REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIURE ET DE LA RECHERCHE SCIENTIFIQUE

Université de 08 Mai 45Guelma
Faculté des sciences et de l'Ingénierie

Département de l'informatique

Présenté pour l'obtention du diplôme de

Magister en Informatique

Option : Intelligence Artificielle et Imagerie

Titre du Mémoire

==========

Modélisation d'un système d'agent mobile par les ECATNets

==========

Présenté par :
Zelit Saoussan

Devant le Jury :

Président :	Pr Seridi Hamid	Université de Guelma
Rapporteur :	Dr Chaoui Allaoua	Université Mentouri de Constantine
Examinateur :	Dr Kholadi Mohamed Khair Eldine	Université Mentouri de Constantine
	Dr Chikhi Salim	Université Mentouri de Constantine

2008

Remerciement

En premier lieu, je tiens à présenter mes sincères remerciements à mon encadreur Mr Allaoua Chaoui pour m'avoir aider et m'encouragé vivement à entreprendre ce travail avec une extrême bienveillance et autant de gentillesse, et grâce à ses conseils judicieux je tiens à exprimer ici mon profond respect et ma très grande reconnaissance.

Je tiens aussi à remercier chaleureusement mon mari Mr Abdeli Djallel pour son aide précieuse, sa patience, et sa compréhension durant la réalisation de ce travail.

Je tiens aussi à remercier les membres de jury pour avoir accepté de juger ce modeste travail.

En terminant, toutes les personnes qui m'a aidé, orienté et ont participé de près ou de loin à la concrétisation de ce modeste travail trouvent dans ce mémoire l'expression de mes forts remerciements et ma reconnaissance la plus sincère.

Dédicace

Avec un grand hommage je dédie ce modeste travail en reconnaissance à tous qui me sont chères :

A la mémoire de mes défunts très chère père et ma chère sœur Linda, je prie Dieu tout puissant et miséricordieux de les accueillir dans son vaste Paradis.

A la l'être la plus cher au monde, l'effective, la tendre: maman, et mon très cher père pour tous les sacrifices qu'ils sont consentis à mon éducation et à ma formation de long de ma vie.

A mon mari Djallel, et ma petite Raghad.

A toutes ma famille mes chers frères, mes chères sœurs, mes beaux frères et mes belles sœurs.

A ma belle famille et mon âme sœur mon amie Sana et toute sa famille.

TABLE DES MATIÈRES

Chapitre II: MODELISATION DES AGENTS MOBILES PAR LES RESEAUX DE PETRI

Chapitre III: LES ECATNets

PARTIE II : MODELISATION DES AGENTS MOBILES PAR LES ECATNets

Chapitre I : Description et motivation

Chapitre II: Les Firewalls et NetFilter

Chapitre II: Etude de Cas

LISTE DES FIGURES

RESUME

Dans ces dernières années, on assiste à l'accroissement spectaculaire des réseaux, l'émergence des systèmes distribués et d'Internet en particulier. Les systèmes d'agents et en particulier les systèmes d'agents mobiles, qui sortent à peine des laboratoires, montrent des potentialités intéressantes par leur flexibilité, adaptations et réduction de la charge des réseaux à grande échelle. Les agents mobiles sont une forte abstraction pour la conception des systèmes complexes répartis. Ils peuvent exhiber la majorité des attributs des agents comme l'intelligence, l'autonomie, l'adaptabilité, et la sociabilité et ils peuvent aussi se déplacer vers d'autres ordinateurs pour bénéficier de la performance des interactions locales**.**

Dans ce mémoire on a présenté une approche formelle et intuitive pour la modélisation du comportement des agents mobiles basée sur les ECATNets, qui sont une catégorie de réseau de Petri de haut niveau. Les ECATNets exploitent les forces des types abstraits algébriques qui permettent de décrire abstraitement les données, et leur sémantique est définie en termes de la logique de réécriture qui offre une base solide et rigoureuse pour toute démarche de vérification. Pour bien illustrer les apports des ECATNets dans la modélisation des agents mobiles, on a démontré l'approche proposée par la modélisation d'un système d'administration à distance des Firewalls basée sur le paradigme d'agents mobiles.

ABSTRACT

For the last few years, networks, distributed systems particularly Internet have increased incredibly. Many efforts are put on new technologies, still in laboratories, of agents and mobile agents. These systems show interesting capabilities considering flexibility and reduction of network load, especially in large networks. Mobile agents are a powerful abstraction for conceptualizing large-scale distributed network systems. They can exhibit the same attributes of agents like: intelligence, autonomy, adaptability, sociability, they can also move towards other computers to profit from the performance of the local interactions**.**

In this work we present a formal and intuitive approach for the modeling behavior of mobile agents based on ECATNets, which are a category of algebraic Petri nets (APNs) based on a safe combination of algebraic abstract types and high level Petri nets. The semantic of ECATNets is defined in terms of rewriting logic, allowing us to built models by formal reasoning. As Petri nets, ECATNets provide a quickly understood formalism due to their simple construction and graphical depiction. Moreover, ECATNets have a strong theory and

development tools based on powerful logic with sound and complete semantic. We use this approach for modeling a system of remote administration of firewalls based on the mobile agent paradigm.

ملخص

تشهد السنوات الأخيرة نموا مبهرا لشبكات المعلوماتية و بروزا ملحوظا للأنظمة المتفرعة و على الأخص الإنترنت. برزت في الآونة الأخيرة تقنية جديدة و التي لا تزال قيد التطوير تتمثل في استعمال العملاء(Agents) و بالأخص العملاء المتحركين(Agents mobiles), هذه التقنية أبرزت قدرات كبيرة من حيث مرونتها, قدرتها على التأقلم و تقليلها من المعلومات المتبادلة عبر الشبكات.

يمنح استعمال العملاء المتحركين تجريدا جيدا لتصميم الأنظمة المعقدة المتفرعة, فلديهم أغلبية خصائص العملاء مثل الذكاء, الاستقلالية, التأقلم و التصرف كمجتمعات... بالإضافة إلى خاصية التنقل من جهاز إلى آخر من أجل الاستفادة من قدرات هذا الأخير.

حاولنا في هذه المذكرة تمثيل تصرفات العملاء المتحركين بطريقة مجردة تستند على نوع من شبكات Petri عالية المستوى تسمى ECATNets, هذه الأخيرة تستغل إيجابيات و قوة الأنواع المجردة للمعطيات(TAD) التي تسمح بالوصف التجريدي للمعطيات, و كذلك قدرة la logique de réécriture على وصف كل الأوضاع الممكنة في العالم الحقيقي و التي تمنح قاعدة متينة لفحص الأنظمة.

لتوضيح إسهام ECATNets في تمثيل الأنظمة التي تعتمد على تقنية العملاء المتحركين, قمنا بتقديم مثال خاص بأنظمة الإدارة عن بعد لجدران النار (Firewalls).

INTRODUCTION

Dans ces dernières années, on assiste à l'accroissement spectaculaire des réseaux, l'émergence des systèmes distribués asynchrones et d'Internet en particulier, tant en quantité de données présentées et échangées, qu'en nombre d'utilisateurs ou en étendue géographique. Cette expansion s'est appuyée sur la technologie client/serveur qui a, jusqu'à présent, réussi à s'adapter et à transporter et traiter un volume de données toujours plus grand. Cependant, cette technologie a aussi montré une faiblesse pareille : sa grande centralisation conduit à un problème d'évolutivité, de manque de personnalisation et d'un manque de prise en compte de la topologie du monde réel et virtuel. Ce dernier problème n'est toutefois pas attribuable uniquement à la technologie client/serveur, mais aussi à la création d'Internet. C'est pourquoi on s'oriente vers des nouvelles technologies, qui sortent à peine des laboratoires, les systèmes d'agents et en particulier les systèmes d'agents mobiles. Malgré des défauts certains et le manque d'applications où ils marquent une réelle différence avec les technologies existantes, ces systèmes montrent des potentialités intéressantes pour la flexibilité, l'adaptation et la réduction de la charge des réseaux, en particulier des grands réseaux.

Les agents mobiles sont élaborés du concept d'agents autonomes, ils sont une forte abstraction pour la conception des systèmes asynchrones et répartis de grande échelle. Le concept d'agents mobiles peut supporter différents types d'applications tels que le commerce électronique, la gestion de réseaux, les wireless, la gestion du workflow, les services de télécommunication…etc.

Un système à base d'agents mobiles est généralement constitué d'agents stationnaires et d'agents mobiles chargés à représenter ses commettants (agent ou humain) et qui collaborent pour accomplir des tâches. L'agent stationnaire est immobile et responsable d'offrir des services de base et des ressources que les agents mobiles peuvent les utiliser pour implanter leurs propres services. L'agent mobile peut exhiber la majorité des attributs des agents comme l'intelligence, l'autonomie, l'adaptabilité, et la sociabilité. Il a aussi la capacité de migrer vers d'autres ordinateurs pour bénéficier de la performance des interactions locales. Le résultat produit par un agent mobile peut être transmis ou transporté avec lui s'il retourne à son lieu d'origine.

Bien que, les agents mobiles soient élaborés pour palier à des problèmes des technologies obsolètes, il reste très intéressant de pouvoir trouver une approche formelle

permet de modéliser le comportement des agents mobile dans le but de bien comprendre, analyser et valider les systèmes basés sur ce paradigme.

Problématique et Contribution

Les systèmes informatiques sont devenus de plus en plus complexes et répartis et le paradigme d'agents et agents mobiles est fortement utilisé pour la mise en œuvre de ces systèmes. Beaucoup de travaux s'orientent vers l'obtention d'une approche formelle permet de modéliser ces systèmes d'agent en particulier les agents mobiles.

