du réseau à un instant donné, peuvent passer d'une place à une autre par un **franchissement** ou par un **tir** d'une transition. Dans le cas des réseaux dits à arcs simples ou de poids égal à 1 (cf. la figure 3.3), Le franchissement d'une transition consiste à retirer un jeton dans chacune des places d'entrée de la transition et à en ajouter un dans chacune de sorties de places de celle-ci.

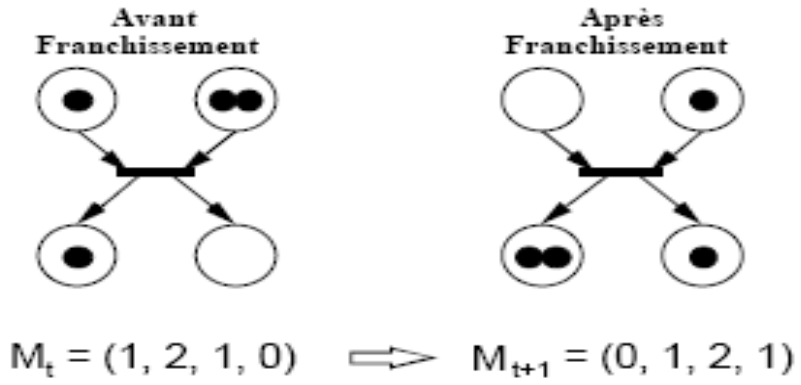


Figure 3.3. Evolution d'états d'un réseau de Petri **[Claud.K01]**

En général l'évolution des états d'un réseaux de Petri marqués simple obéit aux règles suivantes :

- Une transition est **franchissable** ou **sensibilisée** ou encore **tirable** lorsque chacune des places d'entrées possède au moins le nombre de jetons correspondant au poids de l'arc le reliant à la transition.
- Le réseau ne peut évoluer que par franchissement d'une seule transition à la fois, transition sélectionnée parmi toutes celles validées au moment du choix.
- Le franchissement d'une transition est indivisible et de durée nulle.

Ces règles introduisent un certain indéterminisme dans l'évolution des réseaux de 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 [Claud01].

3.2.6. Sémantique du parallélisme et problème de conflits

Dans un réseaux de Petri, plusieurs transitions peuvent être franchissables à un moment donné. Cette aspect de parallélisme du tir des transitions pose un problème de choix pour l'état futur du réseau. En général, ce conflit est résolut par le choix d'une sémantique dite du parallélisme qui définit une stratégie de tir, tel que le tir d'une seule transition à la fois.

3.4. Méthodes d'analyse pour les réseaux de Petri

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

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

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

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

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

3.5.Extensions des réseaux de Petri

La modélisation d'un système réel impose beaucoup de contraintes qui peuvent mener à des réseaux de Petri de taille importantes. En plus, les RdPs usuels ne permettent pas d'exprimer certaines propriétés, telle que la mobilité, rendant ainsi leur analyse assez difficile. Ceci a motivé plusieurs extensions de réseaux de Petri tels que : les réseaux de Petri colorés [Jensen 97, Jensen98], Les ECATNets [Bettaz92,

Bettaz93], etc. Dans la section suivante, nous présentons un aperçu de quelques extensions des Rdps.

3.5.1. Les réseaux de Petri colorés

Afin d'augmenter l'expressivité d'un RdP, les jetons placés dans les états du réseau seront colorés, ce qui permettra de les distinguer. Ce procédé de marquage permet de distinguer les jetons d'un même état. L'exemple du producteur consommateur sera agréablement modélisé par un réseau de Petri coloré. Chaque processus sera représenté par un jeton de couleur différente dans un même état du réseau, ce qui facilitera le tir de la transition.

3.5.2. Les réseaux de Petri temporisés

Dans ce modèle de réseau de Petri, la durée d'une activité est explicitement intégrée. La temporisation peut concerner les places(réseaux de Petri P-temporisés) ou bien les transitions (réseaux de Petri T-temporisés) selon les événements modélisés.

Beaucoup d'autres extensions des réseaux de Petri sont développées. Dans cette thèse, nous nous intéressons aux systèmes à base d'agents mobiles. Nous focaliserons dans les sections suivantes sur les réseaux de Petri étendus pour modéliser la mobilité.

3.6. Modélisation des systèmes mobiles par les RdPs

La modélisation des systèmes mobiles par les RdPs ordinaires ou étendus tel que les RdPs colorés s'avere insuffisante à cause des particularités des applications mobiles(migration clonage, etc.). Ainsi, l'introduction de nouvelles extensions aux RdPs s'avere nécessaires pour modéliser les applications mobiles.

3.6.1. Le paradigme (*nets-within-nets*)

Le paradigme ***nets-within-nets*** **[Koh07]** est un formalisme qui fournit une technique innovante de modélisation en donnant aux jetons mêmes du réseau la structure d'un réseau de Petri **(Figure 3.13)**. Contrairement aux autres types de réseaux de Petri où les jetons sont passifs, les jetons dans les ***nests-within-nets*** sont actifs. La notion de jeton dans ce paradigme permet de modéliser les systèmes récursifs et hiérarchiques de façon simple. Elle permet aussi la modélisation des systèmes adaptatifs et mobiles.

Dans l'exemple de la Figure 3.4 du réseau de Petri nets-within-nets, nous considérons la situation où nous avons une hiérarchie à deux niveaux. Un niveau inférieur qui s'appelle ***System Nets*** (le RdP à ce niveau est formé de places contenant des jetons qui sont aussi des RdPs) et un niveau supérieur qui s'appelle ***object nets*** (les places dans le RdP à ce niveau contiennent des jetons simples). La place **P1** de

System Nets de la figure 3.4 contient un ***Object Nets*** (Le RdP dont les places sont **b1** et **b2**).

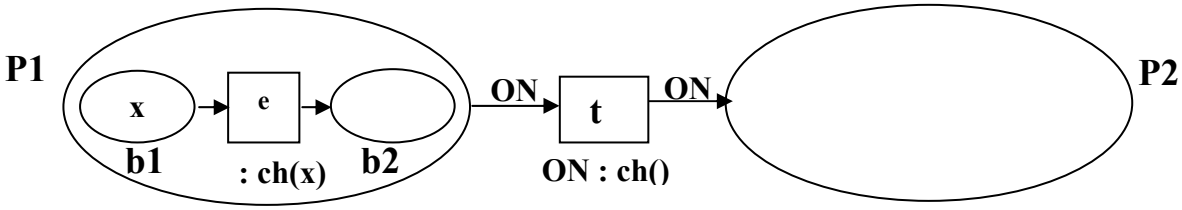


Figure 3.4. Un exemple d'un RdP Nets-within-Nets **[Koh07]**

Le franchissement d'une transition dans le niveau inférieur peut être synchronisé par le franchissement d'une transition dans le niveau supérieur. Dans l'exemple de la figure 3.4, la transition ***t*** n'est franchissable que lorsque la transition ***e*** est franchissable. Cette synchronisation se fait par des fonctions qui s'appellent ***down-link*** et ***up-link*** :

- Le ***down-link*** est la fonction qui mène la transition du niveau inférieur (System Nets).

Le ***up-link*** est la fonction qui mène la transition du niveau supérieur (Object Nets).

Le franchissement de la transition ***t*** donne lieu à la situation de la Figure 3.5.

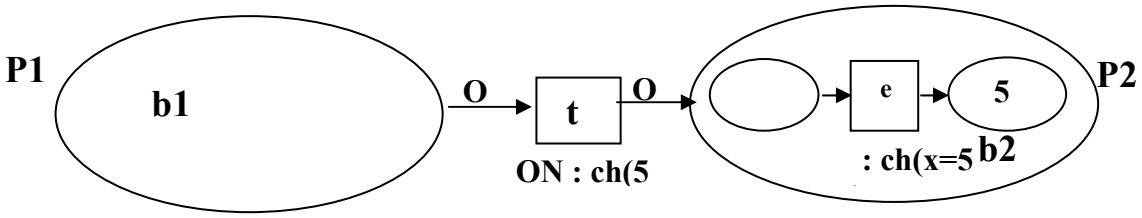


Figure 3.5. Un exemple d'un RdP Nets-within-Nets après franchissement.

L'interaction entre les ***Object Nets*** et les ***Système Nets*** induit quatre possibilités pour contrôler le type de déplacement et par conséquent la mobilité des ***Object Nets*** :

- L'***Object Net*** déclenche son propre mouvement dans le ***Système Nets.*** dans cette situation le ***Système Nets*** n'aura aucune influence sur la mobilité des ***Object Nets***.

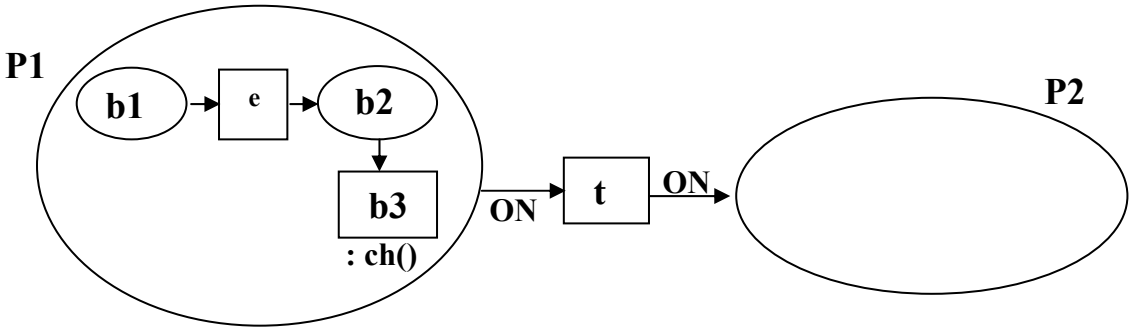


Figure 3.6. Mobilité subjectif

- Le ***Système Nets*** force l'***Object Net*** à se déplacer, dans cette situation l'***Object Net*** subit une transportation forcée.

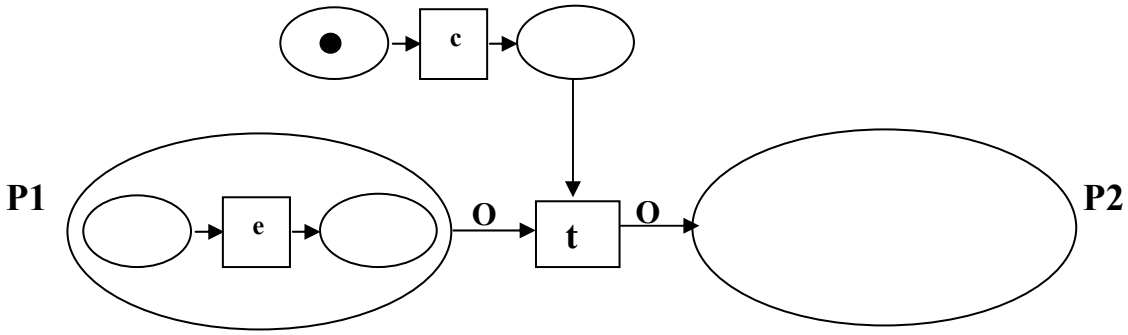


Figure 3.7. Mobilité Forcée

- L'***Object Net*** et le ***Système Net*** parviennent à un accord pour la mobilité de l'***Object Net***.

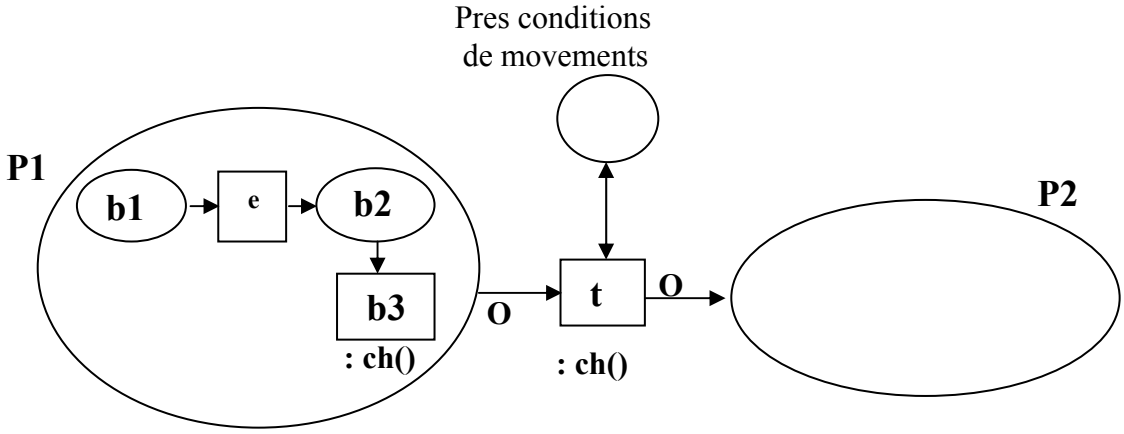


Figure 3.8. Mobilité par Accord

- L'***Object Net*** se déplace à l'interieur du ***System Net.*** Dans cette situation le mouvement de l'***Object net*** est non controlé (cf.figure)

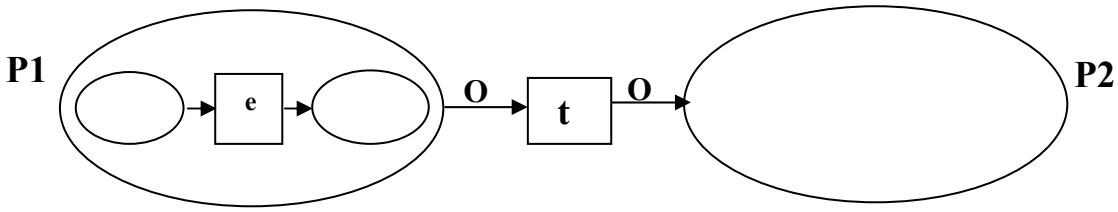


Figure 3.9. Mobilité spontanée

3.6.2. Le paradigme (Nested Nets)

Dans le paradigme ***Nested Nets*** chaque jeton peut être aussi un réseau de Petri. Ce paradigme est basé sur les notions introduites par le paradigme **nets-within-nets** en se basant sur l'étude d'une sémantique autre que la sémantique de référence, pour modéliser les systèmes adaptatifs et mobiles [Kee].

Ce paradigme est une extension des RdPs colorés dont les jetons peuvent changer de couleur sans le franchissement de transitions [Kee], (cf. la figure 3.15).

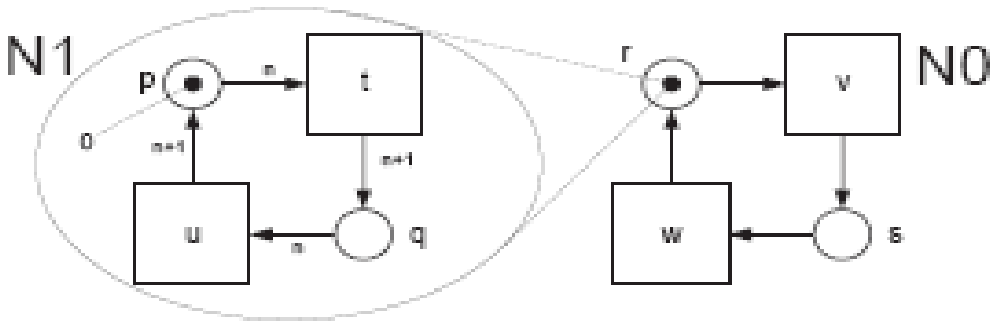


Figure 3.10. Nested Nets **[Kee]**

Le réseau peut supporter N niveaux d'abstraction. Pour chaque niveau, le franchissement d'une transition peut être synchronisé avec le franchissement d'une transition de niveau supérieur par adjacence, et ainsi de suit.

Le paradigme ***Nested Nets*** permet de représenter les concepts de la mobilité tels que : la localité, la migration, le clonage ainsi que le suivit du cycle de vie d'un agent mobile. Un Nested Net à deux niveaux d'abstraction permet par exemple de modéliser au niveau 0 les locations, la migration et le clonage des agents mobiles. Le niveau 1 peut etre utilisé pour modéliser le cycle de vie d'un agent mobile.

3.7. Conclusion

Dans se chapitre, nous avons abordé dans un cadre général la modélisation par les réseaux de Petri. Par la suite, nous avons étudié quelques types de RdPs avec leur variantes. Nous avons également introduit une brève présentation des extensions de RdPs pour la mobilité, notamment les Nested Net.

Dans le chapitre suivant, nous abordons une approche intégrée Mobile-UML/Réseaux de Petri pour l'Analyse des systèmes distribués à base d'agents mobiles. Cette transformation de modèles, basée sur les l'MDA, va permettre de traduire les diagrammes d'UML Mobile vers les Nested Net, afin de pouvoir effectuer des analyses et des vérifications au niveau de ces diagrammes.

Chapitre 4

La transformation de Modèles

4.1. Introduction

La modélisation au sens large est l'utilisation efficace d'une représentation simplifiée d'un aspect de la réalité pour un objectif donné. Loin de se réduire à l'expression d'une solution à un niveau d'abstraction plus élevé que le code [MDA05], la modélisation en informatique peut-être vue comme la séparation des différents besoins fonctionnels et préoccupations extra-fonctionnelles (telles que : la sécurité, la fiabilité, l'efficacité, la performance, la flexibilité, etc.).

Ce que propose l'approche de l'ingénierie des modèles (IDM ou MDE en anglais pour Model Driven Engineering) est la mécanisation du processus que les ingénieurs expérimentés suivent à la main [MDA05]. L'intérêt pour l'IDM a été fortement ressentit à la fin du 20^{ième} siècle, lorsque l'organisme de standardisation OMG (Object Modeling Group) a rendu publique son initiative MDA (Model Driven Architecture). Celle-ci peut être vue comme restriction de l'IDM à la gestion de l'aspect particulier de la 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 se réduit à un ensemble de transformations de modèles partiellement ordonnés. Chaque transformation prend des modèles en entrée et produit d'autres en sortie, jusqu'à l'obtention d'artéfacts exécutables.

Dans cette thèse, nous nous intéressons particulièrement aux concepts de transformation de modèles. Les diagrammes d'UML (modèle source), seront vérifiés par les réseaux de Petri NestedNets (modèle cible). La transformation du modèle source (UML) vers le modèle cible (NestedNets), sera réalisée par une transformation de graphe en utilisant l'outil AToM^3[ATOM3].

Nous commençons par une présentation des concepts de base pour la transformation de modèles (dans le cadre général). Par la suite, nous exposerons quelques types de ces transformations, ainsi que leur classification. Nous focaliserons par raffinement sur la transformation de graphes comme entant un type spécifique de la transformation de modèles.

4.2. Une approche pour l'architecture MDA

L'esprit de l'architecture MDA, découle d'une approche de développement des applications et des systèmes dirigé par les modèles. L'MDA vise à modéliser des applications et des systèmes indépendamment de l'implémentation cible (niveau matériel ou logiciel), ce qui permet la réutilisation des modèles développés. Les modèles ainsi créés (PIM :Platform Independant Model) seront transformés pour obtenir des modèles d'applications spécifiques à la pateforme cible (PSM : Platform Specific Model). Des outils de génération automatique du code, permettent par la suite de générer les programmes directement à partir des modèles.
Cette approche, permet en plus de faire évoluer aisément les applications et leurs architectures à partir des modèles. Par conséquent, Le MDA s'investit dans un grand atelier dont le défit relève techniquement de la manipulation des modèles. Dans ce cadre spécifique, la transformation de modèle basé sur les techniques de métamodélisation et l'ingénierie de modèles ouvre un champ d'investigation prometteur pour la recherche.

4.2.1. Principes de mise en œuvre

Le MDA est perçu comme un processus spécifique de type IDM qui repose sur les principes suivants :

• ***La maitrise de la complexité*** : le MDA s'illustre généralement selon deux principales préoccupations distinctes, l'une pour l'élaboration de modèles métier indépendants des plateformes de mise en œuvre PIM, l'autre pour l'élaboration de modèles spécifiques de la plateforme de mise en œuvre PSM [MDA05]. Cette séparation permet de réduire la complexité de développement des applications.

• ***L'abstraction et la capitalisation des modèles:*** les modèles doivent être indépendant des technologies de mise en œuvre, afin d'adapter la logique métier à différents contextes et permettre l'évolution des applications vers de nouvelles technologies, permettant ainsi une meilleure réutilisation de ces modèles.

• ***La Modélisation***: Elle est abordée selon une vision productive (générer le code final du logiciel développé pour une technologie de mise en œuvre donnée) par opposition à la traditionnelle vision contemplative (but de documentation, spécification et de communication).