L'utilisation des réseaux de Petri pour représenter des systèmes d'agents est une idée intéressante dans le sens où la symbolique assez simple des réseaux de Petri permet une compréhension rapide de l'architecture globale, ainsi la capacité de décrire les comportements parallèles et asynchrones. Cependant, la représentation en réseau de Petri peut paraître aujourd'hui obsolète sous certains aspects. Si on se base sur la définition d'un agent, on peut voir qu'un agent doit avoir les propriétés suivantes: autonomie, réactivité, initiative, raisonnement, apprentissage et l'habilité sociale plus la capacité de déplacement pour les agents mobiles. Certaines de ces caractéristiques et leurs incidences dans la structure des agents ne pourraient pas être intuitivement traduites par des réseaux de Petri. L'utilisation donc de réseaux de Petri comme base pour le développement et la représentation de systèmes d'agents peut encore être envisagée pour des systèmes simples dont les agents travaillent dans un environnement statique (non réel) ; pour des systèmes plus complexes et distribués, l'utilisation des réseaux de Petri ordinaires est à éviter. Des représentations plus puissantes et tout aussi intuitives existent de nos jours.

Après l'étude de plusieurs systèmes complexes répartis qui sont basés sur le paradigme d'agents mobiles, on a trouvé qu'il existe une panoplie de modélisation de ces systèmes qui utilisent le modèle de réseaux de Petri plus sophistiqué tel que les réseaux de Petri hiérarchique, les réseaux de Petri colorés, les hypernets…. Chacune de ces modélisations a des avantages et des inconvénients. Dans ce travail on essaye de présenter une approche formelle pour la modélisation des systèmes d'agents mobiles en utilisant les ECATNets.

Les ECATNets sont un modèle formel permet la spécification et la validation des systèmes complexes, ils une catégorie des réseaux de Petri de haut niveau. Ils combinent la force des réseaux de Petri, les types abstraits données, et la logique de réécriture. La représentation graphique donne une puissance d'expression aux réseaux de Petri, ils ont aussi la capacité d'exprimer la concurrence, le parallélisme, et l'aspect dynamique des systèmes. La sémantique des ECATNets est définie en terme de la logique de réécriture qui est une logique

saine, complète, permet de raisonner de manière concrète, elle unifie plusieurs modèles formels qui expriment la concurrence, et peut expliquer tout situation du monde réel. Les types abstraits algébriques sont utilisés pour leur puissance de décrire abstraitement les données. Ils permettent une compréhension rapide à la structure de données et ils sont faciles à implémenter.

Suite aux inconvénients des réseaux de Petri ordinaires et même des réseaux de Petri de haut niveau, et suite aux avantages des ECATNets dans la modélisation des systèmes complexe répartie, on a proposé dans ce mémoire une approche de modélisation formelle basée sur les ECATNets pour modéliser le comportement des agents mobiles dans un environnement distribué, et on a modélisé un système réel, le système d'administration et de configuration à distance des Firewalls en utilisant les agents mobiles.

Plan du mémoire

Le mémoire comprend deux parties, La première partie constitue l'état de l'art elle est composée de trois chapitres. Le chapitre I fait le point sur le paradigme d'agent mobile et il présente quelques domaines d'application et l'apport de ce paradigme pour les systèmes complexes et répartis. Le chapitre II donne une brève présentation des modélisations des systèmes basés sur le paradigme d'agents mobiles par plusieurs types de réseaux de Petri. Et on termine cette partie par un chapitre qui est consacré à exposer Les réseaux de Petri ECATNets, dans ce chapitre on a introduit la logique de réécriture et a présenté ses concepts de base ainsi que son langage Maude. Nous donnons aussi une présentation générale des ECATNets et de leur description dans la logique de réécriture en plus de son implémentation des ECATNets dans Maude.

Dans la deuxième partie on a introduit notre modélisation des systèmes à base d'agents mobiles par les ECATNets, et on a présenté une étude de cas dans laquelle on a donné certaines notions de base des Firewalls et le NetFilter. Ensuite, on a définit un système d'administration de Firewall NetFilter à distance en utilisant les agents mobiles, et on termine par une modélisation de ce système par les ECATNets. Finalement, ce mémoire se termine par une conclusion générale.

Partie I :

ETAT DE L'ART

Chapitre I :

Les Agents Mobiles

- Introduction
- Historique
- Définition
- Agent Mobile
- Avantages et inconvénients des agents mobiles
- Applications des agents mobiles
- Performances et perspectives

INTRODUCTION

Les dernières décennies ont été marquées par une révolution dans le domaine de l'informatique distribuée, des télétraitements et de l'émergence des réseaux d'informations à grande échelle. Les logiciels deviennent en effet de plus en plus complexes et intègrent de manière intime des techniques venant du génie logiciel, de l'intelligence artificielle, des bases de données et de la programmation répartie. Les agents mobiles sur le réseau est une technologie prometteuse identifiée comme la future plate-forme de base pour une architecture de services électroniques et qui pourra s'échapper des contraintes d'hétérogénéité des plates-formes cibles en raison de l'adaptabilité qui caractérise les agents mobiles.

1 HISTORIQUE

La plupart des applications reparties utilisent le paradigme Client-Serveur, dans lequel le client et le serveur se communiquent en utilisant une technique ***Message Passing*** ou ***Remote Procedure Calls*** (RPC). RPC est synchrone : le client suspend son exécution après avoir envoyé sa requête au serveur, et attend les résultats de l'appel. Dans le RPC, les données sont transférées entre le client et le serveur, dans les deux directions. Une autre approche appelée ***Code on Demand*** (COD) permet au client d'accéder aux ressources et aux données dont il a besoin, mais il n'y a pas de code pour manipuler les ressources disponibles chez le client. Une autre architecture appelée ***Remote Evaluation*** (REV) permet, au lieu d'invoquer une procédure distante, le client envoie son code au serveur, en demandant de ce dernier de l'exécuter avec ses données et ses ressources pour retourner les résultats. Plus récemment, le concept de message actif a été introduit. Un message actif peut migrer d'un noeud à un autre, en transportant le code du programme pour que celui-ci puisse être exécuté par les noeuds. Une approche plus générale est l'***Agent mobile*** qui encapsule des données avec leurs opérations pour migrer d'un noeud à un autre.

2 DEFINITION

Les agents mobiles sont au carrefour de deux concepts plus anciens : le concept des agents utilisé dans les systèmes multi-agents et le concept de mobilité du code. Dans ce chapitre nous allons définir ces deux concepts en détailles, ce qui déterminent les caractéristiques des agents mobiles.

2.1 L'agent :

Dans le domaine informatique, le terme « agent » a plusieurs définitions et il n'y a pas de définition commune. Cependant, l'agent est généralement défini comme une entité logicielle autonome ayant une activité propre et agissant pour le compte d'une personne ou d'une organisation**.** En IA (Intelligence Artificielle), un agent est un programme intelligent et autonome. Dans le domaine de la gestion de réseaux, un agent est un programme autonome qui réagit à des stimulus externes en fonction de règles préétablies. Le concept d'agent est aussi très semblable au concept d'acteur utilisé dans le domaine des systèmes temps réel**.** Et finalement, dans le domaine des systèmes répartis, un agent est un programme mobile.

2.2 Propriétés des agents :

De ces définitions nous pouvons identifier les propriétés suivantes qui caractérisent les agents :

• Autonomie : Les agents peuvent agir sans l'intervention directe de ces commettants et ont un certain contrôle de leurs actions et état interne.

• Réactivité : Les agents peuvent percevoir leur environnement, qui peut inclure le monde physique, un usager derrière une interface graphique, des applications sur le réseau, ou d'autres agents, et répondre à des changements qui se produisent dans celui-ci.

• Initiative : Le comportement des agents est déterminé par les buts qu'ils poursuivent et par conséquent ils peuvent produire des actions qui ne sont pas seulement des réponses à leur environnement.

• Habilité sociale : Pour satisfaire ses buts un agent peut demander la collaboration d'autres agents à qui il délègue ou avec lesquels il partage la réalisation de tâches. Pour cela il a besoin de se coordonner, négocier et interagir.

• Raisonnement : Un agent peut décider quel but poursuivre ou à quel événement réagir, comment agir pour accomplir un but, ou suspendre ou abandonner un but pour se dédier à un autre.

• Apprentissage : L'agent peut s'adapter progressivement à des changements dans des environnements dynamiques grâce à des techniques d'apprentissage.

2.3 La mobilité :

Dans des applications déterminées il peut être intéressant de permettre aux agents de migrer d'un noeud à un autre dans un réseau tout en préservant leur état lors de sauts entre

noeuds. L'idée est issue de techniques de migration des processus qui visent à transférer un processus d'une machine vers une autre dans le réseau en tenant compte que le processus est une abstraction du code d'un programme, de données et de son état d'exécution. Ces techniques sont implantées principalement au niveau du système d'exploitation qui doit être complexe et exclusif. Leur difficulté principale est de pouvoir transférer l'état du processus interne au système, ainsi que ses ressources. Pour cela la migration de processus n'a pas connu de succès mais elle a introduit les notions de code mobile et mobilité.

Le paradigme d'agents mobiles donc hérite la technique de migration de processus et permet d'introduire un fonctionnement asynchrone, en proposant la migration d'activité qui supprime la contrainte de la connexion constante qui n'est évidente ni pour les réseaux de grande envergure ni pour les stations nomades.

On distingue deux types de migration : migration ***proactive*** ou ***autonome*** où l'agent lui-même prend la décision de migration (quant et où il se déplace), l'agent demande explicitement d'échanger leur environnement d'exécution via une instruction comme *jump* , *go* ou *go to*, par contre dans la migration ***passive*** ou ***réactive*** l'environnement d'exécution qui prend la décision du déplacement ; cette décision peut être prise en se basant sur certaines exigences du système (load balancing ou la demande faite par un autre agent [9].

La migration peut être ***forte*** ou ***faible***. La ***migration forte*** permet à un agent de se déplacer quelque soit l'état d'exécution dans lequel il se trouve et de reprendre son exécution après la migration exactement là où elle était avant de ce déplacement ; Mais la ***migration faible*** ne fait que transférer avec l'agent son code et ses données, ce qui nécessite des moments privilégiés (explicites) dans le code de l'agent (point d'arrêt) pour pouvoir le lancer [8].

La migration forte, bien que beaucoup plus exigeante à implanter (plate-forme) que la migration faible, n'en est pas moins indispensable pour toutes les applications pour lesquelles les notions de fiabilité et de tolérance aux pannes sont primordiales

3 AGENT MOBILE :

Un agent mobile donc est un programme exécutable qui peut migrer d'une machine vers une autre sur le réseau (environnement hétérogène) en fonction de ses besoins pour réaliser une tache complexe. Il présente les majeurs propriétés des agents (l'intelligence, l'autonomie, l'adaptabilité, et le comportement social) ce qui signifie qu'il est capable de prévenir et résoudre les problèmes rencontrés sur le réseau durant son parcourt.

4 AVANTAGES ET INCONVENIENTS DES AGENTS MOBILES

4.1 Avantages :

Les agents mobiles offrent une panoplie d'avantages qui justifient leur utilisation, on en retiendra les suivantes :

- Extension des capacités de traitement en surmontant les limitations d'un petit ordinateur. Il suffit d'envoyer un agent pour exécuter une tâche sur un serveur ayant de plus grandes capacités de calcul, de mémoire et de communication [5].

- Personnalisation : il est plus facile à un utilisateur de personnaliser son agent qu'un programme résidant sur un serveur distant [5].

- Survivabilité : alors qu'un programme classique est lié à une machine, un agent mobile peut se déplacer pour éviter une erreur matérielle ou logicielle, ou tout simplement un arrêt de la machine [5].

- S'exécuter de manière asynchrone et autonome : Les agents mobiles s'exécutent indépendamment de leur plate-forme d'origine, c'est à dire un agent peut continuer à parcourir le réseau même sans interaction avec l'utilisateur, grâce à son autonomie, cela permet au client de déconnecter sa machine pendant le travail de l'agent [9, 8].

- Réduction de la charge du réseau : en se rapprochant des données sur lesquelles il veut travailler, un agent mobile peut permettre de diminuer le nombre de messages de communication, et par là même, la charge du réseau [9].

- Indépendance par rapport au système d'exploitation et hétérogénéité : la multiplicité des systèmes augmente l'incompatibilité des langages de commandes et des formats de données. Les agents peuvent résoudre ce problème sous contrainte que les primitives du langage de l'agent soit exécutable sur tout système [5].

- Facilité de développement: le concept d'agent devrait (à terme) permettre de développer des applications mobiles plus facilement en masquant à l'utilisateur tes problèmes liés au transport dans les réseaux, et même certains choix d'optimisation faisables par l'agent, ainsi qu'une analogie avec le monde réel. Un agent mobile et intelligent pourrait être vu comme un utilisateur humain habituel [5].

- Surmonter le temps de latence sur le réseau : Les agents mobiles sont envoyés dans la zone d'activité, afin d'entreprendre des actions sur place. Cela leur permet de répondre en temps réel à des changements dans leur environnement [10, 8].

- S'adapter dynamiquement aux environnements d'exécution : Les agents mobiles ont la capacité de s'adapter dynamiquement aux changements de leur environnement [9].

4.2 Désavantages

- Manque d'applications où les agents mobiles apportent un avantage certain [5].

- Sécurité : c'est une difficulté majeure pour le développement de systèmes d'agents mobiles, comme système fonctionnant en environnement ouvert **(i.e.** Internet) et non fiable (réseaux mobiles) [5].

- Manque d'infrastructure et de standards : les agents ont besoin du support fixe d'une "plate-forme"**.** Or, aucun standard n'est réellement appliqué et leur conception varie encore beaucoup d'un système à l'autre. De même, les difficultés non résolues empêchent la formation d'une infrastructure suffisamment éprouvée pour permettre le développement d'applications économiquement intéressantes [5].

- De point de vue 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 a la méthode classique (client/serveur) bien mieux maîtrisé [10].

- Lors la phase de développement la complexité de mise au point des programmes qui constitue une partie critique du processus de développement pose un problème important. La plupart des environnements de programmation possèdent des outils permettant de suivre les différents éléments d'une application durant son exécution afin d'en trouver les erreurs. Cependant, ce genre d'outil s'utilise parfaitement avec des éléments statiques mais est difficilement applicable avec les éléments déplaçant [10].

- Le déboguage : la difficulté de mettre en place une vérification des applications à base d'agents mobiles. La technique la plus utilisée encore de nos jours, bien qu'elle ne soit pas totalement satisfaisante, est sans aucun doute le test. Dans cette méthode, on va mettre à l'épreuve une application en lui injectant des données dans ses différents points d'entrée dans le but de simuler le comportement utilisateur et de recouvrir une plage de situations la plus large possible [10]. Pour vérifier les applications construites sur des agents mobiles, il faut utiliser une méthode permettant de garantir le comportement et les interactions des agents avant leur déploiement.

5 APPLICATIONS DES AGENTS MOBILES :

Les domaines d'application des agents mobiles sont à peu près les mêmes que ceux où les agents statiques sont actuellement appliqués. On peut citer : la recherche et filtrage d'informations, le commerce électronique, l'administration des réseaux, et la télécommunication.

5.1 Recherche et filtrage d'informations

C'est le plus grand champ d'application et d'expérimentation des agents. Les systèmes qui utilisent des agents statiques pourraient être avantageusement remplacés ou secondés par des agents mobiles.

Les agents, et spécialement les agents mobiles, sont adaptés pour agir comme des logiciels qui naviguent continuellement sur la toile pour trouver de nouvelles informations. Cette technologie, qui utilise des techniques de data-mining, est déjà bien étudiée et possède ses standards [5].

5.2 Commerce électronique

Les agents mobiles peuvent être utilisés pour le commerce électronique de plusieurs façons. Ils peuvent donner aux utilisateurs un accès personnalisé aux informations données en ligne. Nomad *(2000)* par exemple, est système d'agents mobiles, utilisé dans un site d'enchères, eAuctionHouse, à l'université de Washington. Tabican (site Aglets, détails en Japonais) est un marché virtuel pour des billets d'avions et des tours (air + hôtel) où des milliers d'agents mobiles de clients et vendeurs peuvent se rencontrer et trouver le meilleur prix pour le client et le vendeur à la fois, sans leur faire perdre de temps [5].

5.3 L'administration de réseaux

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

L'administration de réseau est par nature asynchrone et répartie. De plus, il est souvent important d'avoir une vue locale du système pour pouvoir déterminer les causes et les conséquences d'un problème. L'administrateur doit alors se déplacer vers des machines lointaines qui nécessitent des taches de maintenance ou de mise à jour [5]. L'installation et la

maintenance des logiciels deviennent difficiles avec l'augmentation du nombre de machines. Les agents mobiles sont ici adéquats pour voyager dans le réseau et effectuer des tâches périodiques

5.4 Agents mobiles en télécommunications

La fonctionnalité de mobilité prend toute son importance dans le domaine des télécommunications. Les agents mobiles peuvent y être utilisés pour couvrir toutes les couches des protocoles de communication, de la maintenance de réseau, jusqu'aux applications mobiles, suivant l'usager dans ses déplacements. A titre d'exemple dans le système SPIN un *Personal Communicator Agent* est un agent mobile chargé de délivrer un message au destinataire, quel que soit son appareil (téléavertisseur, téléphone, ordinateur, portable ou téléphone sans fil) [5]. Il doit pouvoir recevoir les messages et les conduire sans interruption dans des réseaux hétérogènes à I'utilisateur. Par exemple, si le seul moyen de délivrer un message urgent à un utilisateur est un téléphone sans fil, l'agent personnel doit convertir le message textuel en message vocal.

NetChaser (di Stefano et C. Santoro, 2000) est un ensemble d'assistants personnels développé à l'université de Catania, en utilisant leur propre système d'agents mobiles [5]. La mobilité permet à ces agents de suivre l'utilisateur même quand il change de machine.

6 PERFORMANCES ET PERSPECTIVES

Les systèmes d'agents mobiles permettent le développement rapide d'applications qui utilisent beaucoup la mobilité et peu d'algorithmes complexes d'intelligence artificielle. Ces systèmes montrent de bonnes performances sur des réseaux fermés et sûrs, mais ils ont encore besoin de plus d'autonomie et d'intelligence pour aborder des réseaux plus changeants et risqués. Il y a une contradiction flagrante entre les capacités actuelles des agents mobiles et c'est pourquoi ils sont prévus et adaptés [5]. C'est pourquoi il est important d'aborder des problèmes tels que la sécurité, et le routage.

L'intégration des algorithmes complexes d'IA dans les agents mobiles augmente la taille de leur code et les rend moins efficaces pour des tâches simples, mais cela leur permet d'effectuer des tâches plus complexes dans des environnements plus ouverts et hasardeux. L'important est de ne pas surestimer les agents mobiles et de garder un bon équilibre entre la difficulté de la tâche à accomplir et la quantité d'IA inclue, lors du développement d'un système [9].

.

Les tests et évaluations effectués ont pu montrer la supériorité d'un système client/serveur ou d'un système d'agents mobiles, suivant la situation. En fait, les agents mobiles semblent surpasser les solutions classiques pour la charge du réseau ou même la vitesse d'exécution, mais les vrais tests vont à peine commencer, avec les environnements réalistes comme SPIN, et des applications plus complexes [5].

Les agents mobiles sont l'aboutissement de l'évolution des concepts de mobilité, mais ils utilisent aussi des techniques d'intelligence artificielle quelle doit prendre de plus en plus d'importance pour pouvoir exploiter toutes les possibilités que l'on peut attendre des agents mobiles.