• ***La Métamodélisation*** : ce processus permet de définir les entités nécessaires pour la modélisation des systèmes spécifiques [MDA05]. Un métamodèle permet de définir la structure des modèles en spécifiant leur syntaxe et leur sémantique.

• ***La transformation de modèles :*** les techniques de métamodélisation permettent d'effectuer les transformations de modèles, en considérant ces transformations comme modèles.

• ***Les standards OMG:*** l'apport de l'abstraction dû à l'introduction de la métamodélisation, permet de détecter certaines incohérences au niveau des dialectes présentés par les métamodèles. Dans ce cadre, L'OMG adopte MOF(Méta Object Facility) qui permet la définition des éléments essentiels, la syntaxe et la structure des métamodèles.

Par ailleurs, les différents outils MDA utilisent XMI (XML Metadata Interchange) comme format standard d'échange de MetatDonnées. Ce standard de l'OMG, permet de décrire par exemple une instance du MOF sous forme textuelle grâce au langage XML (eXtensible Markup Language).

D'autre part, l'UML(Unified Modeling Language) avec sa capacité de modélisation des systèmes indépendamment de toute plateforme, s'est imposé comme outil de modélisation des PIM et quelques PSM dans le MDA.

Pour ne citer que cela, d'autres standards de l'OMG trouvent leurs rôles dans MDA, tels que : OCL(Object Constraint Language) qui apporte plus de précisions aux modèles sources et aux définitions de langages, ainsi que CWM(Common Warehouse Métamodel) le langage de modélisation dédié à la modélisation des applications de *datawarehouse* (cf. la figure 4.1).

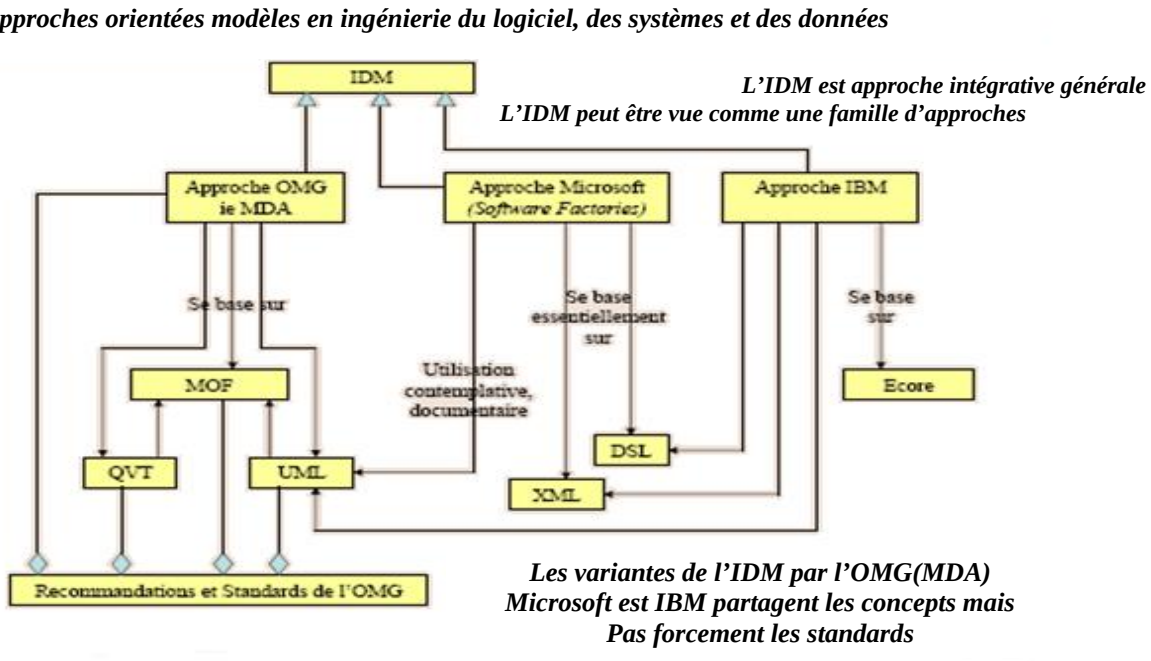


Figure 4.1. Les variantes de l'IDM **[Favre]**

4.2.2. Modèle d'architecture MDA à quatre niveaux

L'OMG a défini une architecture à quatre niveaux d'abstraction, comme carde général pour l'intégration des métamodèles, en se basant sur l'MOF comme le montre la figure 4.2. Dans cette architecture, les modèles de deux niveaux adjacents sont liés par une relation d'instanciation :

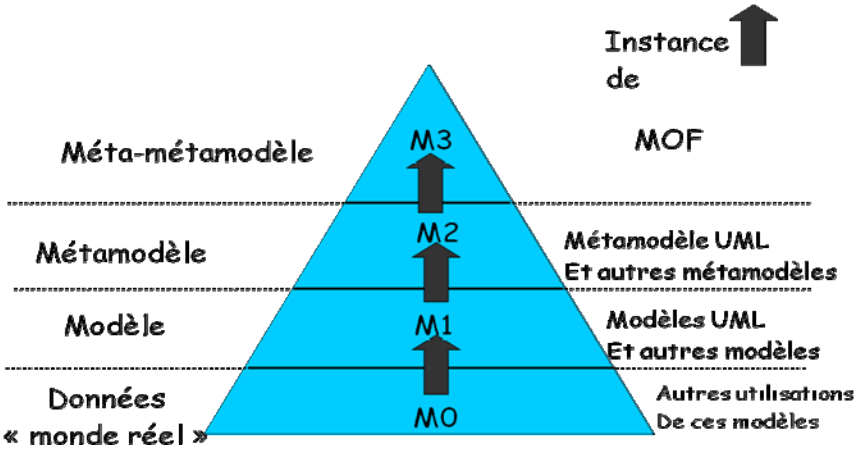


Figure 4.2. Les quatre niveaux d'abstraction pour MDA **[OMG04]**

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

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

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

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

4.2.3. Les modèles dans l'Architecture MDA

L'évolution continue des technologies et du coût élevé de l'adaptation des applications logicielles à ces technologies, incita l'OMG à proposer le MDA **[OMG04]** vers la fin de l'année 2000. Le but principal de cette approche est de séparer les parties métiers de leur mise en œuvre technologique et 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.

• ***Le modèle CIM :*** le CIM (*Computational Independent Model*) correspond au modèle des besoins au niveau métier. Il permet de recenser les différents besoins du clients indépendamment de toute implémentation.

• ***Le modèle PIM :*** le PIM (*Platform Independent Model*) correspond au modèle de spécification de la partie métier d'une application. Cette spécification doit être conforme à une analyse informatique cherchant à répondre aux besoins métiers indépendamment de la technologie de mise en œuvre.

• ***Le PSM* :** le PSM (*Platform Specific Model*) correspond au modèle de conception et d'implémentation d'une application par rapport à une plateforme spécifique.

4.2.4. La transformation de modèles dans l'approche MDA

4.2.4.1 Principe de transformation

La transformation de modèles se base essentiellement sur les relations entre les modèles. La transformation d'un modèle source en un modèle cible, nécessite la connaissance parfaite des relations existantes entre les deux modèles. Cette transformation doit mettre en œuvre des opérations permettant la création d'un modèle cible à partir des informations fournies par le modèle source.

En MDA la transformation de modèles est pilotée par les métamodèles et comprend deux étapes [MDA05] successives, qui consistent en :

• La spécification de règles de transformation exprimant la correspondance entre les concepts du métamodèle décrivant le modèle source et les concepts du métamodèle décrivant le modèle cible.

• Application des règles de transformation définies au modèle source pour générer le modèle cible.

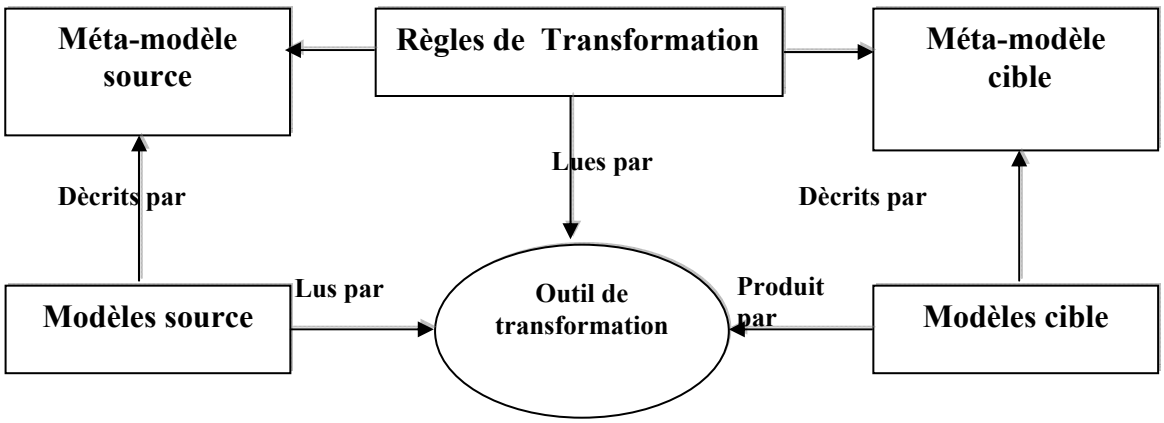


Figure 4.3. Processus de transformation de modèles pilotée par les métamodèles

4.2.4.2. Types de transformation

Le paradigme MDA repose sur la transformation de modèles. On distingue selon la littérature [MDA05] deux types de transformation de base :

• ***La transformation Horizontale*****:** dans ce type de transformation, le niveau d'abstraction en terme de modélisation est préservé(PIM à PIM, ou bien PSM à PSM). Cette transformation permet de faire des mises à jours au niveau des modèles transformés. L'exemple du passage de l'analyse à la conception, illustre la transformation PIM à PIM.

• ***La transformation verticale*****:** dans ce type de transformation, le niveau d'abstraction des deux modèles, source et cible est différent (PIM vers PSM, ou bien PSM au PIM). Le raffinement PIM vers PSM, permet par exemple d'intégrer les éléments spécifiques à une plateforme technique, tels que J2EE, Corba, .NET, etc. Par contre, l'abstraction PSM au PIM, s'avère très utile pour la reconstruction des systèmes en migration.

Le développement de systèmes en MDA, nécessite le plus souvent une série de transformations de modèles. Ceci peut induire à une mixture de transformations horizontales et verticales, d'où la possibilité d'intégrer un autre type de transformation hybride, utilisant les deux types définis précédemment.

4.2.4.3. Classification des approches de transformation de modèles

La classification des approches de transformation de modèles se base selon *NPrakash* [Prakash06] sur plusieurs points de vue, Chacun d'entre eux permet une classification particulière. D'après *CZarnecki* [Czar03] la classification des approches de transformation de modèles se base sur les techniques de transformation utilisées dans ses approches et les facettes qui les caractérisent, telles que : la technique des patrons et celle des langages de programmation. D'autre part, *NPrakash* [Prakash06] propose une classification selon un point de vue multidimensionnel, il s'intéresse particulièrement au domaine d'application des modèles et à leur généricité (capacité de transformer n'importe quel modèle d'entrée à n'importe quel modèle de sortie).

.

Dans cette section, nous allons présenter une classification des approches de transformation de modèles, en s'inspirant des travaux de *CZarnneki* [Czar] où l'on trouve une décomposition de la transformation de modèles en deux catégories : les transformations de type *modèle vers code* et les transformations de type *modèle à modèle.*

a- La transformation de type modèle vers code

Cette transformation propose deux approches transformationnelles. Une première approche basée sur le principe du *visiteur (Visitor-based approach)* et la seconde basée sur le principe des *patrons (Template-based approach*).

• Les approches basées sur le principe du ***visiteur*** transforment le modèle en entrée vers un code écrit dans un langage de programmation en sortie. Les visiteurs qui seront rajoutés au modèle d'entrée, réduisent la différence de sémantique entre le modèle et le langage cible. Le code cible est obtenu en parcourant le modèle enrichi par les visiteurs pour créer un flux de texte en sortie.

• Les approches basées sur le principe des ***patrons*** reposent sur l'utilisation des fragments de méta-code du code cible pour l'accès aux informations du modèle source. Ces approches sont actuellement très utilisées dans les outils MDA, tel que : *AndroMDA* (un générateur de code qui utilise la technologie ouverte *Velocity* pour l'écriture des patrons).

b- La transformation de type modèle à modèle

➢**Modélisation de la transformation**

La transformation de type modèle à modèle ne cesse de se développer dans le grand chantier MDA. Les différences, de la sémantique et de la syntaxe entre les deux modèles, source et cible tels que : les PMIs et les PSMs, imposent une démarche rigoureuse pour une telle transformation. La modélisation de transformation de modèles apparaît alors comme solution à ce problème. Modéliser une transformation relève de la métamodélisation qui s'impose actuellement comme technique prometteuse, dans le cadre de la transformation de modèles.

➢**Structure de la transformation**

Une transformation de modèle est un modèle en soit même. Elle se définit par un ensemble d'éléments à savoir : des règles de transformation ainsi que leurs organisation et ordonnancement, une traçabilité et une orientation. La combinaison de ses éléments permet de décrire la transformation. Toutefois, ces règles de transformation doivent être d'abord spécifier afin de pouvoir exprimer les correspondances entres les concepts des Métamodèles source et cible.

• ***Règles de transformation :*** une règle de transformation est une description de la manière dont une ou plusieurs constructions dans le langage du modèle source peuvent être transformées en une ou plusieurs constructions dans le langage du modèle cible [MDA05]. En plus, toute règle de transformation possède une logique de forme déclarative ou impérative pour exprimer des contraintes ou des calculs sur les modèles sources et cibles de la transformation.
Techniquement, une règle de transformation se compose de deux parties: une partie gauche appelée ***LHS*** *(Left Hand Side*) qui accède au modèle source, et une partie droite appelée ***RHS*** *(Right Hand Side*) qui accède au modèle cible. l'exécution d'une règle de transformation orientée de gauche à droite, permet de remplacer les constructions du modèle souce conformes au LHS de la règle avec les RHS de la même règle pour la construction du modèle cible. Cette technique nécessite une organisation et un ordonnancement des règles ainsi que leurs orientation. Le mécanisme de traçabilité permet en plus de renforcer la technique par archivage des corrélations qui existent entre les éléments des modèles de la transformation.

• ***Spécification des règles de transformation*** **:** les règles de transformation expriment la correspondance entre les concepts du métamodèle source et les concepts du métamodèle cible. Le rôle de la spécification est la description de telles relations indépendamment de toute exécution. MDA prévoit l'automatisation complète de cette phase.
Actuellement, il n'existe pas de standard permettant d'exprimer les règles de transformation. Les travaux actuels de l'OMG sur le standard MOF/QVT (Query Views Transformation) semble être prometteurs. Dans notre travail, nous avons opté pour l'outil ATOM^3 [ATOM3].

• ***Approches pour la définition de la transformation*****:** la transformation Modèle à Modèle se base sur une structure variée. Par conséquent, Plusieurs approches tentent de définir ce type de transformation. Dans la littérature, on distingue généralement cinq approches :
- les approches par manipulation directe,
- les approches relationnelles.
- les approches basées sur la transformation de graphes,
- les approches basées sur la structure,
- les approches hybrides.

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

4.3. Transformation de graphes

La modélisation des systèmes se heurte à la difficulté de la description de leurs structures complexes. Les modèles et métamodèles (par exemple UML) possèdent le plus souvent une représentation sous forme de graphe. Par ailleurs, Les graphes offrent un support agréable et efficace qui permet la modélisation de systèmes. Par conséquent, les techniques de transformation et de réécriture des graphes peuvent être appliquées à la transformation de modèles.

Avant d'aborder les techniques de transformation de graphes et les outils rendant de telles techniques applicables, il est nécessaire de rappeler quelques concepts de bases relatifs à la théorie de graphes.

4.3.1. Concept de graphe

Un graphe se définit généralement par un ensemble de sommets reliés par des arêtes. Principalement, on distingue deux catégories de graphes : les graphes orientés et les graphes non orientés (cf. les figures 4.4 et 4.5).
Dans la suite de cette section nous allons présenter quelques définitions relatives aux propriétés et caractéristiques des graphes :

• **Le Graphe** : on appelle graphe G le couple (S,A) constitué d'un ensemble S non vide de sommets et d'un ensemble A non vide d'arêtes. Chaque élément de A relie deux éléments de S.

• **Graphe orienté** : un graphe est dit orienté si ses arêtes possèdent une orientation (sommet de départ est sommet d'arrivé) , dans le cas contraire, le graphe est dit non orienté.

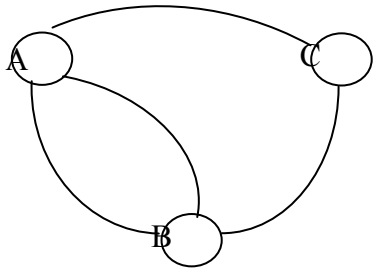


Figure 4.4. Graphe non orienté

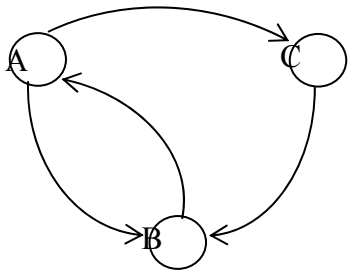


Figure 4.5. Graphe orienté simple

• **Graphe étiqueté** : un graphe utilisé pour la représentation d'un modèle doit être capable de prendre en charge les différentes relations entre les éléments du modèle représenté. Les graphe orientés ne suffisent pas pour exprimer les relations entre les éléments du modèle. Un étiquetage des arcs dans les graphes permet de spécialiser les relations entre les différents sommets dans un graphe.
Un graphe étiqueté est donc un graphe orienté, dont les arcs possèdent des étiquettes.

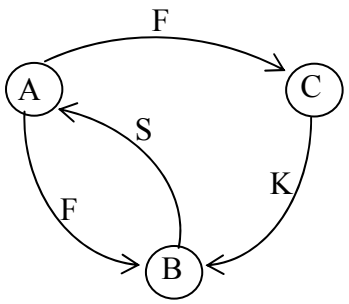


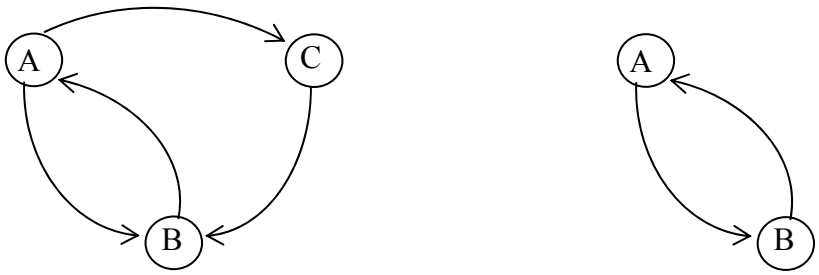
Figure 4.6. Graphe orienté étiqueté

• **Le Degré d'un graphe:** le degré d'un sommet dans un graphe est le nombre d'arêtes dans ce sommet. Si le graphe est orienté le degré de sommet se définit en degré entrant et degré sortant, relatifs respectivement au nombre d'arcs entrant et sortants dans ce sommet.
Le degré du sommet A de la figure 4.4 est de 3, les degrés entrant et sortant du sommet A de la figure 4.5 sont respectivement 1 et 2.

• **L'ordre d'un graphe** : l'ordre d'un graphe ***G*** se définit par le nombre de sommets présents dans ce graphe .
L'ordre du graphe de la figure 4.4 est de 3.

• **Le Chemin dans un graphe:** le chemin dans un graphe se définit par la succession des arcs parcourus dans un sens définit dans le graphe, il possède les propriétés et les caractéristiques suivantes :
- la longueur du chemin est égale au nombre d'arcs parcourus ;
- le chenin est dit une ***chaîne*** si l'on ne tient pas compte de la direction des arcs ;
- le chemin est dit ***circuit*** s'il revient à son point de départ ;
- la ***distance*** entre deux sommets dans un graphe se définit comme étant la longueur du plus court chemin entre ces deux sommets, si cette longueur est de 1 les sommets sont dits adjacents.
- le ***diamètre*** d'un graphe est la plus grande distance séparant deux sommets dans ce graphe.

• **Le sous-graphe** : un sous graphe ***G'***(***S',A'***) d'un graphe ***G***(***S,A***) est un graphe composé d'un sous ensemble de sommet ***S'*** $\in$ ***S*** et d'un sous ensemble d'arêtes ***A'*** $\in$ ***A'*** tel que ***A'*** représente les arêtes reliant les sommets ***S'*** dans le graphe ***G*** (cf. la figure 4.7).