Chapitre II :

Modélisation des systèmes d'agents mobiles par les réseaux de Petri

---

- Introduction
- Modélisation du comportement des agents mobiles par les OPN (Object Petri Nets)
- Modélisation du comportement des agents mobiles par les réseaux de Petri Hiérarchiques
- Modélisation du comportement des agents mobiles par les Hypernets
- Modélisation du comportement des agents mobiles par les PrTnets
- Conclusion

INTRODUCTION

Dans le monde informatique les systèmes sont devenus de plus en plus complexes et réparties, la manipulation et l'application de ces systèmes seront aussi difficiles et délicates. C'est pourquoi beaucoup de recherches s'orientent vers les méthodes formelles de modélisation dans le but de bien comprendre et maintenir ces systèmes. Les réseaux de Petri sont une méthode formelle permet la modélisation des systèmes complexes et répartis tel que les systèmes d'agents mobiles, et actuellement, plusieurs types de réseaux de Petri de haut niveau ont été inventé pour répondre aux manques de réseaux de Petri ordinaire. Dans ce chapitre on va essayer de présenter quelques modélisations des différents systèmes qui utilisent le paradigme d'agent mobiles. Ces modélisations sont basées principalement sur les réseaux de Petri de haut niveau.

1. MODELISATION DU COMPORTEMENT DES AGENTS MOBILES PAR LES OPN (OBJECT PETRI NETS)

Dans [7] l'agent est considéré comme une entité abstraite avec un comportement interactif. Pour cela la modélisation d'un système d'agents mobiles doit être vue de trois niveaux. Le premier est le niveau agent qui décrit le comportement individuel de l'agent. Le second est le niveau hôte où l'auteur décrit les interactions entre agents et la manière de communication entre eux dans une machine. Et finalement, le niveau système qui est considéré comme une vue globale de tout le système où le comportement de migration d'une machine vers une autre est décrit.

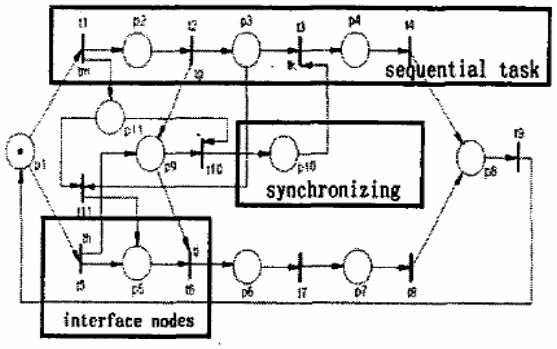
Niveau agent:

Un agent est une entité software autonome créée par l'utilisateur ou par un autre agent et identifié par certains attributs. Il peut créer des copies de lui-même constituant un clone de l'agent original (l'agent parent). Les agents peuvent également mettre en application d'autres taches telles que détruire un autre agent (tuant un processus) et être suspendus temporairement (sommeil) ou bien suspendus à long terme.

Le comportement individuel d'un agent est décrit par un réseau de Petri. Ce comportement est un ensemble d'actions (sous tâches) peuvent êtres exécutées d'une manière séquentielle ou parallèle. Elles sont définies sous forme de fonction comme un appel d'une méthode ou un envoi de message d'un autre agent, chaque fonction peut retourner un résultat et avoir un nombre de paramètres d'entrée. Ces actions peuvent s'exécuter en concurrence et

réalisent un comportement de haut niveau, dans se cas elles sont coordonnées par un événement interne qui est spécifié par une transition entre états, chaque transition est attachée à certaines conditions.

Dans ce cas un agent mobile est modélisé par un réseau de Petri nommé *Generic Agent Template (GAT)* qui est constitué de trois composants principales : l'état, le comportement et l'interface. Le comportement de GAT est décrit par un ensemble d'actions qui sont modélisées par le réseau de Petri. L'interface est modélisée par une structure spécifique de réseau de Petri appelée *Connector PN (C-PN)*, et l'état concret d'un agent est expliqué par le marquage de GAT qui est instancié pour un ensemble spécifique de sous tâches. La figure suivante montre un GAT abstrait avec certains types de sous tâches (actions) et des synchronisations.



FigureI.2.1 : La structure de GAT

Niveau hôte

Il existe un ensemble d'agents dans chaque hôte, cet ensemble est modélisé par un réseau de Petri qui implique plusieurs sous structures (GATs) interconnectées par les nœuds d'interface de communication. *Connector PN (C-PN)* permet aux agents d'interagir entre eux par l'échange des messages à l'intérieur de l'hôte.

Un C-PN est le tuple (P,T, V, F, μ_o, R) où (P,T,V,F) est une structure de réseau de Petri , μ_o est l'état initial du réseau de Petri, et R est la règle de connexion de GATs, de cette définition l'interaction entre agents a lieu à travers les structures de réseau de Petri au lieu des arcs spécifiques entre nœuds basés sur certaines règles prédéfinies.

Le schéma suivant représente l'interaction entre deux agents simples, chacun se compose d'une tâche séquentielle.

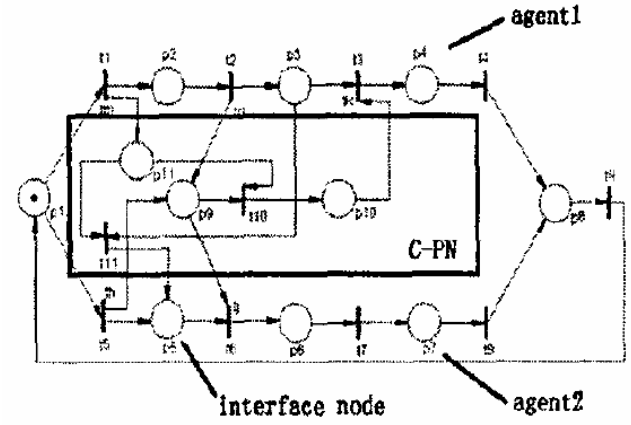


Figure I.2.2 : modèle de deux agents interagissent entre eux a travers C-PN

Niveau système :

Ce niveau est le plus haut niveau de cette approche de modélisation, il décrit les interactions entres machines a travers les actions de migration des agents entre machines ou l'envoi de messages de communication.

La migration signifie qu'un agent se déplace physiquement d'une machine à une autre et cette mobilité est modélisée par des transitions extérieures le réseau de Petri en considérant l'agent un jeton dans le réseau de Petri représentant le système entier

Dans ce qui suit, on considère le réseau de Petri modélisant le scénario du traitement d'une simple tache dans trois machines différentes.

L'agent commence son voyage à partir de son hôte et visite les unités de traitement des différentes machines. Une fois les unités ont complété ses traitements, l'agent retourne à sa machine pour traiter les résultats, et puis les annoncer (par exemple, des affichages) à toutes les unités de traitement**.** Par exemple la transition T1 est une transition de migration, T7 une transition signifie la terminaison du traitement, T3 transition de synchronisation…etc.

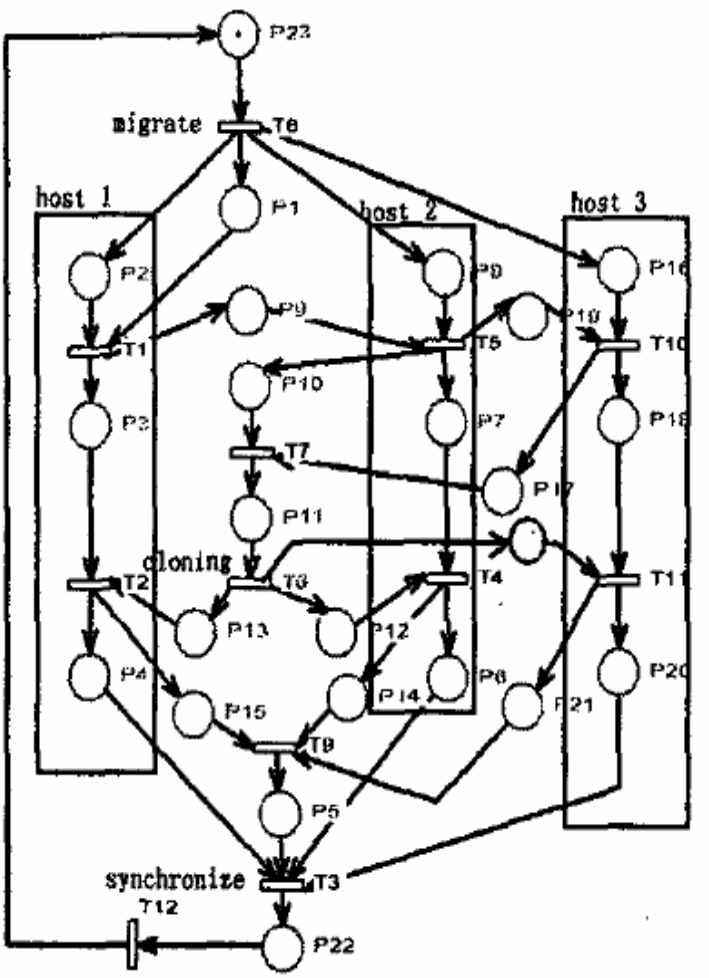


Figure I.2.3 : Le comportement mobile d'un agent

2. MODELISATION DU COMPORTEMENT DES AGENTS MOBILES PAR LES RESEAUX DE PETRI HIERARCHIQUE

Dans ce travail l'auteur [17] utilise pratiquement le même modèle de réseau de Petri hiérarchique pour une modélisation d'un système appelé MUD. Ce dernier est un système qui offre un espace virtuel partagé entre plusieurs utilisateurs.

Le système MUD se compose d'une base données unique, un serveur central, plusieurs serveurs de régions chacun manipule certains région dans le monde virtuel, et plusieurs utilisateurs qui utilisent ce qui est appelé client pour se connecter au serveur.

Les clients se connectent aux serveurs de régions pour avoir accès au serveur central. Ce dernier vérifie les conditions d'entrée (nom utilisateur et mot de passe) ou crée un nouveau compte pour enregistrer un nouveau utilisateur. Chaque utilisateur dans l'espace virtuel partagé a besoin de connaître l'état de tout utilisateur dans l'espace. Les informations des états

sont stockées dans le serveur de région dans lequel l'utilisateur a crée son compte. les connexions entre les machines sont bidirectionnelles ce qui signifie qu'elles peuvent communiquer dans deux directions. La base de données est connectée à multiples serveurs de région par similarité un seul serveur de région est connecté à plusieurs clients.

La première étape est déterminer les agents résidants dans ses machines et ceux qui peuvent se déplacer. On introduit *Agent User* pour représenter le compte d'information de chaque utilisateur. Le système MUD permet à un utilisateur de créer multiple *Players*, ainsi il doit être capable de créer un nouveau *Player Agent* ou réveiller un *Player Agent* existant créé par un autre utilisateur. *User Agent* sera statique parce qu'il traite seulement les travaux lies à son serveur de région. L'étape suivante est déterminer les places et les transitions, pratiquent il existe trois événements: créer un *User Agent*, réveiller un *User Agent* et logoff. Un arc allant d'une place à une autre place représente une migration d'un agent.

Un *Player Agent* migre à un serveur de région qu'il était en attente, une fois arrive à son destination, le serveur de région génère un message de test et *Player Agent* attend d'acquérir les informations du serveur de base de données centrale.

Au niveau du serveur central de la base de données, plusieurs migrations sont arrivées de *Region Server*, *Central Agent* était en attente pour identifier les informations d'entrée ou créer un nouveau compte; Si l'identification est correcte *Central Agent* envoie un message *à Region Server* affirmant que les informations sont correctes. Si l'utilisateur veut créer un nouveau compte le *Central Database Disk Space Agent* alloue un espace pour le nouvel utilisateur si les informations entrées par ce dernier sont valides et l'espace disque est suffisant. Sinon un message d'erreur est créé et envoyé à Region Server et éventuellement au client pour corriger ses informations. Dans le cas contraire *Player Agent* qui est en attente peut maintenant faire des copies, chacune migre à un serveur de région et un migre au client, l'agent original reste dans son *Region Server* pour des raisons de sécurité. *Player Agent* reste dans *Region Server* jusqu'à ce que le client décide de logoff. Une fois l'utilisateur décide de quitter *User Agent* génère un message de logoff ce message sera envoyé à son serveur de région. La figure 2.2 montre qu'à l'arrivée de message de logoff, *Player Agent* migre au client. Dans la figure 2.1 *Player Agent* arrive de *Region Server* et met en sommeil, il peut être réveillé lorsque l'utilisateur décide de login une autre fois et réutilise son joueur.

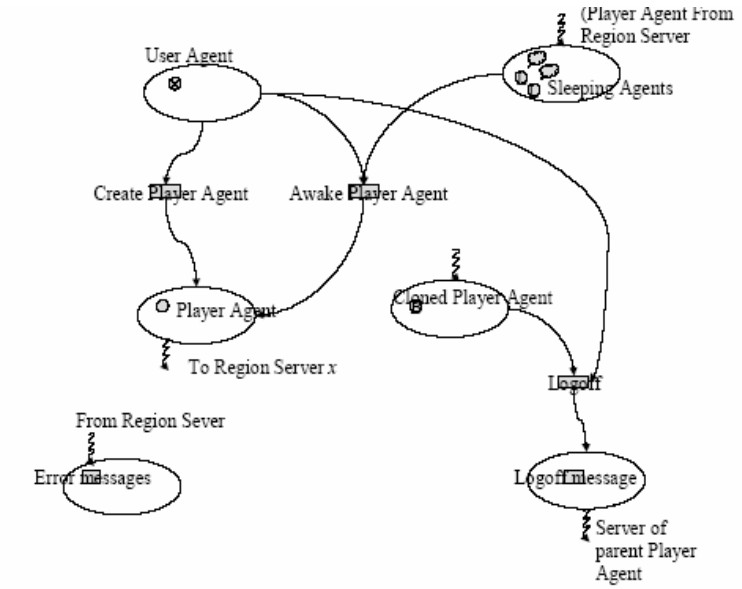


Figure I.2.4: Client

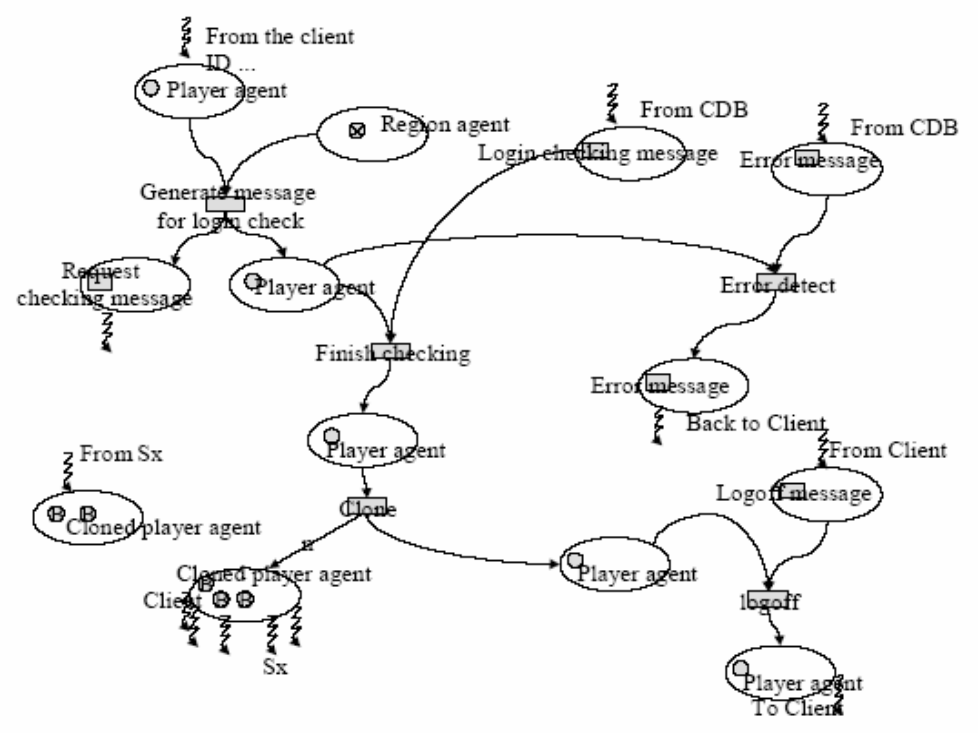


Figure I.2.5: Region Server

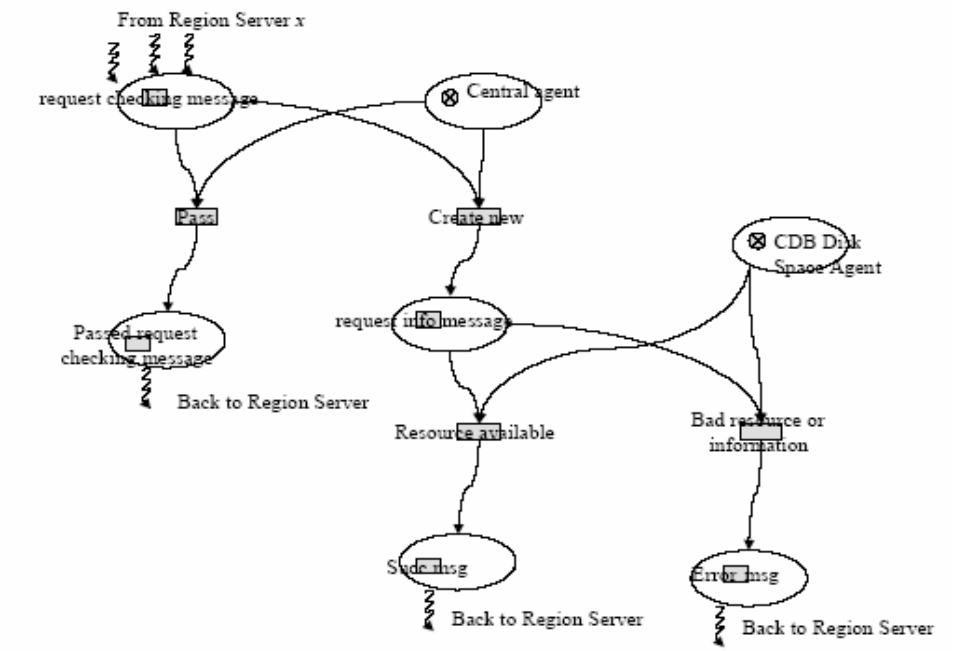


Figure I.2.6 : Central Data Base Server

3. MODELISATION DU COMPORTEMENT DES AGENTS MOBILES PAR LES HYPERNETS

Dans [18] l'idée de l'auteur est de modéliser les agents mobiles par des réseaux de Petri qui peuvent être manipulés comme des jetons dans d'autres réseaux de Petri. Il utilise le réseau de Petri appelé Hypernets.

L'Hypernets modélise une collection réseaux ouvert (open net) permettant la description modulaire et la hiérarchique du monde réel. Chaque open net est une composition des modules synchrones ce qui exprime la modularité des objets mobiles dont chacun de ces open net est responsable du déplacement des objets mobiles à travers un canal. La hiérarchie des agents correspond à l'hypothèse que chaque agent devrait être d'un niveau hiérarchique supérieur des agents qu'il manipule.