Graphe – **G** Graphe – **G'**

Figure 4.7. G' est un sous graphe de **G**

4.3.2. Transformation de graphes

La transformation de graphes est un système de transformation de modèles[MDA05], elle se réalise grâce aux règles de transformation [Guerra06]. Les règles s'appliquent à un graphe source, en se basant sur le principe de ***pattern matching***, pour produire un graphe cible. Le principe de base est d'appliquer les règles de transformation qui remplacent des sous graphes dans le graphe source par des sous graphes dans le graphe cible et ce, en utilisant la production définie dans les règles de transformation. Ces fragments de sous graphes peuvent être exprimés dans des syntaxes concrètes ou abstraites respectives des modèles sources et cibles. **ATOM**, **UMLX** et **VIATRA** sont des exemples d'outils qui supportent la transformation de modèles basés sur la transformation de graphes.

Par analogie au grammaires de Chomsky qui s'appliquent aux textes, la transformation de graphes se réalise grâce aux règles de transformation organisées dans des grammaires de graphes. L'idée de base est de pouvoir dériver des modèles cibles à partir des modèles sources.

4.3.2.1. Grammaires de graphes

Les grammaires de graphes, d'un certain point de vue, sont une généralisation des grammaires de Chomsky appliquées aux graphes. La transformation de graphe est spécifiée sous forme d'un modèle de grammaires de graphes. Ces grammaires sont composées de règles dont chacune est constituée de deux parties : une partie gauche appelée **LHS** (Left Hand Side) qui correspond à un graphe, et une partie droite appelée **RHS** (Right Hand Side) qui correspond aussi à un graphe.

4.3.2.2. Principe de Transformation

Le processus de transformation de graphe distingue deux types de graphes: les graphes non terminaux, qui sont les résultats intermédiaires sur lesquels les règles de la grammaire sont appliquées, et les graphes terminaux qui sont dans le langage engendré par la grammaire et sur lesquels on ne peut plus appliquer de règles.

• ***Anatomie d'une règle de transformation de graphe***

Une règle de transformation de graphe est constituée d'un ensemble d'attributs lui permettant de substituer les graphes du modèle source par les graphes du modèle cible:

- **L :** graphe du coté gauche sur lequel va s'appliquer la règle.
- **R :** graphe du coté droit que va produire la règle.
- **K :** un sous graphe de **L.**
- **F :** un ensemble de fonction permettant à la règle de réaliser la substitution.

• ***Application d'une règle de transformation de graphe***

La transformation d'un graphe source '**G**' vers un graphe cible ' **G'** ', consiste en l'application d'une ou de plusieurs fois des règles de la grammaire de transformation au graphe **G**. L'application d'une règle de transformation s'effectue généralement selon le principe suivant :

- Choisir une occurrence 'ℓ' du graphe du coté gauche **L**, selon la règle à appliquer.
- Vérifier les conditions d'application de la règle.
- Réaliser la substitution des sous graphes 'ℓ' de **L** par des sous graphe 'ℓ**'** ' dans G'.

L'application d'une règle de transformation à un graphe **G** pour produire un graphe **G'**, est appelée une dérivation directe depuis **G** vers **G'** à travers la règle de transformation.

L'ordre de déclenchement des règles d'une grammaire reste arbitraire. Toutefois, il est possible d'exprimer un ordre d'exécution des règles en leur attribuant par exemple des priorités.

4.3.3. Outils de transformation de graphes

Plusieurs outils de transformation de graphes existent actuellement, parmi lesquels : AGG [AGG], AToM^3 [ATOM3], VIATRA [Viatra], etc. Dans le cadre de cette thèse, nous avons opté pour ATOM^3 à cause de sa simplicité et sa disponibilité.

Présentation de l'outil AToM^3

AToM^3 **[ATOM3]** « **A *To*ol for *M*ulti-formalism and *M*eta-*M*odeling** » est un outil visuel pour la transformation de modèles, écrit en **Python** [Python07] et s'exécute sur de différentes plateformes (Windows, Linux, etc.). Il utilise et implémente un ensemble de concepts tel que : la modélisation suivant un multi formalisme, la métamodélisation et les grammaires de graphes. Il est utile pour la modélisation, la métamodélisation et la transformation de modèles à l'aide des grammaires de graphes. Par ailleurs, il permet par extension de manipuler la simulation ainsi que la génération du code à partir des modèles élaborés. Les modèles dans AToM^3 ont une représentation graphique et une construction basée sur des règles issues d'une spécification d'un formalisme déterminé.

Dans AToM^3, la description graphique est attribuée aux modèles ainsi qu'au formalismes. AToM^3 permet de générer des outils de manipulation graphique des modèles décrits selon des formalisme spécifiés dans des méta-spécification. Par la suite, la transformation de modèles s'effectue en utilisant des grammaires de graphes (cf. la figure 4.8).

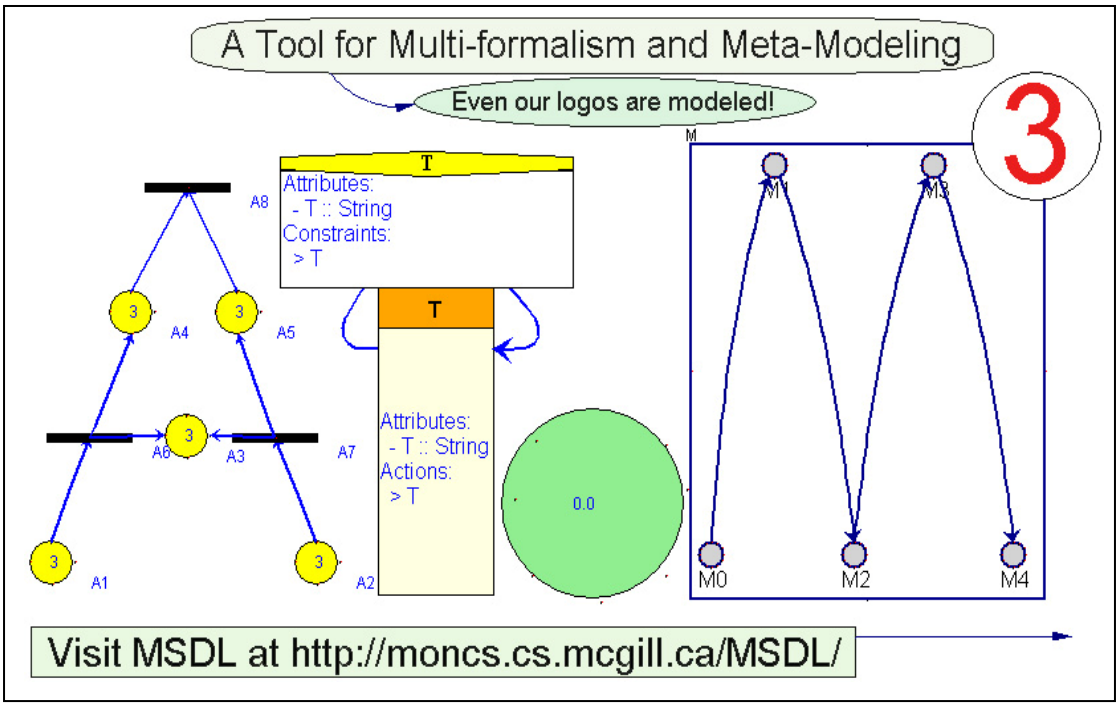


Figure 4.8. Présentation AToM^3

4.5. Conclusion

Dans ce chapitre, nous avons présenté la transformation de modèles, l'une des techniques prometteuses dans l'approche MDA. Cette approche permet l'automatisation des processus de modélisation, depuis les phases de développement jusqu'à celles de tests, en passant par la génération de code.

Dans ce contexte, nous avons présenté quelques approches de modélisation issues de la littérature, en commençant par une introduction à l'approche MDA. Nous avons présenté par la suite une énumération de quelques types de transformation de modèles, suivie par une classification de certaines approches de transformation. En deuxième phase, nous avons focalisé sur un cadre spécifique de transformation de modèles basé sur la transformation de graphes. Enfin, nous avons brièvement présenté AToM^3, l'outil de transformation utilisé dans notre contribution qui consiste en la proposition d'une grammaire de graphe, permettant la transformation des modèles UML-Mobile vers les Réseaux de Petri NestedNets.

Chapitre 5

Une approche intégrée Mobile-UML/ Réseaux de Petri (Contribution II)

5.1. Introduction

Nous avons exposé dans le deuxième chapitre de cette thèse, notre première contribution, qui consiste en l'extension des diagrammes UML2.0 par la Mobilité. En revanche, la transformation de tous les diagrammes étendus proposés dans cette contribution, nécessitent un très grand investissement en terme de temps et d'efforts. Nous avons jugé utile d'investir ces efforts dans nos futurs travaux de recherche, notamment pour l'habilitation.

Dans ce chapitre, nous proposons Une approche intégrée Mobile-UML/Réseaux de Petri pour la transformation du diagramme d'états transitions du M-UML [Kas01], [Kas02]. Afin de pouvoir mettre en œuvre, dans cette thèse, le processus de transformation de modèles, nous nous somme intéressés au diagramme d'états transitions mobiles car il permet de décrire le comportement d'un objet du système modélisé. Par conséquent, ce diagramme permettra de capturer la mobilité d'un agent.

Malgré la commodité de la modélisation et de la compréhension des modèles UML, la vérification de tels modèles s'avère une tâche assez ardue à cause de la sémantique semi-formelle d'UML. Dans ce cadre, plusieurs travaux de recherches se sont intéressés à la vérification et la validation des modèles UML [VIATRA], [Guerra03], [Lara 01], etc.

Notre contribution consiste en la proposition d'une grammaire de graphe, pour transformer les diagrammes d'états transitions mobiles vers les réseaux de Petri Nested-Nets. Nous avons utilisé comme outil de transformation de graphes, l'outil **AToM^3** [ATOM3]. Nous commençons par proposer un métamodèle source pour le diagramme d'états transitions mobiles et un autre métamodèle cible pour le formalisme des réseaux de Petri Nested-Nets. Par la suite, nous exposerons une grammaire de graphe pour transformer les modèles sources vers les modèles cibles.

5.2. UML-Mobile (M-UML)

5.2.1. Déscription du 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. Ce diagramme décrit le fonctionnement des objets en terme de changement d'états, sachant qu'un objet doit se trouver dans un état à un instant donné.

L'***état mobile*** se définit comme étant un état dans lequel se trouve un acteur ou un objet dans une plateforme différente de celle d'origine. Graphiquement, on représente 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 unidirectionnelle reliant deux états (l'état source et l'état cible). On appelle ***transition mobile***, une transition qui relie deux agents ou deux objets qui se trouvent dans des 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 vice versa. Graphiquement, une ***transition mobile*** est représentée comme une transition standard, en ajoutant un petit carré portant le symbole '**M**' dans l'extrémité haute gauche de cette transition. De plus, si un agent arrive dans un état, on 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 distance avec un autre agent, on 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 (cf. la figure 5.1).

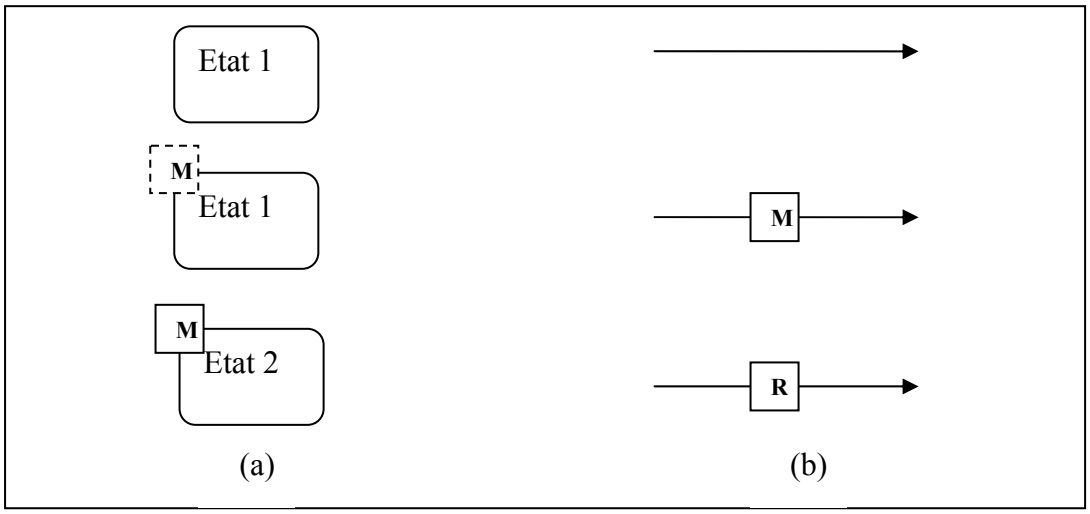


Figure 5.1. Concept de graphe d'état transition mobile
(a): représentation graphique des états ; (b): représentation graphique des transitions.

5.2.2. Méta-modèle du diagramme d'état transition mobile

Notre méta-modèle est composé de 5 classes et 9 transitions comme sur la figure 5.2

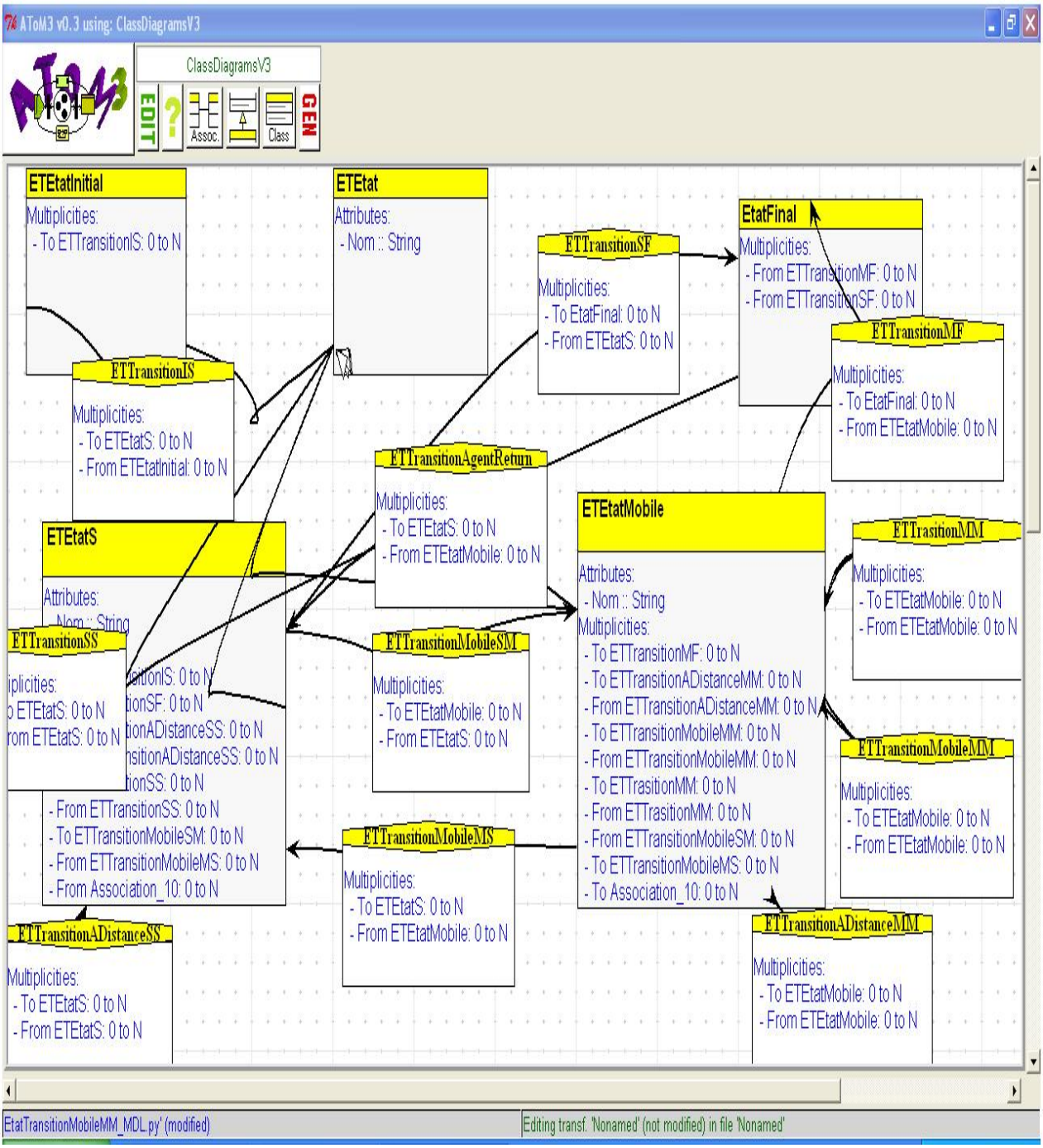


Figure 5.2. Méta-model pour le diagramme d'état transition mobile.

• **La Classe Etat** : Cette classe représente dans un cas général un état dans le diagramme d'état transition mobile. Elle possède un attribut « ***nom »*** de type **String** pour désigner 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***.

• **La Classe Etat initial** : Cette classe représente l'état initial, elle hérite la ***classe Etat*** et elle est graphiquement représentée par un petit cercle plein noir. Elle est connectée avec la ***classe Etat simple*** par ***l'association TransitionIS***, Cette association possède les attributs **PFS** et **PFD** de type String pour désigner 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 **Activités** 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 noire portant tous les attributs.

• **La Classe Etat mobile** : cette classe représente l'état mobile. Elle hérite les attributs de la ***classe Etat*** et elle est graphiquement représentée par un rectangle blanc qui porte à son extrémité haute gauche un petit carré marqué par '**M**'. Dans le cas où cet état est l'état actuel, il sera représenté par un rectangle blanc qui porte à son extrémité haute gauche un petit carré pointillé marqué par '**M**', comme le montre la figure 5.3. Cette classe est connectée à la ***classe Etat simple*** par l'association ***TransitionMobileMS*** dans le cas où une transition mobile existe entre cet état et un état simple, elle est connectée 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 métamodèl, 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 graphiquement représentées par une flèche de couleur noire 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 modéliser le retour de l'agent mobile vers sa plateforme de basse.

La classe ***Etat mobile*** est reliée à elle-même par trois associations:
- L'association ***TransitionMM***: elle modélise la transition simple entre deux états mobiles, et elle est graphiquement représentée 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, Evénement**s et **Activites**. L'association ***TransitionMF*** relie une seule instance de la classe ***Etat mobile*** à une seule instance de la classe ***Etat final***, elle est représentée graphiquement par une flèche de couleur noire 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 (cf. la figure 5.3), 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, Evénement**s et **Activités**. Cette association relie une seule instance de la classe ***Etat simple*** à une seule instance de la classe ***Etat mobile*** , elle est représentée graphiquement comme toutes les transitions mobiles dans le métamodel.

La classe ***Etat simple*** est connectée avec elle-même par deux associations:
- L'association ***TransitionSS*** modélise la transition simple entre deux états simples, elle est représentée 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, elle est représentée graphiquement par une flèche noire portant tous les attributs en ajoutant un petit carrée marqué par **"R"** pour montrer qu'il s'agit 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 à la classe ***Etat final*** par ***l'association TransitionSF***, cette association possède les attributs **PFS, PFD, Evénement**s et **Activités**. L'association ***TransitionMF*** relie une seule instance de la classe ***Etat simple*** à une seule instance de la classe ***Etat final***, elle est représentée graphiquement par une flèche de couleur noire 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.