Un réseau ouvert (open net) est essentiellement le produit de synchronisation des réseaux séquentiels dont les transitions sont capables d'envoyer et recevoir des jetons d'un autre open net. Soit N un ensemble de réseaux ouverts un Hyper-marquage

$$m : \mathcal{N} \rightharpoonup \bigcup_{N \in \mathcal{N}} P_N$$

m est une fonction partielle décrit la distribution des objets d'un monde réel comme jetons dans le réseau de Petri dont la condition suivante : chaque open net n'a qu'un seul open net de plus haut niveau.

L'auteur prend l'étude d'un système de gestion de voyage par avion pour bien présenter et illustrer son réseau de Petri Hypernets. Il exige de différencier entre 3 types d'agents : aéroports, avions, et voyageurs. L'objectif est de décrire les aspects fondamentaux du voyage par avion en impliquant les trois types d'agents mobiles mentionnés précédemment. Un agent de chaque type montre un comportement mobile qui peut entraîner la manipulation des objets d'un autre type et ne peut pas manipuler directement un agent de son type. En fait, la séparation des types des agents est un dispositif de base de ce modèle.

Les activités de l'aéroport qui implique la manipulation des avions sont présentées dans la figure suivante :

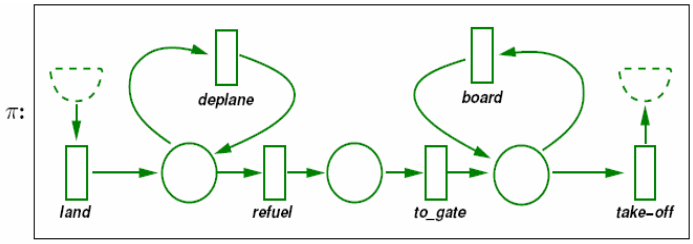


Figure I.2.7 : L'aéroport - Module de manipulation de l'avion

La description graphique des activités de manipulation d'avion dans figure3.1 est essentiellement standard. À l'exception de la pré-condition de la transition *land* et la post-condition de la transition *take-off*. Les demi cercle aux pointillés fournissent des entrées pour la transition *land* et des sorties pour la transition *take-off,* le dessin de demi cercle est censé pour indiquer la co-opération avec le niveau supérieur. Les pointillés accentuent le caractère virtuel de demi-places qui sont des moyens de la synchronisation des transitions entre les niveaux adjacents plutôt que des vraies places qui peuvent stocker la marque.

La figure précédente décrit *le* π - module d'un agent d'aéroport, *le* préfixe π pour indiquer que les marques manipulées dans le module sont des avions. *Un* τ –module pour la manipulation des voyageurs dans l'aéroport est présenté dans la figure suivante.

L'agent de contrôle du trafic aérien de plus haut niveau devrait contenir *un* π -module qui peut délivrer les avions aux aéroports pour l'atterrissage, et manipule les avions dans le ciel lors du leur décollage. Ce mécanisme de la coopération avec les agents de niveau plus bas est présenté dans *le* τ - module sur figure 3.2.

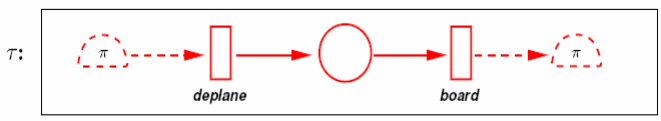


Figure I.2.8 : L'aéroport - Module de manipulation des voyageurs

Intuitivement, le résultat d'atterrissage de l'avion est l'apparition des voyageurs dans l'aéroport, l'action de la transition *deplane* est de déplacer les voyageurs de l'avion à l'aéroport. Ceci est formalisé par les demi-cercles aux pointillés qui représentent la pré-condition de la transition *deplane*. (La post-condition de la transition *board* est traitée d'une façon similaire). Noter que les places virtuelles utilisées pour la coopération avec les agents mobiles de bas niveau portent le nom du canal. Ce ci nécessite de déterminer la place appropriée qui est supposé fournir/recevoir l'agent mobile manipulé.

L'aéroport – open net :

La synchronisation de π –module et τ – module donne la représentation de l'aéroport suivante :

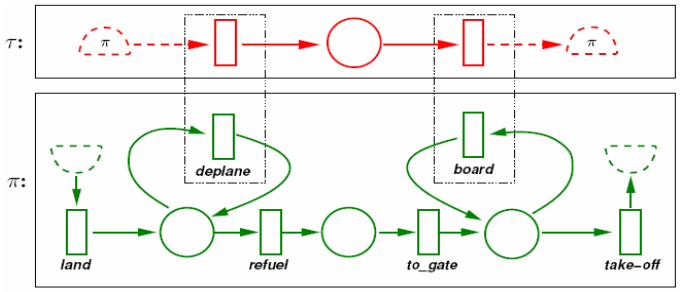


Figure I.2.9 : L'aéroport - Module de manipulation de l'avion et des voyageurs

L'avion :

Le modèle de l'avion est présenté dans la figure3.4, il se compose d'un seul module τ – module pour la manipulation des voyageurs.

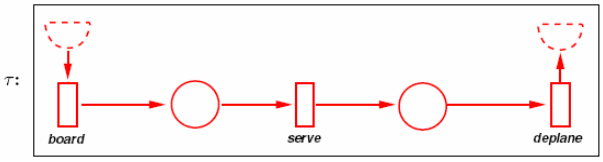


Figure I.2.10 : L'avion

La figure 3.5 présente petri hypernet constitué d'un seul réseau de niveau supérieur l'aéroport, un réseau pour l'avion, et deux pour les voyageurs.

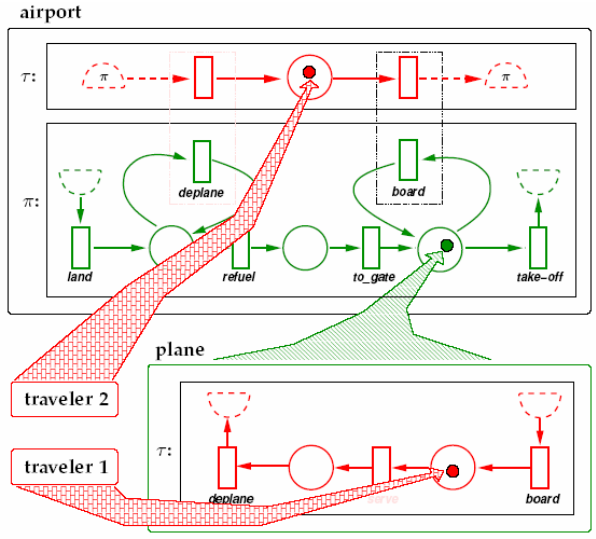
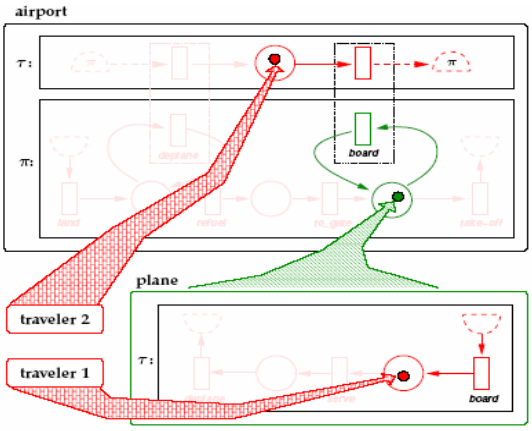


Figure I.2.11 : Le Petri hypernet modélisant le système

Un hypernet est dit ouvert parce qu'il a une transition capable d'envoyer et de recevoir des jetons du niveau supérieur. La figure ci-dessous décrit le résultat de la mise à feu de la transition ***board***



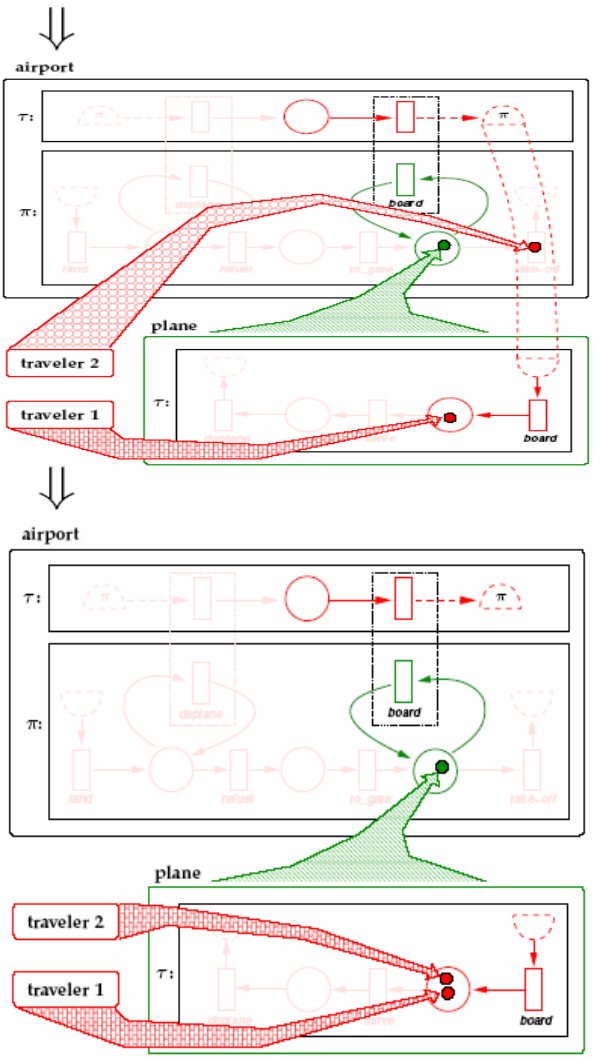


Figure I.2.12 : Le déclanchement de la transition *Board*

4. MODELISATION DU COMPORTEMENT DES AGENTS MOBILES PAR LES PRTNETS

L'auteur [19] utilisent un type de réseaux de patri de haut niveau qui s'appelle PrT nets pour modéliser les agents mobiles interopérables parce qu'il est le plus approprié de son point de vu grâce à similarité avec les système logiques. Les PrT nets sont définies comme suit :

Le Prt nets est un tuplet $(P, T, F, \sum, \varphi, M_0)$ où :

P : est ensemble fini des prédicats.

T : est un ensemble fini de transitions $(P \cap T = \emptyset, P \cup T = \emptyset)$ et $F \subseteq (P \times T) \cup (T \times P)$

(P, T, F) forme un réseau direct.

$\sum$: est une structure composée de certain genre d'individus (constants) avec certaines opérations et relations.

L : est la fonction d'étiquetage des arcs, étant donné un arc $f \in F$ l'étiquette de *f, L(f)* est un ensemble d'étiquettes, qui sont des tuples d'individus et des variables. Le tuple zéro noté par $<\phi>$ indique un prédicat sans argument (la place ordinaire dans les réseaux de Petri ordinaire).

φ*:*est une fonction de passage de formule d'inscription à une transition. L'inscription $t \in T$, $\varphi(t)$ est une formule logique construite à partir des variables, individus, opérations, et des relations dans la structure $\sum$.

M_0: est un marquage initial ou le marquage courant, Chaque jeton est un tuple des individus symboliques ou des termes structurés construits des individus et des opérations dans Σ.

Le modèle PrT nets est constitué de *system nets*, *agent net*, et *connector net* pour modéliser le comportement de l'environnement, d'agent mobile, et le connecteur respectivement. Chaque agent est défini comme un *PrT net*, appelé *agent net*. L'interface, le comportement, et l'état d'un agent sont respectivement modélisés par les prédicats d'entrée-sortie des messages envoyés-reçus, les transitions, et les prédicats *d'agent net*.

Chaque environnement est modélisé comme *PrT net*, appelé *system net*, et chaque composant inclut un *system net* et un réseau interne de connexion. Chaque *system net* a des interfaces d'entrées/sorties internes reliées aux connecteurs internes qui transfèrent les messages entre les agents et le système. Un groupe de systèmes est reliés par l'intermédiaire des connecteurs externes, et des arcs de *connector net* sont censés être correctement marqués de sorte qu'un agent soit toujours migré à une destination appropriée.

Chaque *hôte* se compose d'un *agent system*, d'un certain nombre d'agents, d'un *finder*, et d'un connecteur interne. Des *PrT nets* sont employés pour modéliser les comportements des systèmes d'agent mobiles, agents mobiles, finders, aussi bien que des connecteurs où *Agent net* est pris comme jeton du *system net*. Dans *l'hôte*, chaque *agent system* communique avec le *finder* directement, mais les autres agents communiquent avec le *finder* à travers *l'agent system.*

Le *finder* reçoit des messages des connecteurs externes ou *d'agent system* local. Si le message concernant la localisation d'un *agent* ou un *agent system*, l'information relative est recherchée dans la base de données et envoyée de nouveau à l'enquêteur ; si le message

concernant l'enregistrement ou la suppression d'un *agent* ou d'un *agent system*, alors l'information relative est ajoutée ou enlevée de la base de données et l'information mise à jour est expédiée à la destination. Le modèle PrT net d'un *finder* est défini dans figure 4.1.

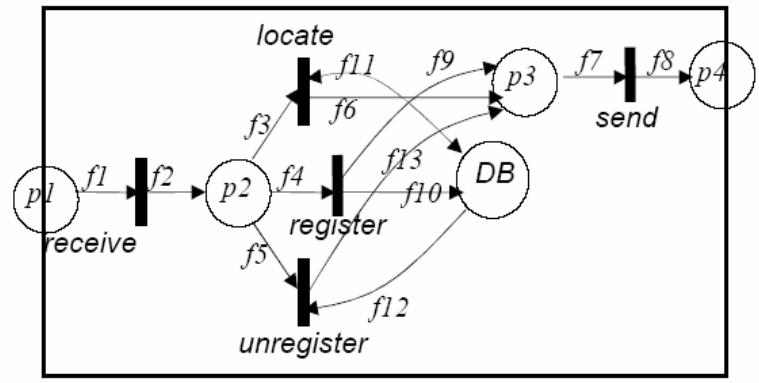


Figure I.2.13: Le PrT net modélisant le *finder*

Chaque *agent system* reçoit des messages d'autres *agents system* par les connecteurs externes, et les messages sont expédiés à la destination par le *finder* de destination. Chaque agent communique avec son système par le connecteur interne. *Agents system* peuvent commander le fonctionnement d'agent, créer un agent en utilisant agent template, et envoyer des données aux agents. La figure suivante représente le Prt Nets qui modélise *Agent System.*

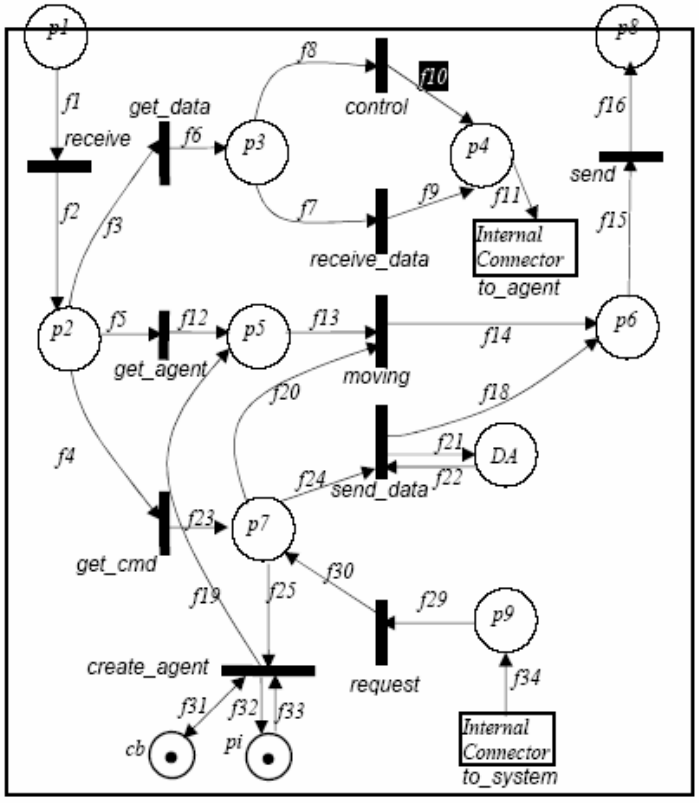


Figure I.2.14: Le PrT net modélisant l'Agent System

CONCLUSION

Dans ce chapitre on a exposé certaines modélisations des systèmes d'agents mobiles par plusieurs types de réseaux de Petri de haut niveau. L'idée est très intéressante et ces modélisations permettent pratiquement de bien comprendre et de bien illustrer les fonctionnalités des agents mobiles. Cependant, et dans la majorité des modélisations présentées, l'agent est modélisé par un réseau de Petri et considéré comme un jeton dans un autre réseau de Petri de plus haut niveau, ce qui peut rendre la représentation du système complet peu intuitive, ainsi l'implémentation et la programmation du système plus délicate. Ce type de présentation masque aussi quelques propriétés des agents mobiles tels que la sociabilité. Mais en générale l'utilisation des réseaux de Petri de haut niveau dans le domaine des systèmes multi agents et spécialement dans les agents mobiles reste la meilleure représentation formelle qui donne des résultats valides et intuitifs.

Chapitre III :

Les ECATNets

- Introduction
- La logique de réécriture
- Maude
- Les ECATNets
- Implémentation des ECATNets dans Maude

INTRODUCTION

Les ECATNets [1] (Bettaz M., Maouche M., 1993) sont une catégorie des réseaux de Petri algébriques basés sur les réseaux de Petri de haut niveau et les types abstraits algébriques. La sémantique des ECATNets est définie en termes de la logique de réécriture qui offre une base solide et rigoureuse pour toute démarche de vérification des propriétés des systèmes décrits. Les réseaux de Petri sont utilisés pour leur pouvoir d'exprimer la concurrence et l'aspect dynamique et les types abstraits algébriques sont utilisés pour leur puissance de décrire abstraitement les données. Le besoin de spécifier explicitement le comportement des processus et les structures des données motive leur association dans un cadre uni. Autant que des réseaux de Petri, les ECATNets sont un modèle formel permet la spécification et la validation des applications complexes de divers domaines tels que la communication réseau, la programmation concurrente et les systèmes de production. Il fournit un formalisme rapidement compris dû à leur description graphique [2]. En outre, les ECATNets ont une théorie puissante et des outils de développement basés sur une logique ayant une sémantique saine et complète.

Ce chapitre est organisé comme suit : La section 1 introduit la logique de réécriture et sa sémantique et présente ses concepts de base. La section 2, présente le langage de la logique de réécriture Maude et ses différents niveaux de spécification. Dans la section 3 nous donnons une présentation générale des ECATNets et de leur description dans la logique de réécriture. . Dans la section 4, nous donnons l'implémentation des ECATNets dans Maude.

1. LA LOGIQUE DE REECRITURE

La logique de réécriture, dotée d'une sémantique saine et complète, a été introduite par Meseguer [Mes92], [Mes96]. Elle permet de décrire les systèmes concurrents. Cette logique unifie plusieurs modèles formels qui expriment la concurrence. La logique de réécriture est une logique qui permet de raisonner d'une manière correcte sur les systèmes concurrents non-déterministes ayant des états et évoluant en termes de transitions. Elle explique n'importe quel comportement concurrent dans un système avec une sémantique de 'vraie concurrence', par des déductions dans cette logique [4]. Le langage de la logique de réécriture Maude est l'un des langages les plus puissants dans la spécification formelle, la programmation et la vérification des systèmes concurrents [2].