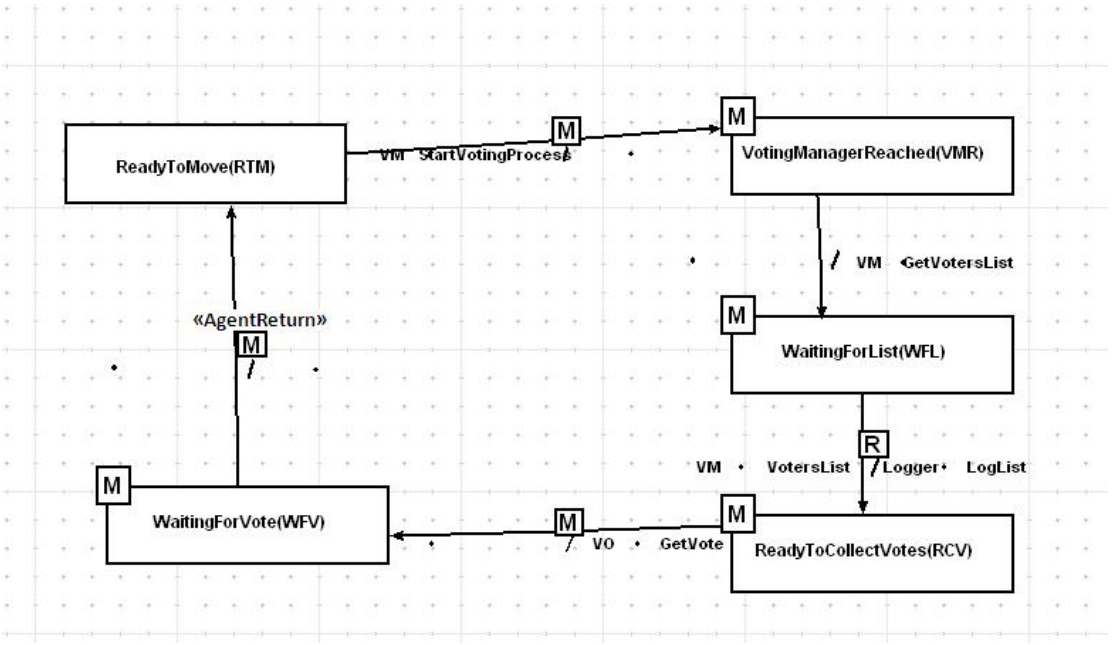


Figure 5.3. Modélisation de diagramme d'états transitions Mobiles

5.3. Les réseaux de Petri Nested Nets

5.3.1 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*** pour représenter les concepts de la mobilité tels que : la localité, la migration, le clonage ainsi que le suivit du cycle de vie d'un agent mobile. Comme il a été présenté dans le chapitre3, un réseau de Petri Nested-Nets possède **n** (n ϵ N) niveaux d'abstraction. Dans notre contribution, nous avons opté pour 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 d'un agent mobile.

Le 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**.

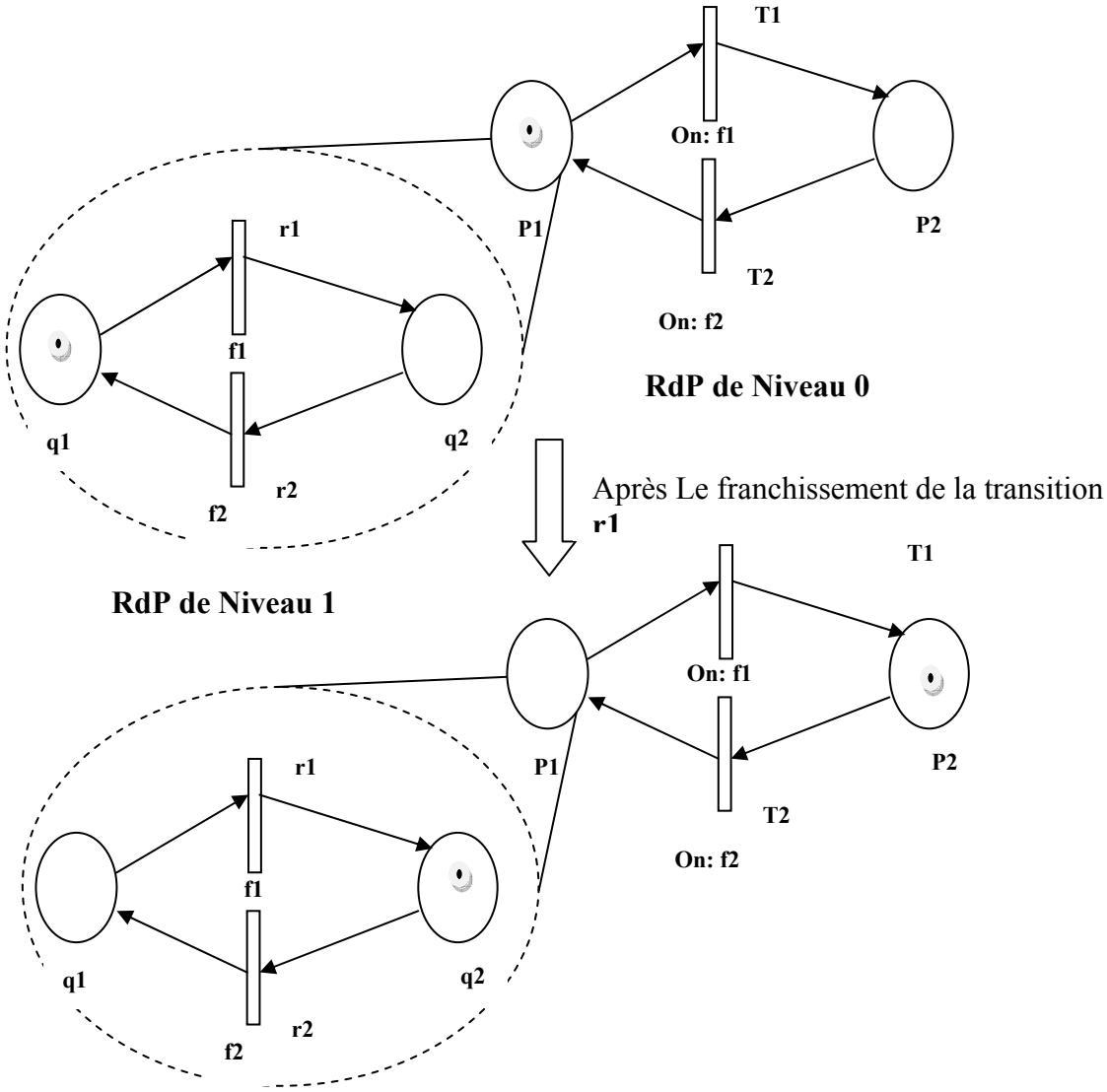


Figure 5.4. Exemple de franchissement d'une transition dans le RdP Nested Nets

5.3.2. Méta-modèle du Nested Nets

Le Méta-modèle proposé est composé de six classes et six transitions comme le montre le figure 5.5 ci dessous:

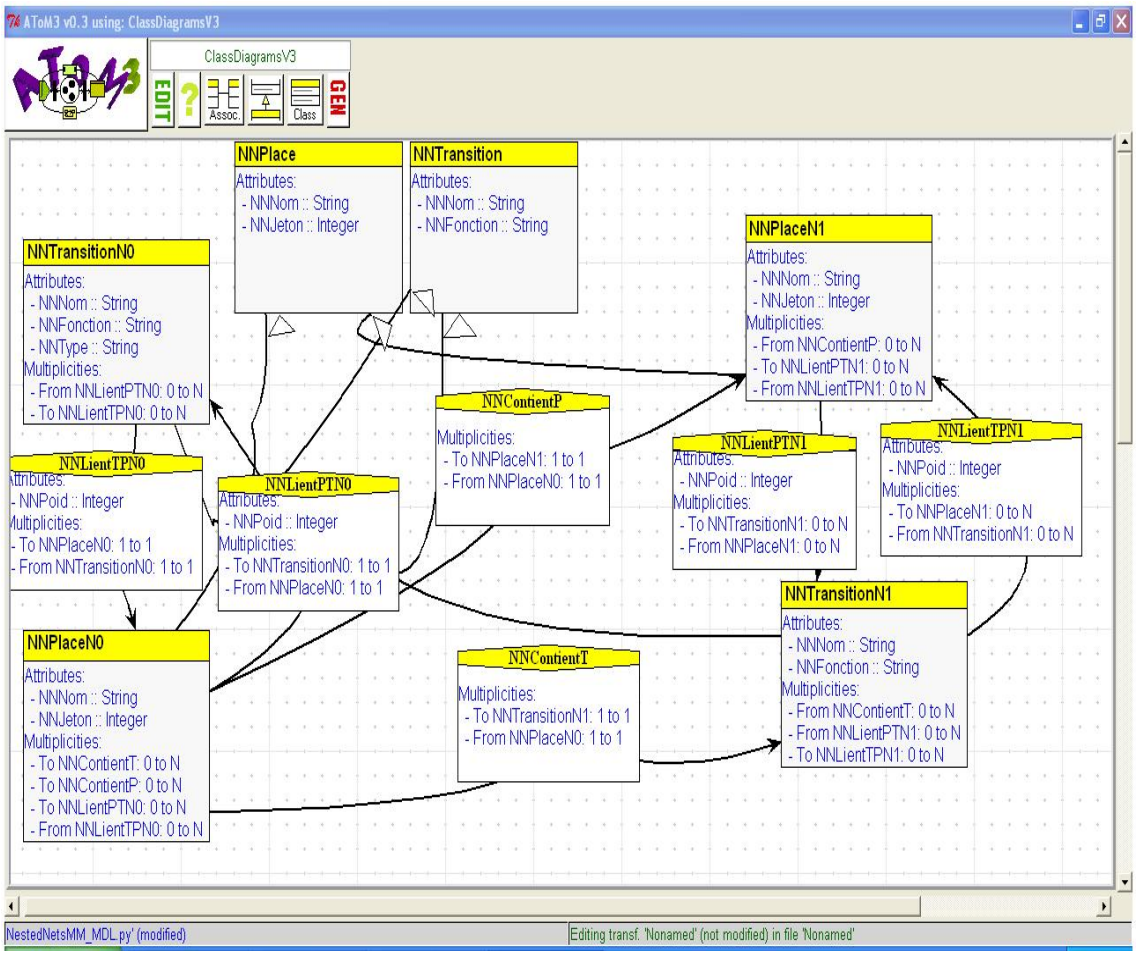


Figure 5.5. le méta-model pour les Nested Nets.

• **La 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 jetons 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***, il est utilisé pour déterminer le nom de la transition, le deuxième s'appelle ***Fonction,*** il 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 lien. L'association ***LientPTN0*** est représentée graphiquement par une flèche noir portant l'attribut ***Poids*** (cf. la figure 5.6).

• **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***. Il existe trois types 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 à droite de ce rectangle.

- ***A distance:*** Ce type de transition est utilisé 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 return***: 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 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 '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 un seul attribut : ***Poids*** de type entier utilisé pour déterminer le poids du lien. Cette association est représentée graphiquement par une flèche noire portant l'attribut ***Poids.***

• **Classe PlaceN1:** cette classe représente la place dans le réseau de Petri de niveau 1, 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 connecte 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 lien, l'association ***LientPTN0*** est représentée graphiquement par une flèche noire 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 représentée 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 noire portant l'attribut ***Poids.***

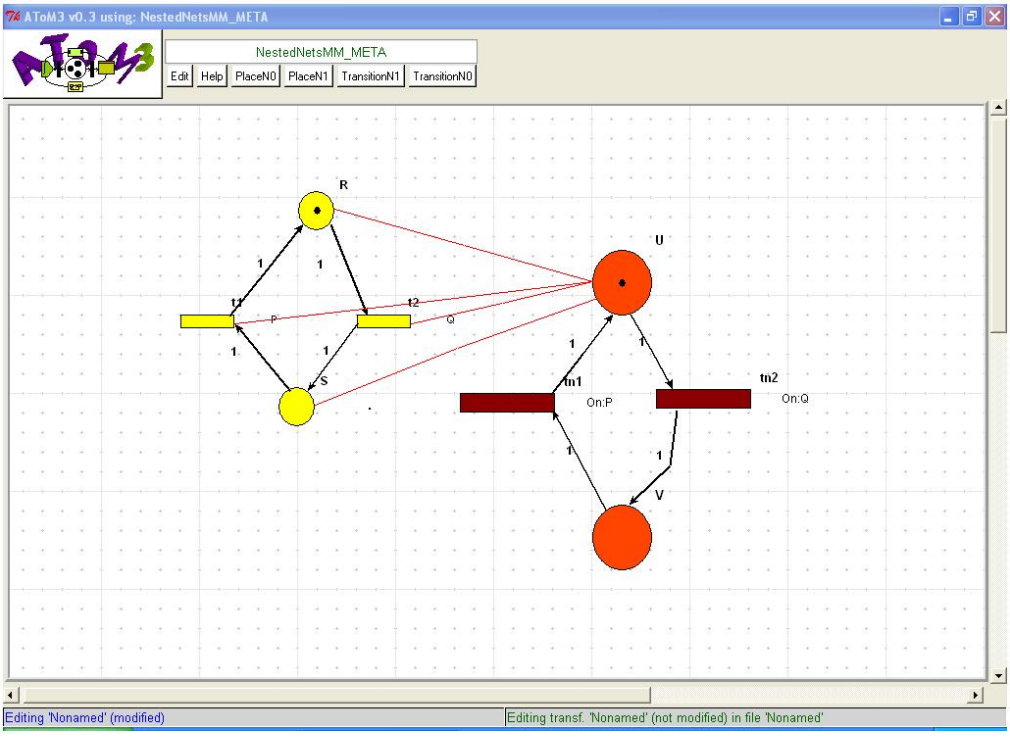


Figure 5.6. Outil de modélisation des modèles diagramme d'état transition

5.4. La grammaire de graphe proposée

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

5.4.1 Les règles 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 (cf. la 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 à 0 car cet état n'est pas considéré comme un état actuel.

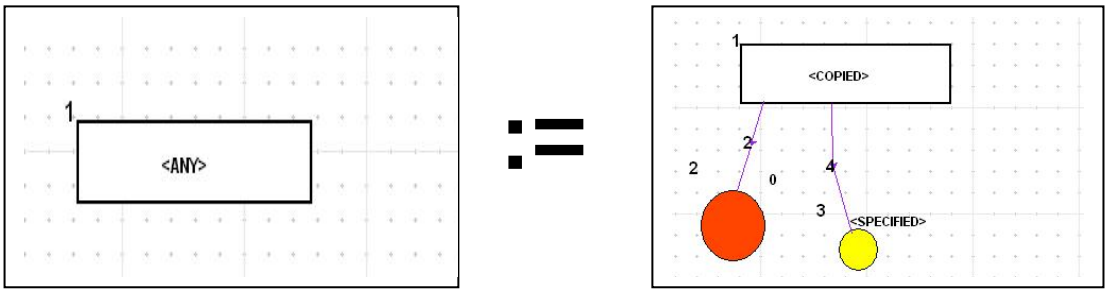


Figure 5.7. Transformation d'état simple

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 (cf. la 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 égal à 1 parceque cet état est considéré comme un état actuel.

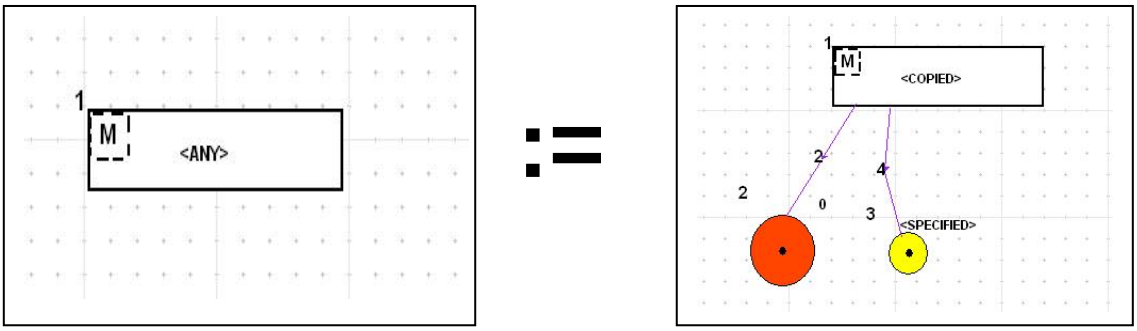


Figure 5.8. Transformation d'état actuel simple

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 (cf. la figure 5.9). 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 égal à 0 parceque cet état n'est pas considéré comme un état actuel.

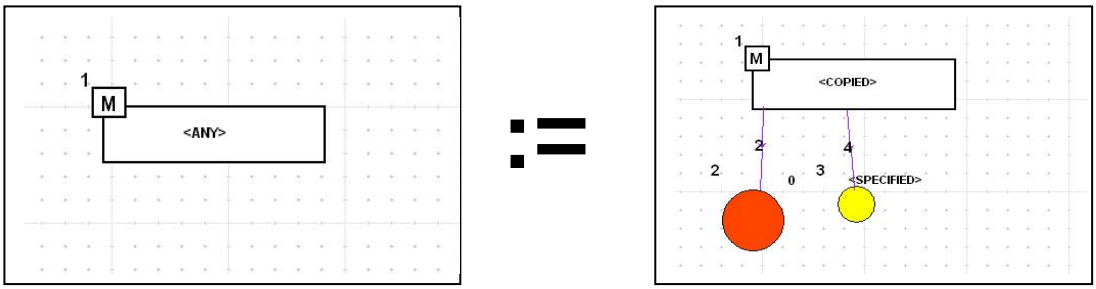


Figure 5.9. Transformation d'état mobile

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 (cf. la 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 égal a 1 par ce que cet état est considéré comme un état actuel.

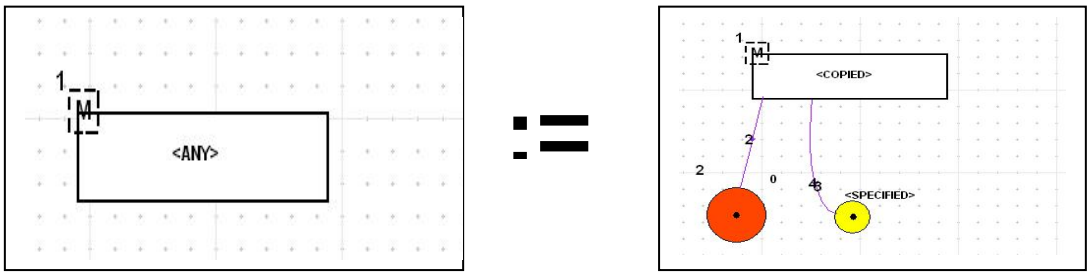


Figure 5.10. Transformation d'état actuel mobile

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 (cf. la figure 5.11). Cette transition porte le nom initial.

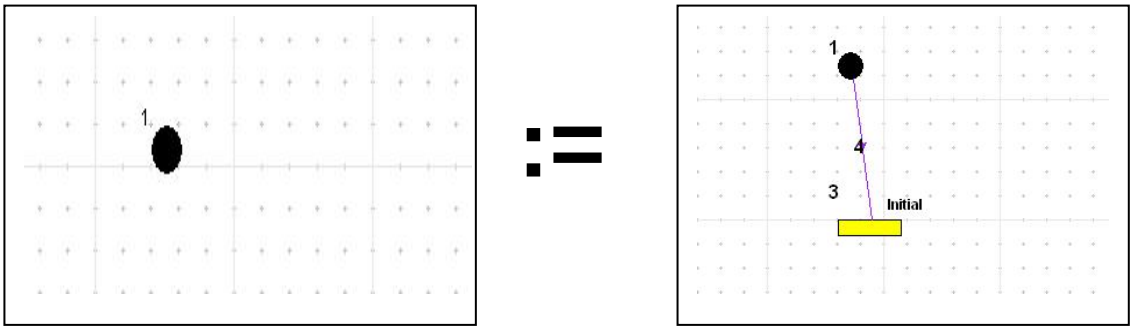


Figure 5.11. Transformation d'état initial

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 (cf. la figure 5.12). Cette transition porte le nom final.

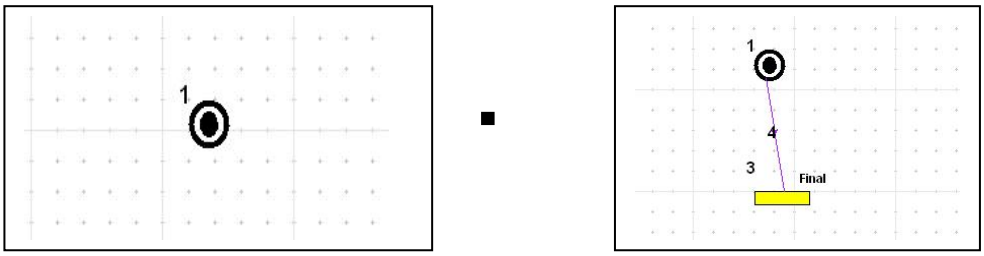


Figure 5.12. Transformation d'état final

5.4.2 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 (cf. la 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, on élimine alors le jeton de la deuxième place (nœud 5 dan LHS), parceque l'agent ne peut se trouver à un instant donné que dans un seul état.

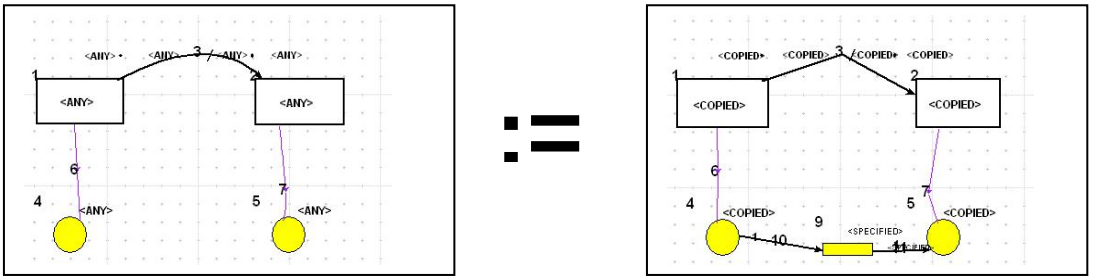


Figure 5.13. Transformation de transition simple entre deux états simples

Règle 8 : **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 (cf. la figure 5.14). Dans ce cas, il n'existe aucune fonction de synchronisation car il n'y a 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, on élimine alors le jeton de la deuxième place (nœud 6 dan LHS) parceque l'agent ne peut se trouver à un instant donné que dans un seul état.

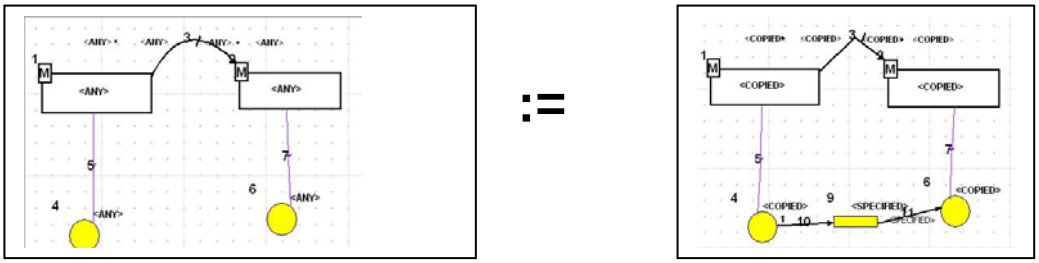


Figure 5.14. Transformation de transition simple entre deux états mobile

Règle 9 : **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 (cf. la figure 5.15). La fonction de synchronisation prend comme nom la concaténation des attributs suivants de la transition du diagramme source: Evénements et Activités

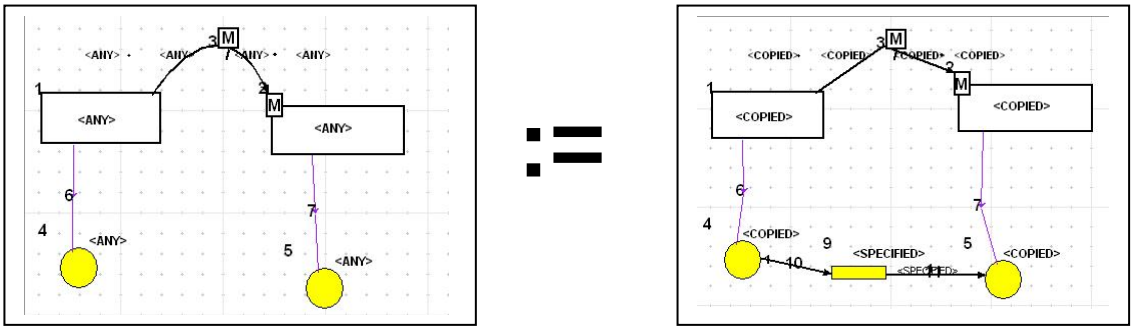


Figure 5.15. Transformation de transition mobile entre un état simple et un état mobile

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 (cf. la 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.

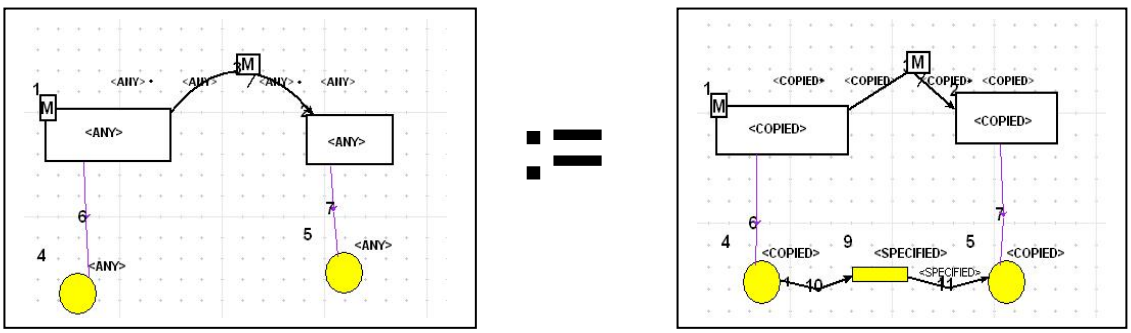


Figure 5.16. Transformation de transition mobile entre un état mobile et un état simple

Règle 11 : **Transformation d'une transition mobile entre deux 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 (cf. la 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.

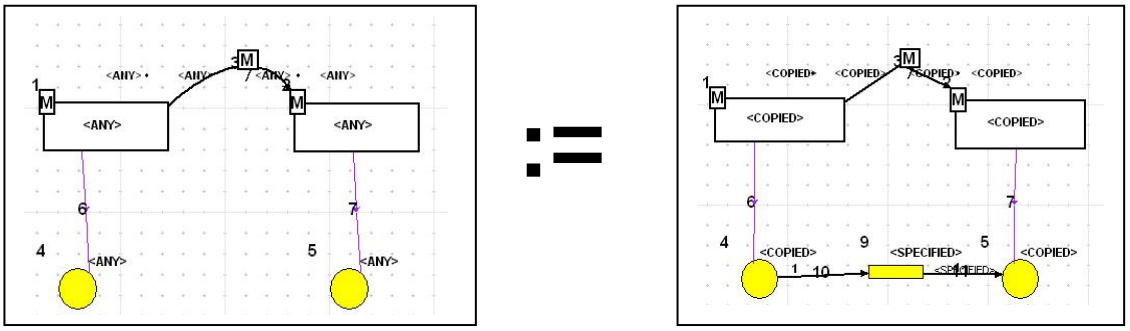


Figure 5.17. Transformation de transition mobile entre deux états mobiles

Règle 12 : **Transformation d'une transition à distance entre deux 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 (cf. la 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.

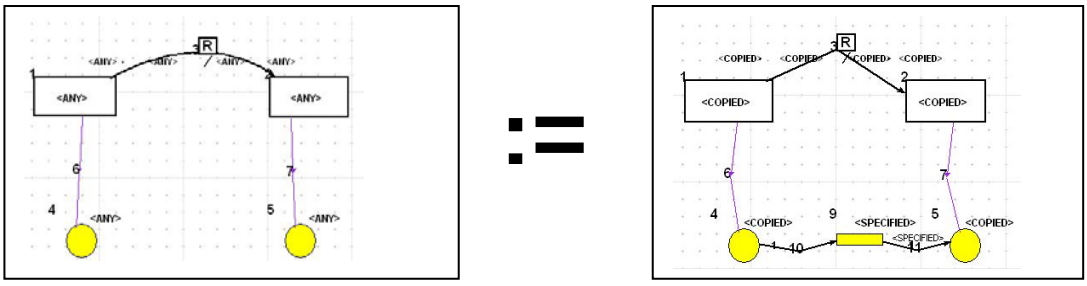


Figure 5.18. Transformation de transition à distance entre deux états simples

Règle 13 : **Transformation d'une transition à distance entre deux 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 transition prend comme nom, à partir de la transition du diagramme source, la concaténation entre les attributs: PFS, PFD, Evénements et Activités (cf. la 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.

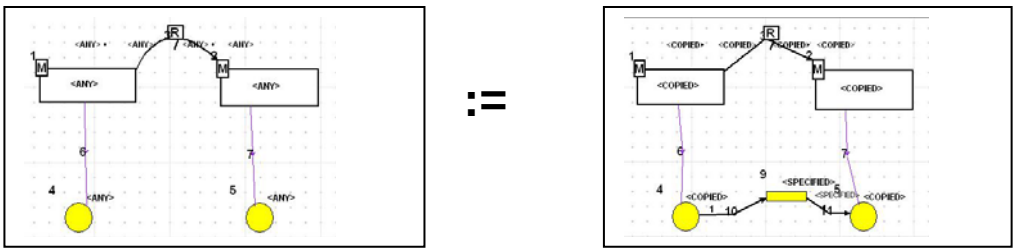


Figure 5.19. Transformation de transition à distance entre deux états mobiles

Règle 14 : **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 (cf. la 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

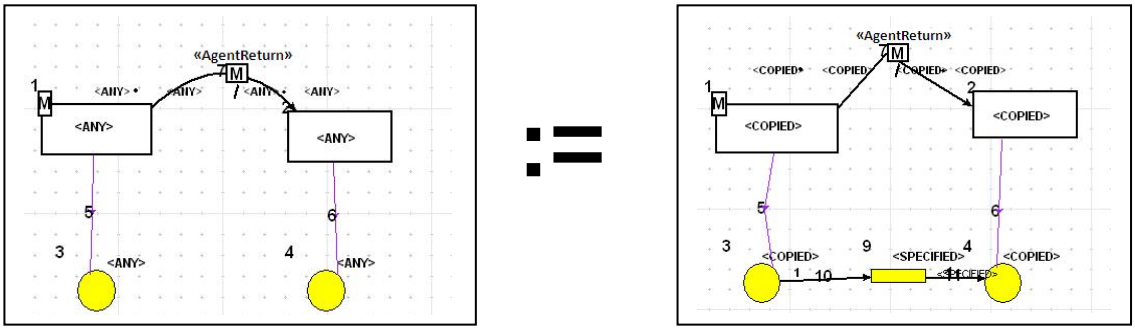


Figure 5.20. Transformation d'une transition mobile «agent return» entre un état mobile et un état simple

Règle 15 : **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 lien de poids égal à 1 dans le Nested Nets (cf. la figure 5.21).

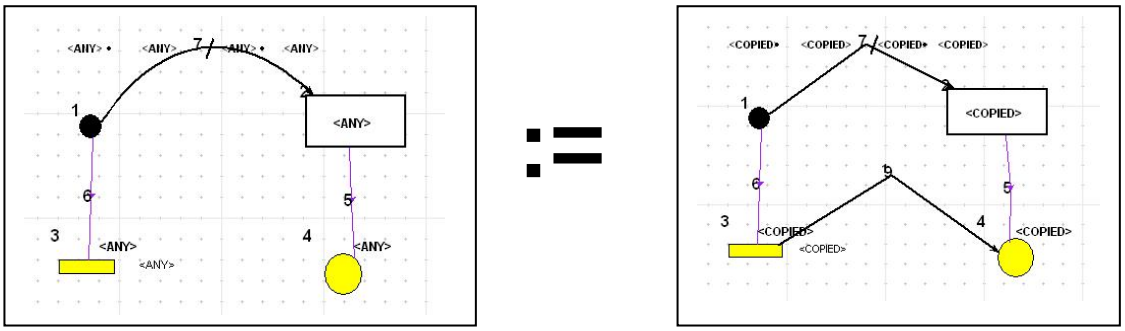


Figure 5.21. Transformation d'une transition entre un état initial et un état simple

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 lien de poids égal à 1 dans le Nested Nets (cf. la figure 5.22).

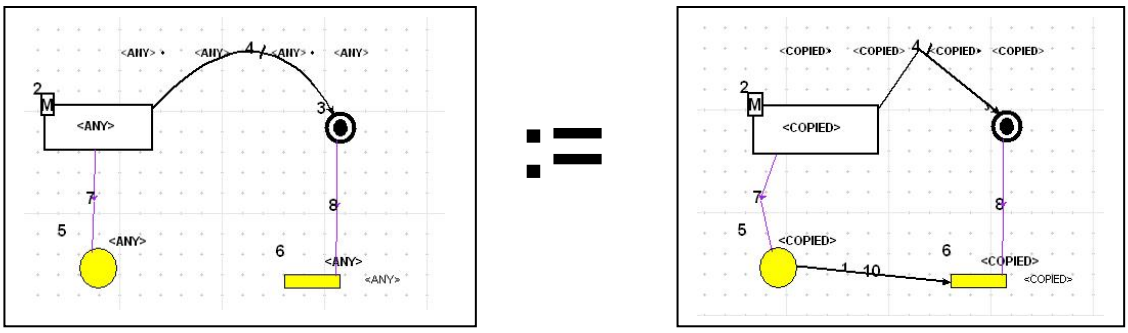


Figure 5.22. Transformation d'une transition entre un état mobile et un état final

<u>Règle 17</u> : **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 lien de poids égal à 1 dans le NestedNets (cf. la figure 5.23).

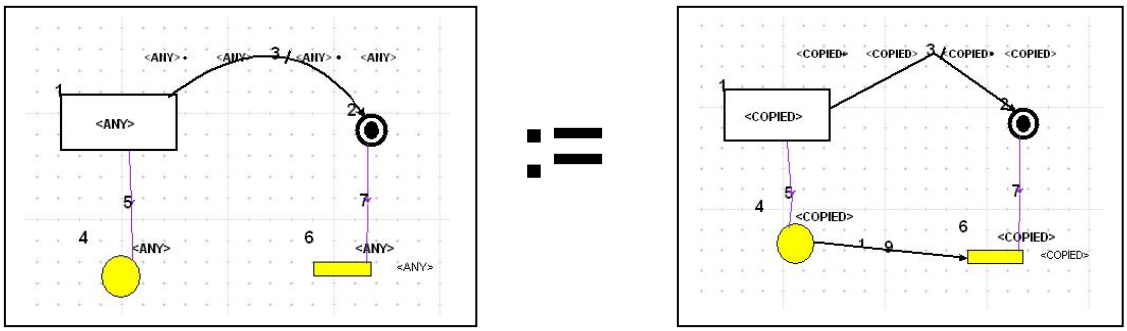


Figure 5.23. Transformation d'une transition entre un état simple et un état final

<u>Règle 18</u> : **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 (cf. la 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**.**

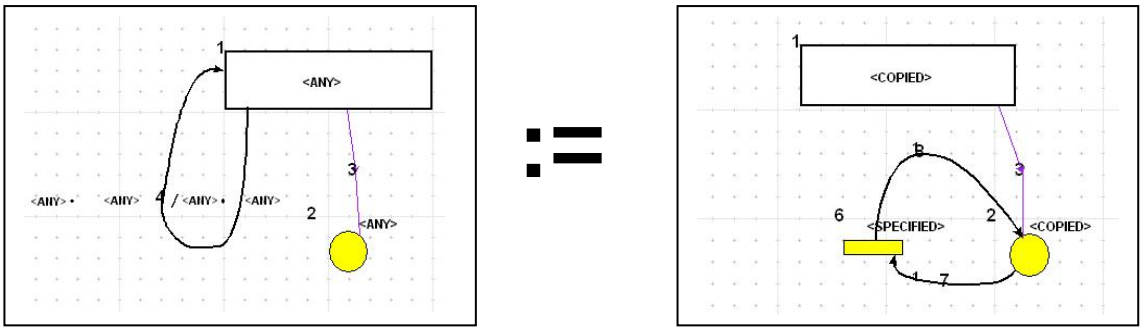


Figure 5.24. Transformation d'une transition d'un état simple vers lui même

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 (cf. la 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**.**

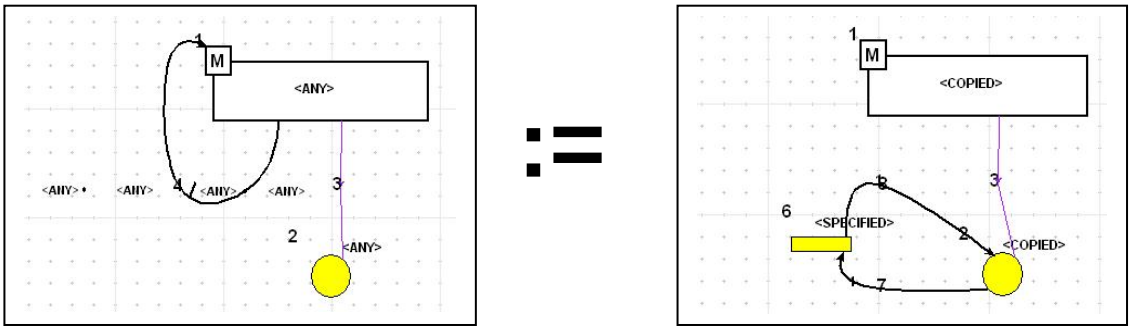


Figure 5.25. Transformation d'une transition d'un état mobile vers lui même

5.4.3. Les règles de transformation de l'ensemble des transitions aux transitions de niveaux 0: cette catégorie regroupe douze règles qui visent à transformer les transitions du modèle source vers des transitions de niveau 0 dans le modele 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 (cf. la figure 5.26). La fonction 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 désigner la plateforme de base**.**

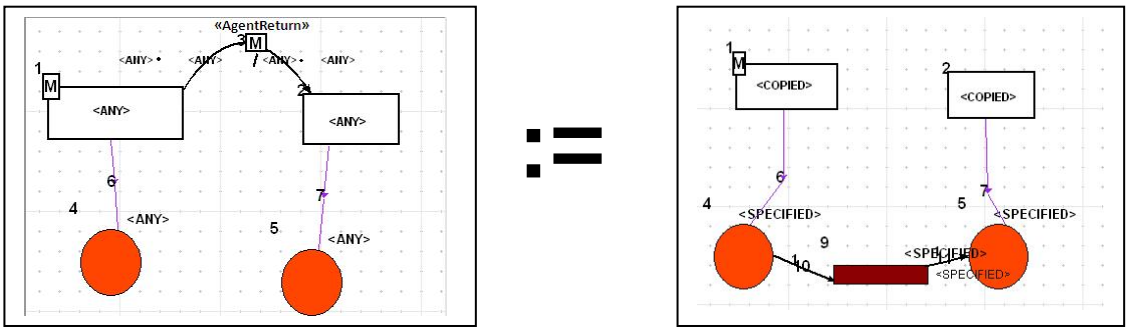


Figure 5.26. Transformation d'une transition entre un état mobile et un état simple

<u>Règle 21</u> : 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 (cf. la 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**.**

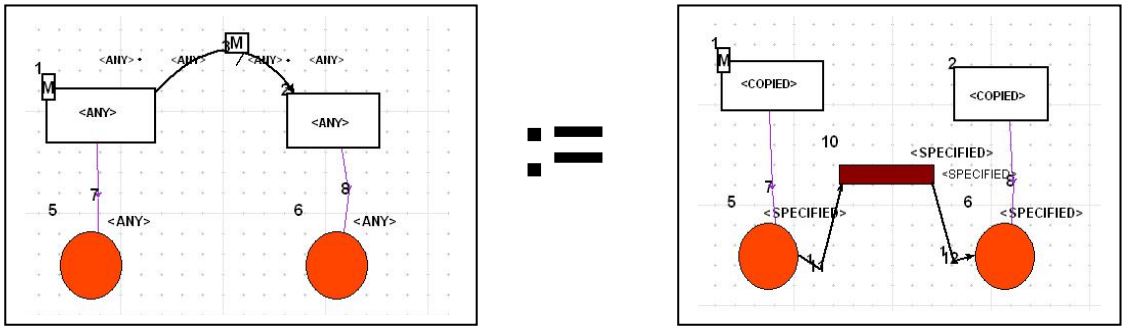


Figure 5.27. Transformation d'une transition mobile entre un état mobile et un état simple

<u>Règle 22</u> : **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 supprimant 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 supprimée (nœud 7 dans LHS) (cf. la 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.

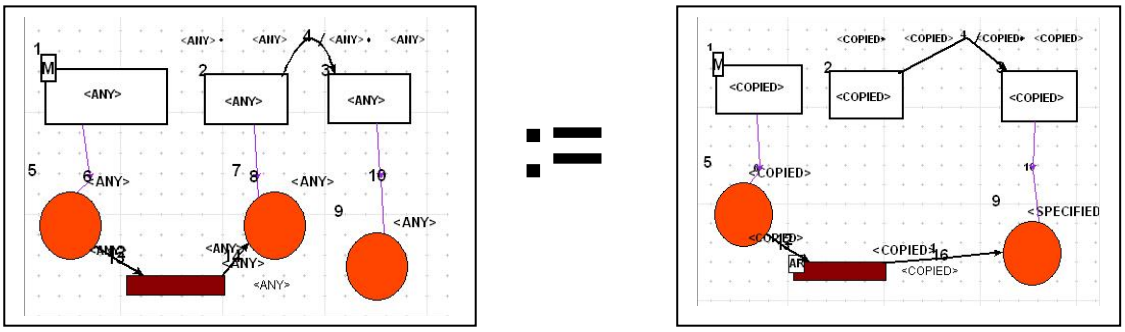


Figure 5.28. Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états simples

<u>Règle 23</u> : **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 (cf. la 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)**.**

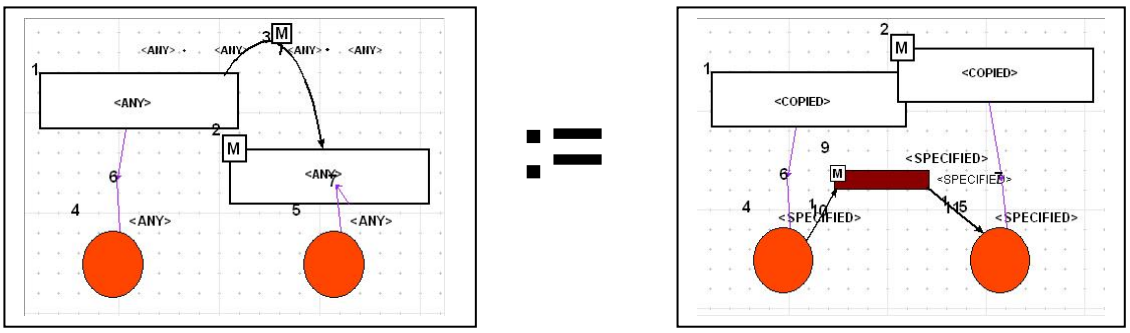


Figure 5.29. Transformation d'une transition mobile entre un état simple et un état mobile

<u>Règle 24</u> : **Transformation d'une transition mobile entre 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 (cf. la 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) prend comme nom l'attribut ***PFS*** de la transition du diagramme source (nœud 3 dans LHS), et la deuxième place (nœud 6 dans LHS) prend comme nom l'attribut ***PFD*** de la transition du diagramme source (nœud 3 dans LHS)**.**

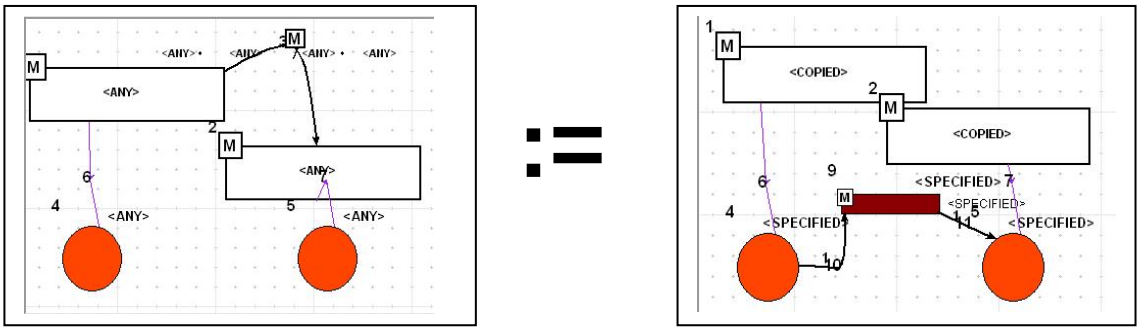


Figure 5.30. Transformation d'une transition mobile entre deux états mobiles

<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 supprimant 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 supprimée

(nœud 5 dans LHS) (cf. la figure 5.31). 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.

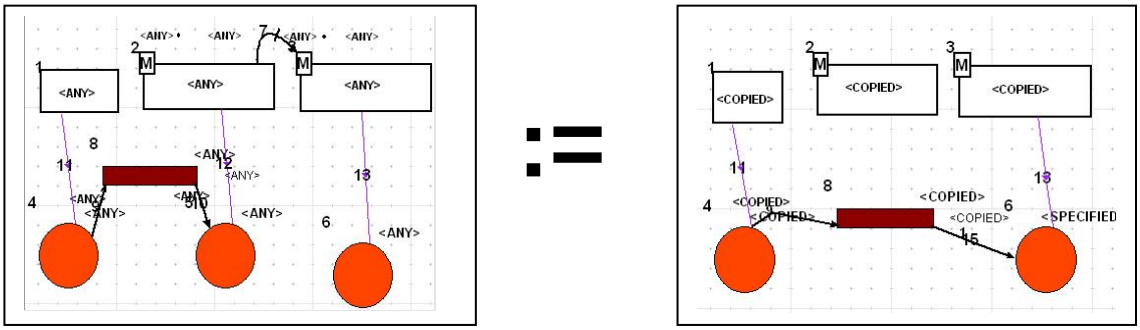


Figure 5.31. Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états mobiles

<u>Règle 26</u> : **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 supprimant 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 supprimée (nœud 6 dans LHS) (cf. la figure 5.32). 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.

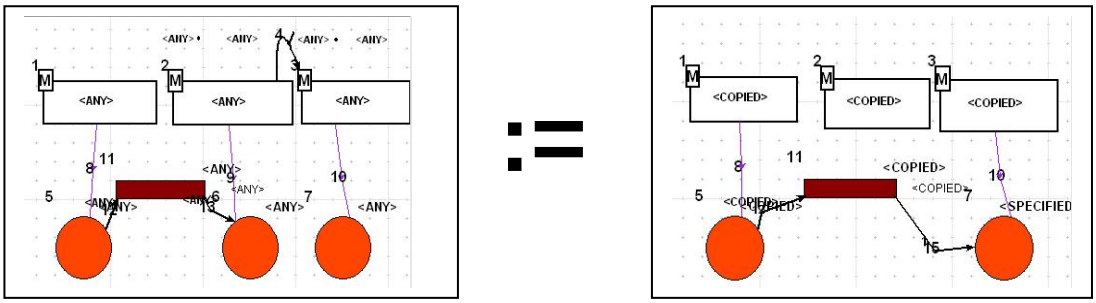


Figure 5.32. Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états mobiles

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 (cf. la 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 précédés par le mot **"On:".** La première et la deuxième place (nœud 4 et 5 dans LHS) prend comme nom le mot "PFB" pour dire que ces plateformes sont des plateformes de base**.**

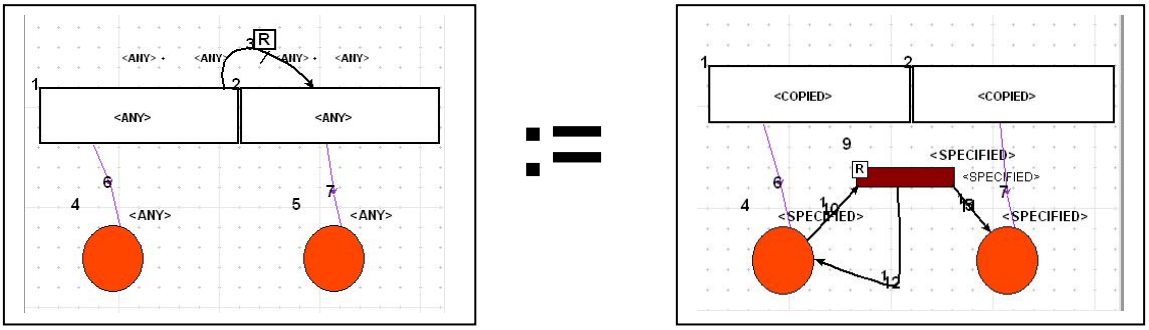


Figure 5.33. Transformation d'une transition à distance entre deux états simples

Règle 28 : **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 (cf. la figure 5.34). La fonction de synchronisation prend comme nom, à partir de la transition

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

-

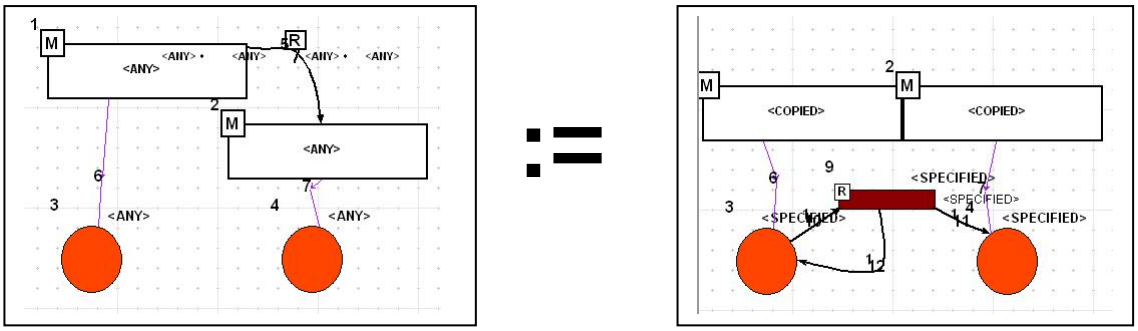


Figure 5.34. Transformation d'une transition à distance entre deux états mobiles

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

- **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 (cf. la figure 5.35).

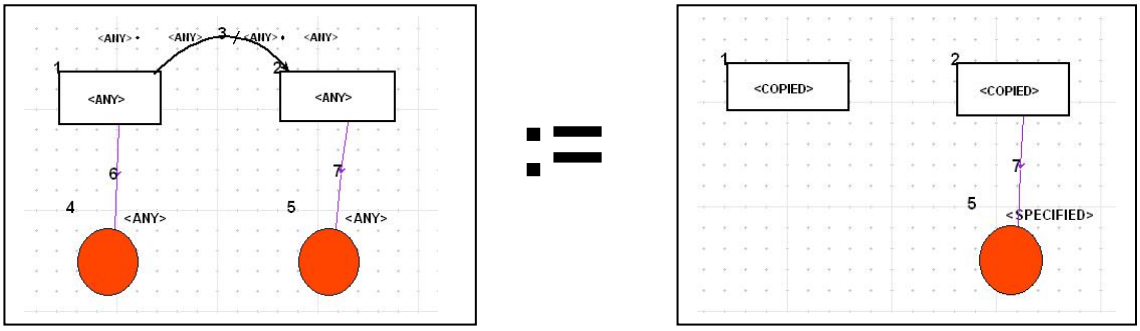


Figure 5.35. élimination des places inutiles de niveau 0

Règle 30 : **élimination des places inutiles de niveau 0**

- **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 (cf. la figure 5.36).

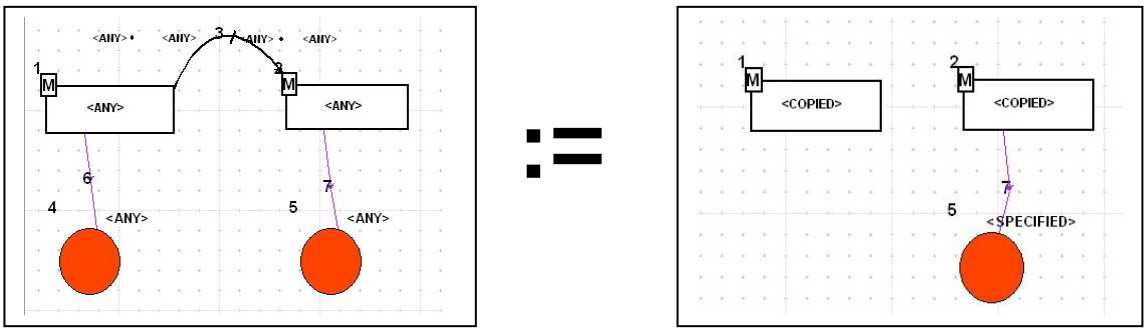


Figure 5.36. élimination des places inutiles de niveau 0

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 supprimant 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 supprimée (nœud 6 dans LHS) (cf. la figure 5.37). Ce remplacement est fait parce que chaque deux états connectés par une transition simple sont des états de la même plateforme, sachant que l'une des deux places de niveaux N0 est toujours supprimée.

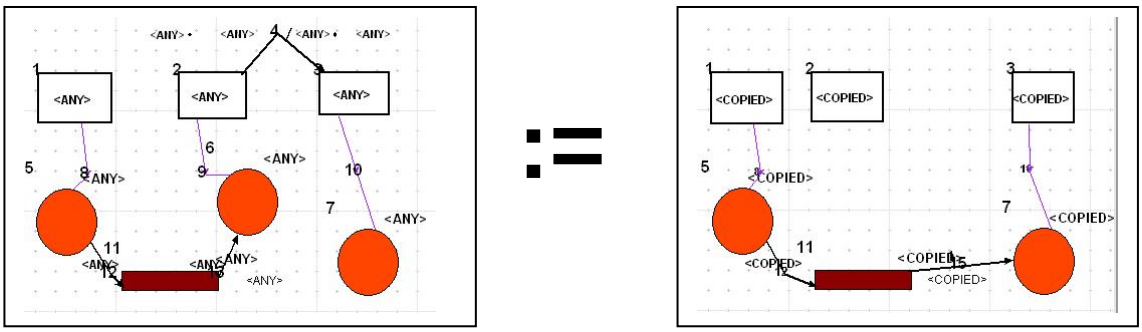


Figure 5.37. Transformation d'une transition entre deux places de niveau N0 et une transition simple entre deux états simples

5.4.4. 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 envoie un message à distance attend une réponse (Transition de retour dans le RDP N0) (cf. la 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).

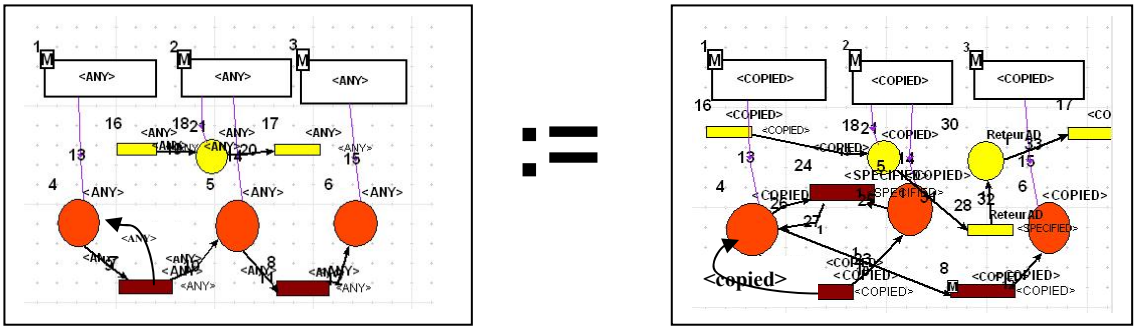


Figure 5.38. Changement de la structure des RDPs de deux niveaux du Nested Nets

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 envoie un message à distance attend une réponse (Transition de retour dans le RDP N0) (cf. la figure 5.39). 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), 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) concaténé 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).

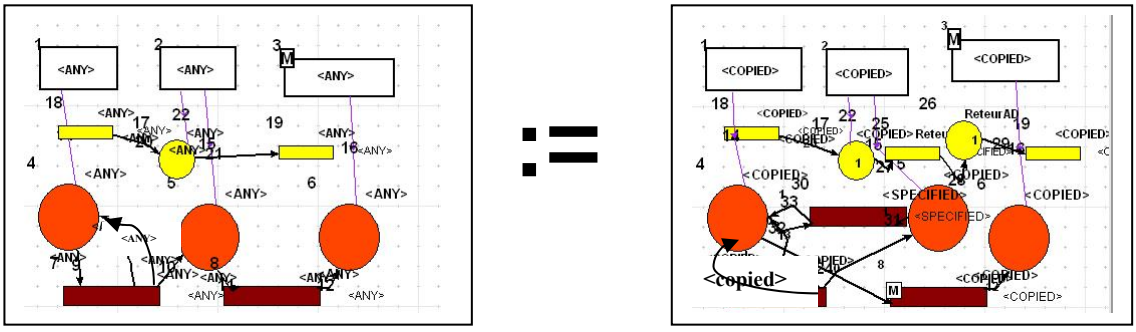


Figure 5.39. Changement de la structure des RDPs de deux niveaux du Nested Nets

Règle 34 : **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 envoit un message à distance attend une réponse (Transition de retour dans le RDP N0) (cf. la 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 prend pour valeur la fonction de la transition N0 de type 'R' dans le LHS (nœud 7) concaténé 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).

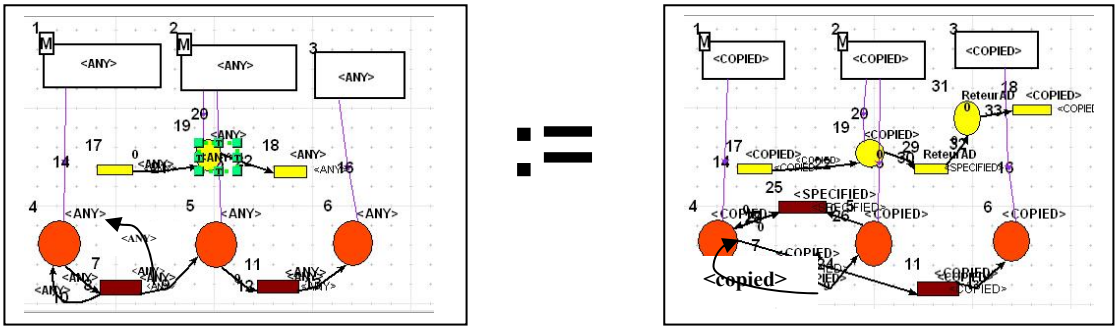


Figure 5.40. Changement de la structure des RDPs de deux niveaux du Nested Nets

5.4.5. Les règles qui éliminent les états du digramme d'état transition mobile: cette catégorie regroupe trois règles qui visent à éliminer les états du diagramme 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 (cf. la figure 5.41).

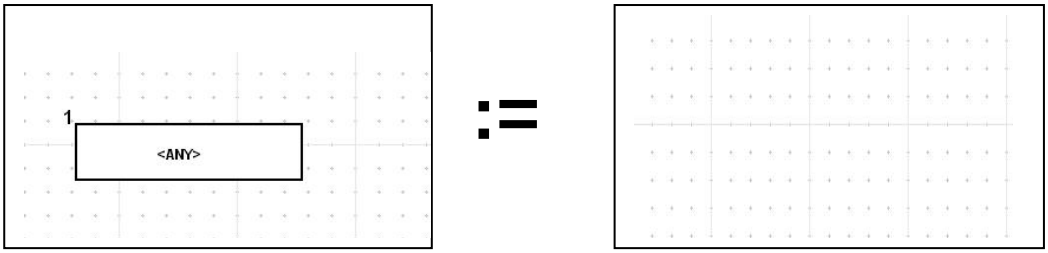


Figure 5.41. Elimination des états simples

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 (cf. la figure 5.42).

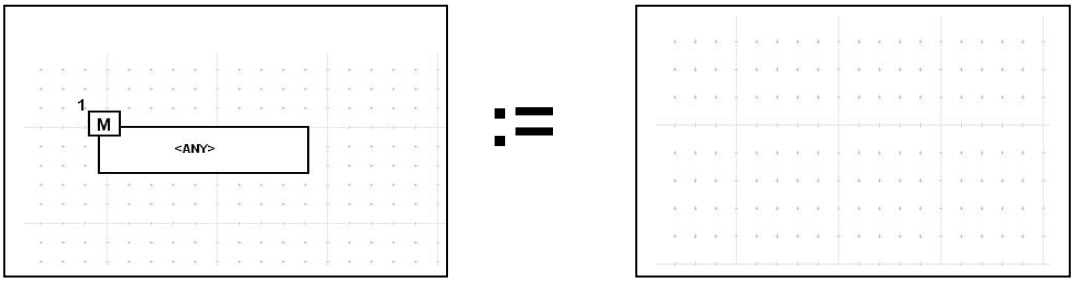


Figure 5.42. Elimination des états simples

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 (cf. la figure 5.43).

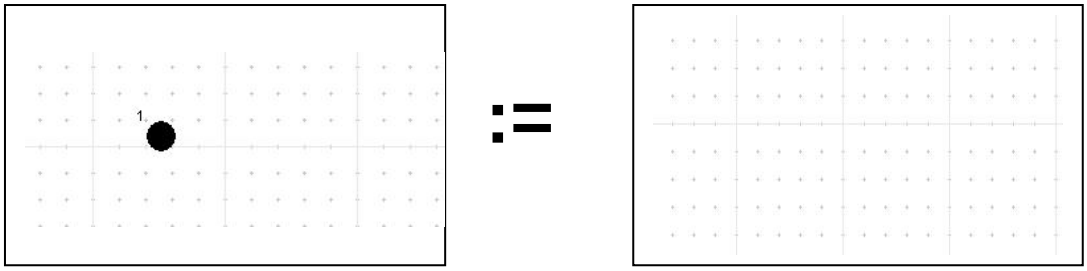


Figure 5.43. Elimination des états initiaux

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 (cf. la figure 5.44).

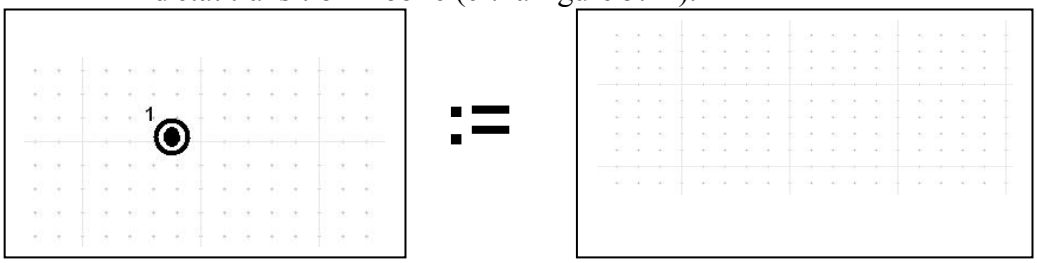


Figure 5.44. Elimination des états finaux

5.4.6. Les règles relient 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 jetons dans la place N0 égale à 1 (cf. la figure 5.45).

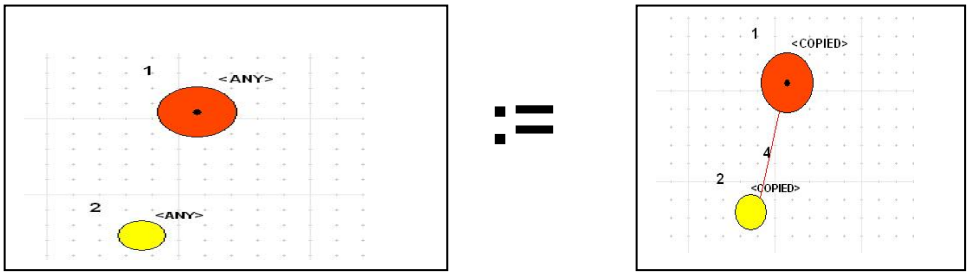


Figure 5.45. Un lien entre une place de niveau N0 et une place de niveau N1

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 jetons dans la place N0 égale à 1 (cf. la figure 5.46).

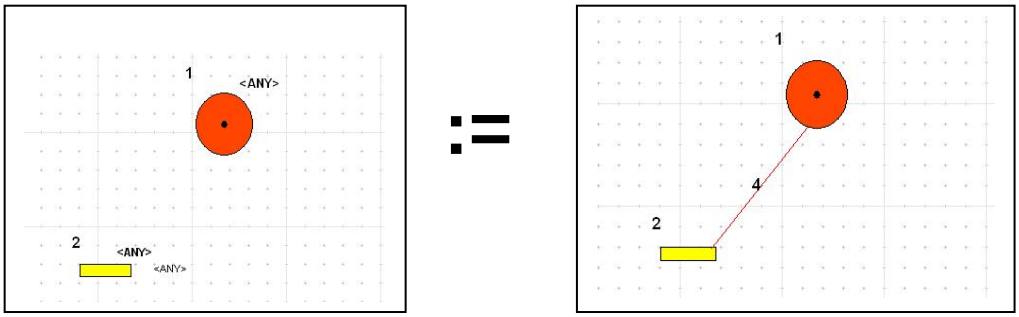


Figure 5.46. Un lien entre une place de niveau N0 et une transition de niveau N1

5.4.7. Premier exemple 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.

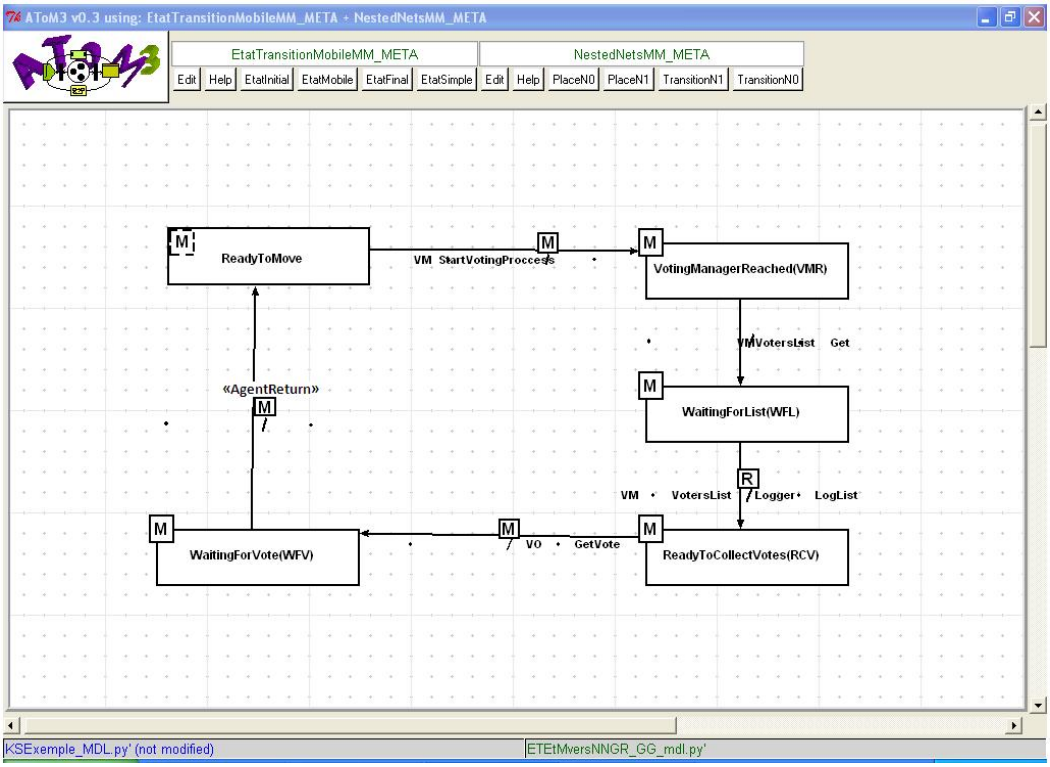


Figure 5.47. Exemple d'un diagramme d'état transition mobile

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

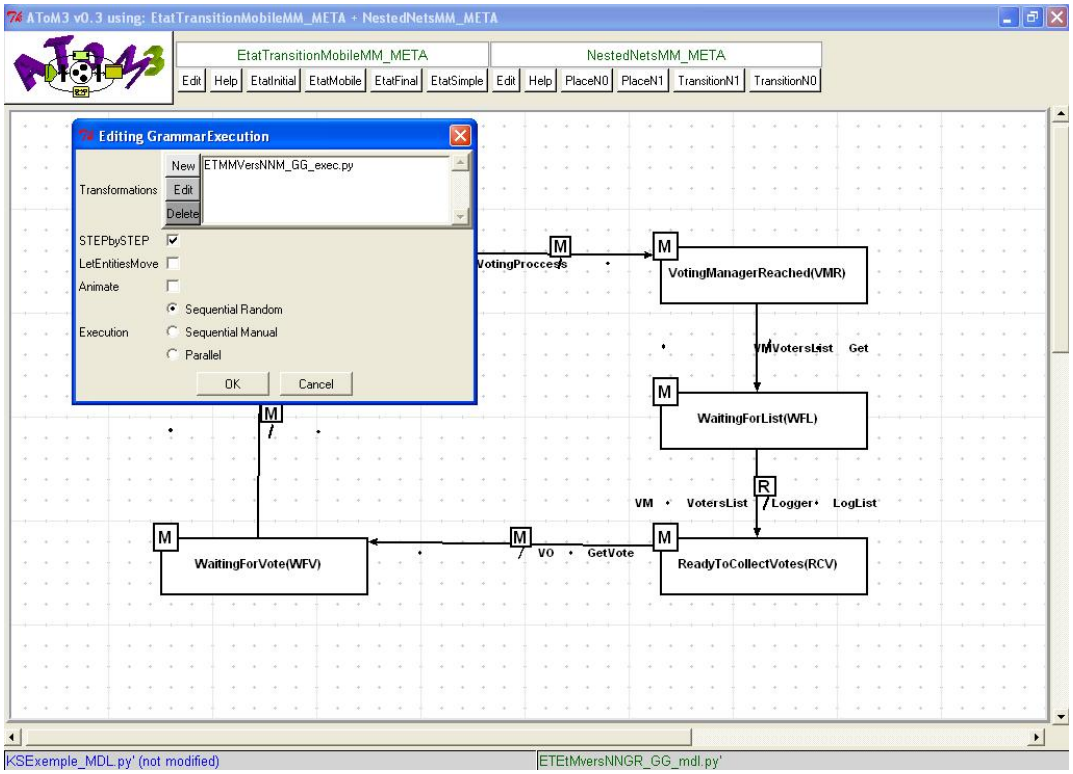


Figure 5.48. Lancement de l'exécution de la grammaire

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.

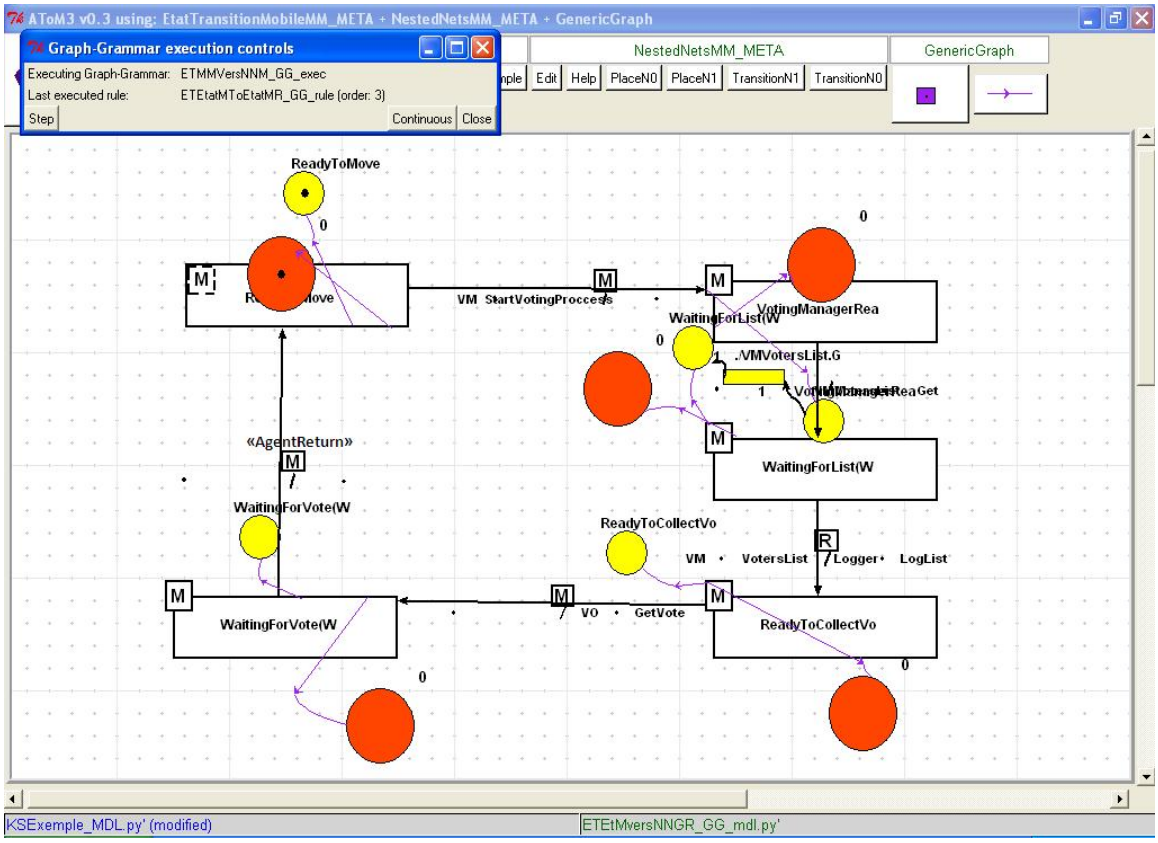


Figure 5.49. graphe intermédiaire obtenu après l'exécution de la règle 8

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.

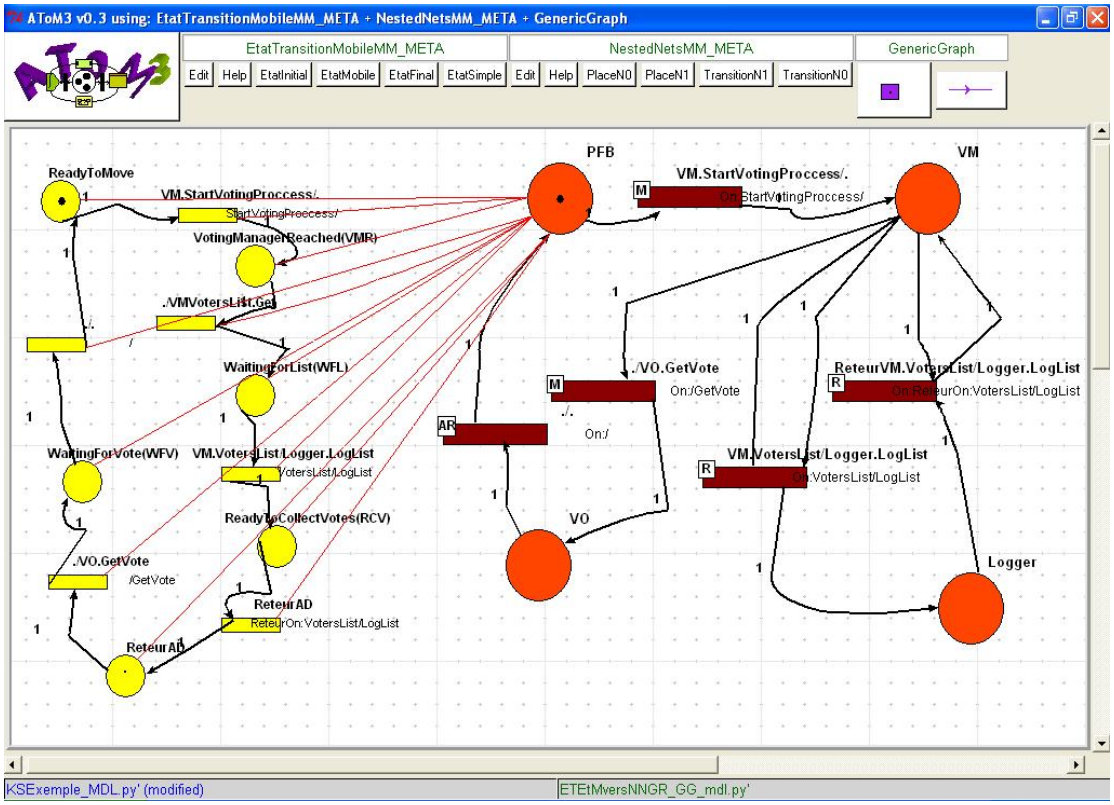


Figure 5.50. NestedNets résultat de la transformation.

Pour vérifier le NestedNets 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 chacun 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. Deuxième exemple de transformation d'un diagramme d'état transition mobile vers un Nested Nets

On utilise l'exemple présenté dans [Kas02] (cf. les figures 5.51, 5.52, 5.53).

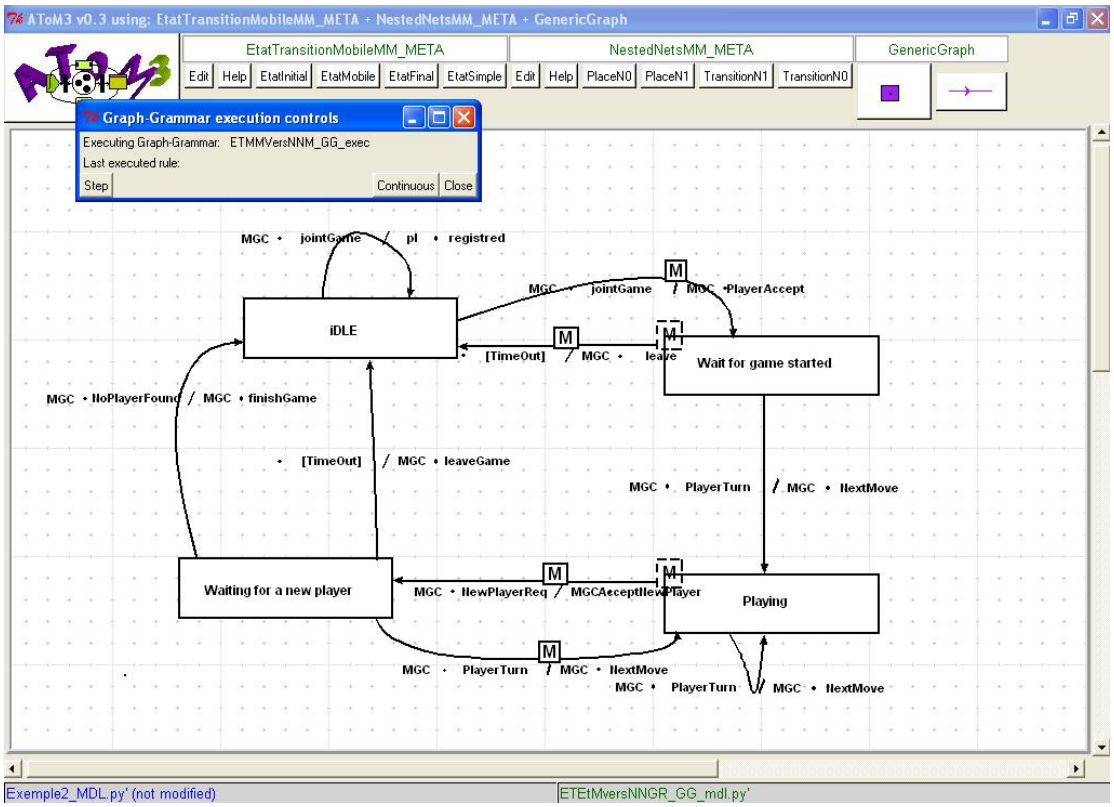


Figure 5.51. Exemple d'un diagramme d'état transition mobile

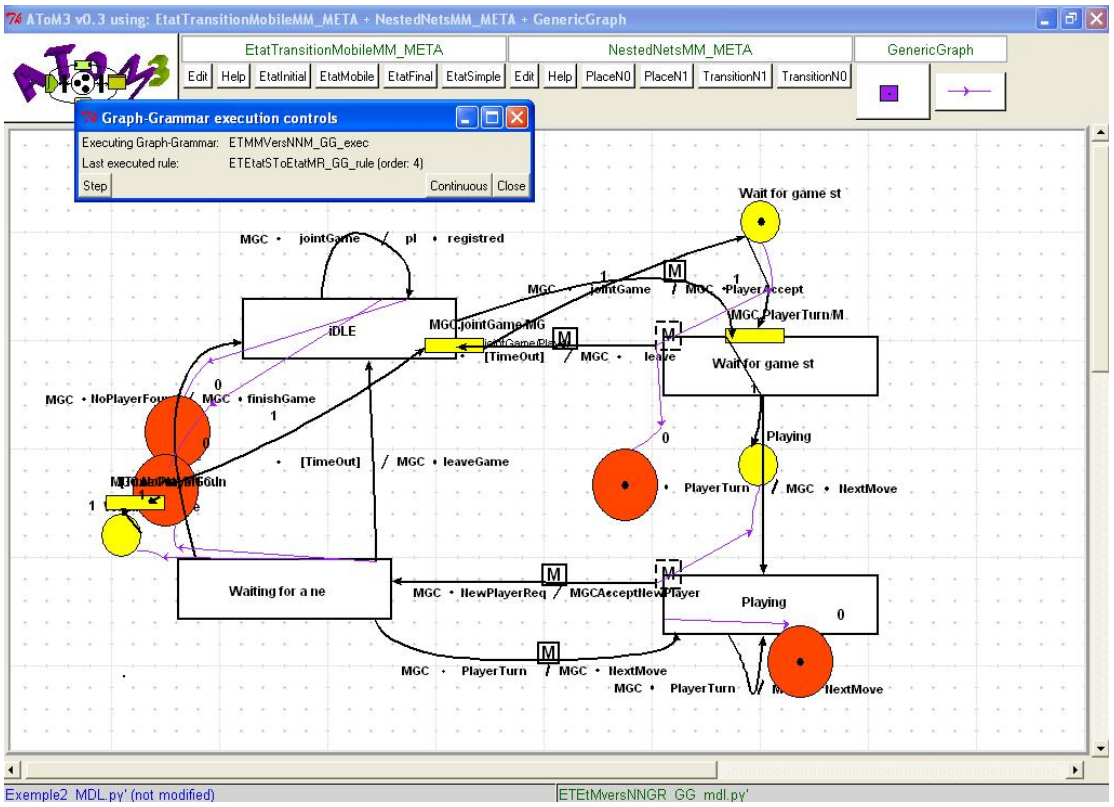


Figure 5.52. Graphe intermédiaire obtenu après l'exécution de la règle 9

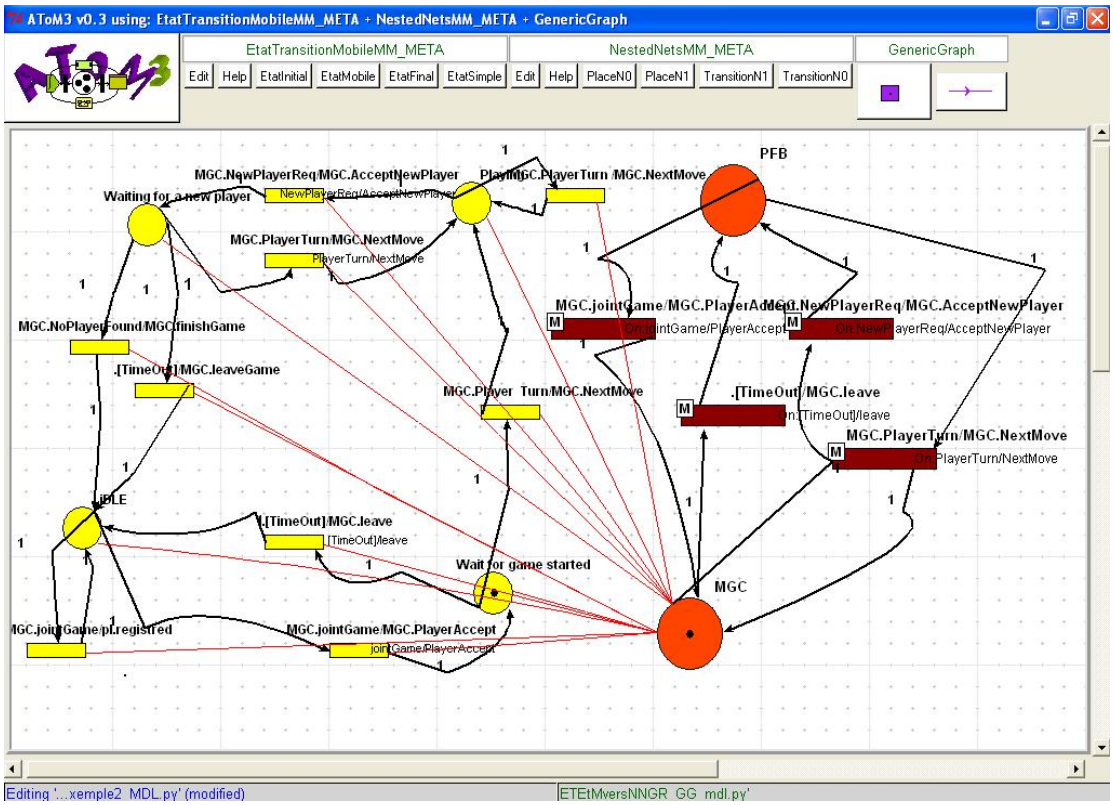


Figure 5.53. NestedNets résultat de la transformation.

5.6. Conclusion

Dans ce chapitre, nous avons proposé une grammaire de graphe pour la transformation automatique de diagramme d'états transitions mobiles vers les réseaux de Petri NestedNets, à deux niveaux d'abstraction, afin de pouvoir effectuer des vérifications automatiques. Nous avons utilisé AToM^3 [ATOM3] comme outil de transformation de graphe.

Nous commençons par une proposition d'un métamodèle pour le diagramme d'état transition mobile et un autre métamodèle pour le formalisme de RdP NestedNets. Nous avons proposé par la suite, une grammaire de graphe pour transformer le formalisme source vers le formalisme cible. Nous avons illustré notre proposition par deux exemples réels, où nous jugeons avoir obtenu de bons résultats.

Comme perspective et pour compléter notre travail futur nous voudrons lancer un grand chantier dans lequel on réalisera la transformation des modèles UML2.0 stéréotypés présentés dans notre première contribution, vers les réseaux de Petri Nested-Net.

Conclusion générale et perspectives

Dans cette thèse, nous nous sommes intéressés au paradigme des agents mobiles. Nous avons introduit dans le premier chapitre la notion d'agent en général, suivie par le paradigme des agents mobiles. Le constat fait à ce niveau a montré que le développement des applications des agents mobiles se heurte à des problèmes inhérents à la sécurité et à l'interopérabilité dans les différentes plateformes d'accueil hétérogènes. Les insuffisances des efforts pour la standardisation de ces plateformes, ont poussé les chercheurs à explorer d'autres horizons pour une meilleure prise en charge et mise en œuvre des applications à base d'agents mobiles.

D'autre part, très peu de travaux de recherches portent sur les méthodes et les outils pour l'analyse et la conception des systèmes d'agents mobiles.

Dans cette optique, nous avons introduit dans le deuxième chapitre, des éléments essentiels de modélisation UML qui consistent à étendre les différents diagrammes d'UML 2.0 pour la prise en charge de la mobilité. Une étude de cas d'un système de bourse électronique, basée sur notre contribution, a fait l'objet d'une publication dans un journal spécialisé(IJCSNS NOV 2009).

L'intérêt d'une telle contribution est de renforcer la robustesse des applications dans les systèmes distribués, en les modélisant avec des outils standards tel que UML. Le revers de la médaille est que UML est un langage de modélisation semi-formel; cet aspect réduit considérablement la possibilité de la vérification et la validation des systèmes modélisés.

La deuxième partie de notre thèse introduit dans le troisième chapitre, les réseaux de Petri de haut niveaux Nested-Net, comme une alternative qui permet de palier au insuffisances de l'approche semi-formelle d'UML pour la vérification et la validation de systèmes complexes et ce, en puisant de la genèse des outils formels, à savoir, les différentes variantes des réseaux de Petri.

Notre deuxième contribution présentée dans le cinquième chapitre a fait l'objet d'une communication dans une conférence IEEE spécialisée. Cette contribution propose une démarche automatisée basé sur la technique de transformation de modèle, présentée dans le quatrième chapitre en utilisant les grammaires de graphes. Il s'agit en fait, de transformer le diagramme d'états transitions de M-UML en tant que graphe, vers un réseau de Petri de haut

niveau et ce, en puisant du concept de la MétaModélisation. L'objectif final est de pouvoir réaliser des vérifications de diagrammes UML étendus.

Bien que nous avons atteint les objectifs fixés pour cette thèse, dans un travail futur, nous comptons transformer les diagrammes UML étendus de notre première contribution, vers les Nested-Nets, en utilisant toujours la technique de transformation de graphes avec éventuellement l'outil AToM^3.

Une perspective intéressante de ce travail, consiste en l'obtention des interprétations automatiques des résultats d'analyse des réseaux de Petri pour les modèles à agents mobiles. Par exemple, comment interpréter un interblocage par les réseaux de Petri de haut niveaux dans un modèle de diagramme de séquences.

D'autres perspectives sont aussi à l'horizon, il s'agit en fait, de réaliser une deuxième passerelle des réseaux de Petri vers les outils d'analyse des réseaux de Petri, tel que INA(Integrated Net Analyser) qui est un outil d'analyse de modèles réseaux de Petri. Ceci nous permettra de passer automatiquement des modèles à agents mobiles vers la spécification (textuelle) de INA éliminant ainsi les risques d'erreurs qui peuvent être causées par une transformation manuelle.

Références Bibliographiques

[Adl03] L. Adlèn, H. Hachicha, K. Ghedira, *A proposed Approach to Model and to Implement Mobile Agents*, SOIE, Institut Supérieur de Gestion de Tunis, Université de Tunis.2003.

[Anu97] A. Maria, *Introduction to Modelling and Simulation*, Proceedings of the 29th conference on Winter simulation, Pages: 7 – 13, Atlanta, Georgia, United States, 1997.

[Arido98] Y. Arido Y. D. B. Lange, *Agent Design Patterns*, Elements of Agent Application Design. *In* Proceeding ofAutonomous Agent '98, ACM Press.

[Aud07] L. 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.

[Bahri09] **M**.R. Bahri, R. Mokhtari, et A. Chaoui, *Modelling of mobile agent-based systems by UML2.0*, Conference international ACIT, Elhamamat, Tunisia 2009.

[Bauer01] B. Bauer, J. Muller, J. Odell, *Agent UML*, a formalism for specifying multiagent interaction, 22nd International Conference on Software Engineering (ICSE), Agent-Oriented Software Engineering, Springer, Berlin, 2001.

[Br05] M. Balaha, J. Rumbaugh, *Modélisation et conception orientées objet avec UML 2*, deuxième édition, Pearson Education, France, 2005.

[Bl-Mo98] D. B. Lange, M. Oshima, *Programming And Deploying Java Mobile Agents with Aglets*, 1998.

[Briot01] J. P. Briot, Y. Demazeau, *Principes et architecture des systèmes multi-agents* 11 octobre 2001 .Paris**.**

[BRJ01] G. Booch, James Rambaugh et Ivar Jacobson *: Le Guide de l'utilisateur UML*, deuxième édition, Eyrolles, 2001.

[BZS93] Bershad (N. Brian), Zekauskas (J. Matthew) et Sawdon (A. Wayne), *The Midway Distributed Shared Memory System*, In: Proceedings of the 38th IEEE International Computer Conference (COMPCON'93). IEEE, Février 1993.

[Car] E. Cariou, *Ingénierie des Modèles Transformation de modèles*, Université de Pau et des Pays de l'Adour Département Informatique Eric.Cariou@univ-pau.fr.

[Cubat05] C. Cubat thèse de doctorat « *Agents Mobiles Coopérants pour les Environnements Dynamiques*», Institut National Polytechnique de Toulouse 2005.

[Chr] C. Kroib, G. Zhang, *TOOL SUPPORTED MODELING OF MOBILE SYSTEMS,* Universit¨at M¨unchen, Germany.

[Claud01] C.Kaiser, *ANNEXE 2 LES RÉSEAUX DE PETRI,* Reproduit avec la permission de *Francis Cottet*, ENSMA décembre 2001.

[Cook94] S. Cook, J. Daniels, Designing Object Systems - Object-Oriented Modelling with Syntropy. Prentice-Hall, 1994.

[Czar03] K. Czarnecki, S. Helsen, *Classification of Model Transformation Approaches,*University of Waterloo, Canada.2003, czarnecki@acm.org, shelsen@computer.org.

[Drieu] B.Drieu, L'intelligence artificielle distribuée appliquée aux jeux d'équipe situés dans un milieu dynamique, l'exemple de RoboCup.

[Edg01] A. Edgardo C. 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] A. Edgardo C. Marcos, *MAM-UML: An UML Profile for the Modeling of Mobile-Agent Applications, ISISTAN Research Institute Facultad de Ciencias Exactas – UNICEN*

[Favre] J. Favre, J. Estublier, M.Blay-Fornarino, L'ingénierie dirigée par les modèles *au-delà du MDA* Hermes Science publications, Lavoisier.

[Ferber95] J. Ferber, Les systèmes multi-agents Vers une intelligence collective, Masson, 1995.

[FIPA] Foundation for Intelligent Physical Agents,1998, URL,(www.fipa.org).

[Frc] F.Cassez, O. 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

[Fpv98] A. Fuggetta, G.Picco, G.Vigna, *Understanding Code Mobility*. IEEE Transactions on Software Engineering. vol. 24. n5. mai 1998.

[Gaia] M. Wooldridge, N. R. Jennings, D. Kinny, *The Gaia Methodology for Agent-Oriented Analysis and Design*, Journal of Autonomous Agents and Multi-Agent Systems, Vol.3, No. 3, pp. 285-312.2000.

[Gg96] G. Gardarin O.Gardarin, *Le Client-Serveur*. Eyrolles, mars 1993.

[G.Kar04] 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.

[Guerra03] E. Guerra, J. de Lara, *A Framework for the Verification of UML Models. Examples using Petri Nets.* Ecole Politechnique Superieur, Ingenieríe de l' Informátique, Université Autónoma de Madrid.Spain 2003

[Hach] H. Hachicha, A.Loukil, K. Ghedira, *MAMT: an environment for modeling and implementing mobile agents,* ISIMS: Institut Supérieur d'Informatique et de Multimédia, Sfax hela.hachicha@fsegs.rnu.tn, INSAT: Institut National des Sciences Appliquées et de

Technologie, Tunis adlen.loukil@insat.rnu.tn: ENSI : Ecole Nationale des Sciences de l'informatique, Tunis khaled.ghedira@isg.rnu.tn

[Hall90] A. Hall, *Seven myths of formal methods*. IEEE Software, September 1990.

[Har] M. Haralambos, J.Odell, G.Manson, *Extending the Unified Modeling Language to Model Mobile Agents,* Department of Computer Science, University of Sheffield, England g.manson@dcs.shef.ac.uk, James Odell Associates, Ann Arbor, MI USA email@jamesodell.com

[Hin95] J. P. Bowen, M. G. Hinchey, *Seven more myths of formal methods*. IEEE Software, pp. 34–41,july 1995.

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

[Jade] F. Luigi Bellifemine, G. Caire, D. Greenwood, *Developing Multi*-Agent *Systems With* Jade: www.Amazon.fr.

[Kang] M. Kang, L. Wang, K. Taguchi, *Modelling Mobile Agent Applications in UML2.0 Activity Diagrams* School of Computing, Leeds Metropolitan University, Department of Computing, School of Informatics.

[Kas01] S. Kassem, E. Christo, *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, June 2003.

[Kas02] S. Kassem, E. Christo, A.Mourtada, M. Yahya, *A mobile-agent platform and a game application specifications using M-UML* The Electronic Library, Research Library pp 22-32, 2004.

[Kee] V. Kees, I. A. Lomazova, O. Oanea, A. Serebrenik, N. Sidorova, M. Voorhoeve, *Nested nets for adaptive systems,* Department of Mathematics and Computer Science Eindhoven University of Technology P.O. Box 513, 5600 MB Eindhoven, The Netherlands ſk.m.v.hee, o.i.oanea, a.serebrenik, n.sidorova, m.voorhoeveg@tue.nl, Program Systems Institute of Russian Academy of Science, Pereslavl-Zalessky, 152020, Russia irina@lomazova.polnet.botik.ru.

[Kim07] M. T. Kimour, *Le processus unifiés (UP, RUP et TUP),* cours de l'école doctoral pôle est 2007.

[Klein01] C. Klein, A. Rausch, M. Sihling, Z. Wen, *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.

[Koh07] M. Köhler, R. Langer, R. Lüde, D. Moldt, H. Rölke, R. 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, 2007. http://jasss.soc.surrey.ac.uk/10/1/3.html 31-Jan-2007.

[Koh 02] M. Köhler, B.Farwer, *Object Nets for Mobility,* Department for Informatics University of Hamburg, Application and Theory of Petri Nets,2002.

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

[Kurt2 97] J. Kurt, *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] M. Kusek, G. 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] P. Laforcade, V. Barré, B. 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.

[Lara01] J.de Lara, H. Vangheluwe, *Computer Aided Multi-Paradigm Modelling to Process Petri-Nets and Statecharts*, ETS Inform_atica Universidad Aut_onoma de Madrid Madrid Spain, Juan.Lara@ii.uam.es, School of Computer Science McGill University, Montr2al Quèbec, Canada hv@cs.mcgill.ca.

[Lbar04]L. Baresi, R. Hekel, *Tutorial Introduction to graph transformation. A software Engineering perspective*, Lecture Notes in Computer Science, Volume 3256/2004, pp 431-433, Springer Berlin, novembre 2004.

[Lakos06] C.Lakos, *A Petri Net View of Mobility (FORTE 2005)* © 2006, The University of Adelaide CNAM - Mars 2006.

[Mand99] M. Andries, G. Engels, A. Habel, B. Hoffmann, h. Kreowski, S. Kuske, D. Pump, A. Schürr, G. Taentzer, *Graph transformation for specification and programming*, Science of Computer programming, vol 34, NO°1, pages 1-54, Avril 1999.

[Mase03] A.Self,S. A. DeLoach, *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).

[MASIF] MASIF, The OMG Mobile Agent System Interoperability Facility,1997, URL www.masif.org .

[MDA05] H. Kadima, MDA,une conception orientée objet guide par les modèles. DUNOD 2005.

[Mer74] P. M. MERLIN, A *study of the recoverability of computing systems*, PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1974.

[Mg00] P. A. Muller, N. Gaertner, *Modélisation objet avec UML*, Deuxième édition, Eyrolles, 2000.

[M-Gaia04] M.Gaia et al, *Extending the Gaia Methodology to Model Mobile Agent Systems*. In the Sixth International Conference on Enterprise Information Systems Porto,Portugal, pp 14-17, April, ICEIS 2004.

[Morge] M. Morge, *Interaction dans les systèmes multi-agents,* Vers les systèmes multi-agents dialogiques.

[Muscutariu01] F. Muscutariu, M. P. Gervais, *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.

[Odel93]J. J. Odell, *Specifying structural constraints.* Journal of Object Oriented Programming, pp. 12–16, 1993.

[OMG03]: Object Management Group (OMG), *MDA Guide Version 1.0.1,* copyright 2003.

[OMG03a] Object Management Group*, Unified Modeling Language Specification*, Version 1.5, mars 2003.

[OMG04] Object Management Group*, Model Driven Architecture (MDA),*URL http://www.omg.org/mda,2004.

[Pablo] J. Pablo López Grao, J. Merseguer, J. Campos, *From UML Activity Diagrams To Stochastic Petri Nets*, Application To Software Performance Engineering, Département d' Informatique Ingenieríe de Systèmes Université de Zaragoza, Spain.

[Prakash6] N. Prakash, S. Srivastava, S. Sabharwal. *The Classification Framework for Model Transformation*, Journal of Computer Science 2 (2): 166-170, 2006.

[Python] Python home page: htpp://www.python.org

[Pic99] G.P. Picco, A.L. Murphy, G. Roman, *LIME: Linda Meets Mobility*. In proc. Of 21th Int. Conf. On Software Engineering ICSE, 1999.

[Perret97] S.Perret, Agents mobiles pour l'accès nomade à l'information répartie dans les réseaux de grande envergure, thèse de doctorat, Université Joseph Fourier - Grenoble I, 1997.

[Ram74] C. Ramchandani, *Analysis of asynchronous concurrent systems by timed Petri nets*, PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, Project MAC Report MAC-TR-120, 1974.

[Rea08] R. El Mansouri, E. Kerkouche, A. Chaoui, A Graphical Environment for Petri Nets INA Tool Based on Meta-Modelling and Graph Grammars, WASET.ORG 2008.

[Rou02] S. Rouvrais, *Utilisation d'Agents Mobiles pour la Construction de Services Distribués,* Thèse de Université de Rennes1, 2002.

[Rus06] S. Russsell, P. Norvig, *Intelligence artificielle*, Pearson Education, France, 2006.

[Rum91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object Oriented Modeling and Design*, Prentice Hall, 1991.

[KSaleh04] K. Saleh, C. Elmorre, *M-UML: An Extension to UML for the Modeling of Mobile Agent-Based Software Systems,* Journal of Information and Software Technology, Vol. 46, No. 4, pp. 219-227, 2004.

[Scor06] G. Scorletti, G. Binet, *Réseaux de Petri*, Université de Caen, France 20 juin 2006. Page web: http: //www.greyc.ensicaen.fr/EquipeAuto/Gerard S/mait_Petri.html.

[Simo] B. Simona, S.Donatelli j.Merseguer*, From UML Sequence Diagrams and Statecharts to analysable Petri Net models,* Dipartiment d'Informatique Université de Torino, Italy, Département d'Informatique e Ingenierie de Systèmes Université de Zaragoza, Spain.

[Sopena] É. Sopena, *Éléments de théorie des graphes*, Université de Bordeaux 1**.**

[Spy] J. M. Spivey, *The Z Notation*, A reference Manual, Prentice Hall, 1989.

[Tran05] V. Tran, V. Moraru, *Réseau de Petri,* Institut de la Francophonie pour l'Informatique Promotion 10 15 juillet 2005.

[Varro] D. Varro, A. 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.

[VIATRA]VIATRA, *Visual Automated Transformations for Formal Verification and Validation of UML Models*,
URL:http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/index.html

[Whi94] J. E. White, *Telescript technology,* The foundation for the electronic marketplace, White paper, General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040, 1994.

[Wool03] Jeennings N. R. and Wooldridge M. (2000), «Agent-Oriented Software Engineering» in Handbook of Technology (ed. J.Bradshaw)AAAI/MIT Press.

[Yann03] Y. Dantal, C. Haug, *Théorie des graphes Principes et programmation,* Soluscience 2003.

[Zhao] Y. Zhao, Y. Fan, X. Bai, Y. Wang, H. Cai, W. Ding*, Towards Formal Verification of UML Diagrams Based on Graph Transformation,* CIM Research Center, Department of Automation, Tsinghua University, Beijing China 100084. IBM China Research Lab, 4F, Haohai Building, 5th Shangdi Street, Beijing China 100085.

Webgraphie

[And] mailto:http://www.andromda.org

[ATOM3]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