Chaque système concurrent, est représenté par une théorie de réécriture $\Re = (\Sigma, E, L, R)$. Sa structure statique est décrite par la signature (Σ, E), tandis que sa structure dynamique

est décrite par les règles de réécriture R. Une conséquence intéressante des définitions de la logique de réécriture est qu'une théorie $\Re = (\Sigma, E, L, R)$ devient une spécification exécutable du système concurrent qu'elle formalise [4].

Une théorie de réécriture $\Re$ est une 4-tuple $\Re = (\Sigma, E, L, R)$, tel que (Σ, E) est une signature ; Σ est un ensemble des sortes et opérations, et E est un ensemble de Σ-équations. La signature (Σ, E) est une théorie équationnelle qui décrit la structure algébrique particulière des états du système (multi-ensemble, arbre binaire, etc...) qui sont distribués selon cette même structure. L'ensemble $R \subseteq L \times (T_{\Sigma,E}(X))^2$ est l'ensemble des pairs tel que le premier composant est une étiquette et le second est un pair des classes d'équivalence des termes modulo les équations E, avec $X = \{x_1,\ldots,x_n,\ldots\}$ un ensemble comptable et infini des variables. Les éléments de R s'appellent les règles conditionnelles de réécriture. Ils décrivent les transitions élémentaires et locales dans un système concurrent. Chaque règle de réécriture correspond à une action pouvant se produire en concurrence avec d'autres actions. La réécriture opérera les classes d'équivalence des termes, modulo l'ensemble des équations E. Pour une réécriture $(r, [t], [t'])$, $([u_1],[v_1]),\ldots,([u_k],[v_k])$), nous utilisons la notation : $r : [t] \rightarrow [t']$ if $[u_1] \rightarrow [v_1] \wedge \ldots \wedge [u_k] \rightarrow [v_k]$, tel que $[t]$ représente la classe d'équivalence du terme *t*. Une règle *r* exprime que la classe d'équivalence contenant le terme *t* a changé à la classe d'équivalence contenant le terme *t'* si la partie conditionnelle de la règle, $[u_1] \rightarrow [v_1] \wedge \ldots \wedge [u_k] \rightarrow [v_k]$, est vérifiée. Etant donné une théorie $\Re$, nous disons qu'une séquence $r : [t] \rightarrow [t']$ est prouvable dans $\Re$ ou $r : [t] \rightarrow [t']$ est $\Re$-réécriture concurrente et nous écrivons $\Re \vdash r : [t] \rightarrow [t']$ Ssi $[t] \rightarrow [t']$ est dérivable des règles dans $\Re$ par une application finie des règles de déduction suivantes [4]:

1- Réflexivité. Pour chaque $[t] \in T_{\Sigma,E}(X)$, $\dfrac{}{[t] \rightarrow [t]}$

2- Congruence. Pour chaque $f \in \sum_n, n \in \mathrm{N}$ $\dfrac{[t_1] \rightarrow [t_1'] \quad \ldots \quad [t_n] \rightarrow [t_n']}{[f(t_1,\ldots,t_n)] \rightarrow [f(t_1',\ldots,t_n')]}$

3- Remplacement. Pour chaque règle de réécriture $r : [t(x_1,\ldots,x_n)] \rightarrow [t'(x_1,\ldots,x_n)]$ dans R,

$$\frac{[w_1] \rightarrow [w_1'] \quad \ldots \quad [w_n] \rightarrow [w_n']}{[t(\bar{w}/\bar{x})] \rightarrow [t'(\bar{w}'/\bar{x})]}$$

Tel que $t(\bar{w}/\bar{x})$ indique la substitution simultanée des w_i pour x_i dans *t*.

4- Transitivité. $\dfrac{[t_1] \rightarrow [t_2] \quad [t_2] \rightarrow [t_3]}{[t_1] \rightarrow [t_3]}$

Une théorie de réécriture est une description statique d'un système concurrent. Sa sémantique est définie par un modèle mathématique qui décrit son comportement. Le modèle pour une théorie de réécriture étiquetée $\Re$ = (Σ, E, L, R), est une catégorie $\tau_{\Re}(X)$ dont les objets (états) sont des classes d'équivalence des termes $[t] \in T_{\Sigma, E}(X)$ et dont les morphismes (transitions) sont des classes d'équivalence des termes-preuves (proof-terms) représentant des preuves dans la déduction de réécriture

2. MAUDE

Maude est un langage de spécification et de programmation basé sur la logique de réécriture. Ce langage est simple, expressif et performant, et offre peu de constructions syntaxiques et une sémantique bien définie. Il est, par ailleurs, possible de décrire naturellement différents types d'applications [2]. Maude est un langage qui supporte facilement le prototypage rapide et représente un langage de programmation avec des performances compétitives. Le langage Maude définit deux niveaux de spécification. Un niveau concerne la spécification du système tandis que l'autre est lié à la spécification des propriétés.

2.1. Niveau de spécification de système

Ce niveau est basé sur la théorie de réécriture. Il est principalement décrit par les modules systèmes. Pour une bonne description modulaire, trois types de modules sont définis dans Maude. Les modules *fonctionnels* permettent de définir les types de données. Les modules *systèmes* permettent de définir le comportement dynamique d'un système. Enfin, les modules *orienté-objet* qui peuvent, en fait, être réduits à des modules systèmes offrent explicitement les avantages du paradigme objet [4].

Modules Fonctionnels.

Les modules fonctionnels définissent les types de données et les opérations qui leur sont applicables en termes de la théorie des équations. L'algèbre initiale est le modèle mathématique (dénotationel) pour les données et les opérations. Les éléments de cette algèbre sont des classes d'équivalence des termes sans variables (ground terms) modulo des équations. Si deux termes sans variables sont égaux par le biais des équations, alors ils appartiennent à la

même classe d'équivalence. En utilisant des équations comme des règles de réduction, chaque expression pourra être évaluée à sa forme réduite dite représentation canonique. Cette dernière est unique et représente tous les termes de la même classe d'équivalence. Les équations dans un module fonctionnel sont orientées. Elles sont utilisées de gauche à droite et le résultat final de la simplification d'un terme initial est unique quelque soit l'ordre dans lequel ces équations sont appliquées. En plus des équations, ce type de modules supporte les axiomes d'adhésions (memberships axioms). Ces axiomes précisent l'appartenance d'un terme à une sorte. Cette appartenance peut être sous certaines conditions ou non. Cette condition est une conjonction des équations et des tests d'adhésions inconditionnels. Ce qui suit un exemple classique d'un type abstrait de données, celui des naturels [4] :

```
fmod NAT is
  sort Nat . sort PositiveNat .
  subsort PositiveNat < Nat .
  op 0 : -> Nat .
  op S_ : Nat -> Nat .
  op _+_ : Nat Nat -> Nat [comm] .    *** N + M = M + N
  vars N M : Nat .
  var P : PositiveNat .
  cmb P : PositiveNat if P =/= 0 .*** axiome d'adhésion indiquant que P est de type PositiveNat
                                      *** si P est différent de 0
  eq N + 0 = N                        *** N + 0 et N dénotent le même terme (même classe d'équivalence) qui est [N]
  eq (S N) + (S M) = S S (N + M) .
endfm
```

Modules Systèmes.

Les modules systèmes permettent de définir le comportement dynamique d'un système. Ce type de module augmente les modules fonctionnels par l'introduction de règles de réécriture. Un degré maximal de concurrence est offert par ce type de module. Un module

système décrit une "théorie de réécriture" qui inclut des sortes, des opérations et trois types d'instructions : équations, adhésions et règles de réécriture. Ces trois types d'instructions peuvent être conditionnels. Une règle de réécriture spécifie une "transition concurrente locale" qui peut se dérouler dans un système. L'exécution de telle transition spécifiée par la règle peut avoir lieu quand la partie gauche d'une règle correspond à (match) une portion de l'état global du système et bien sûr la condition de la règle est valide. Un exemple suivant montre la puissance de ce type de module [4]:

```
mod NAT is

extending NAT .

op _? _ : Nat Nat -> Nat .

vars N M : Nat .

rl [R1] : N ? M => N .

rl [R2] : N ? M => M .

endm
```

L'opérateur ? modélise le choix aléatoire entre deux entiers. Si nous avions utilisé uniquement la logique équationnelle pour décrire ce système, alors nous aurions les équations suivantes : N ? M = N et N ? M =M. Ce qui signifie que N = M, c'est-à-dire que tous les entiers sont égaux, alors le modèle dénotationnel (l'algèbre initiale) de ce module s'effondre à un seul élément! Le module NAT n'est pas protégé. La logique équationnelle est faible pour décrire ce système. La logique équationnelle est réversible, alors que l'évolution des systèmes concurrents ne l'est pas forcément. Les règles R1 et R2 sont irréversibles et elles sont concurrentes. Dans ce cas, la sémantique de l'opérateur ? est bien définie.

2.2. Niveau de Spécification de Propriété

Ce niveau de spécification définit les propriétés du système à vérifier. Le système est bien sûr décrit à l'aide d'un module système. En évaluant l'ensemble des états accessibles à partir d'un état initial, le Model Checking permet de vérifier une propriété donnée dans un état ou un ensemble d'états. La propriété est exprimée dans une logique temporelle LTL (Linear

Temporel Logic) ou BTL (Branching Temporel Logic). Le Model Checking supporté par la plate-forme de Maude utilise la logique LTL essentiellement pour sa simplicité et les procédures de décision bien définies qu'il offre [4].

Dans un module prédéfini LTL, on trouve la définition des opérateurs pour la construction d'une formule (propriété) dans la logique temporelle linéaire. Dans ce qui suit, et en écartant certains détails d'implémentation, on trouve une partie des opérateurs LTL dans la syntaxe de Maude:

```
fmod LTL is                                  *** opérateurs définis de LTL
 op _->_ : Formula Formula -> Formula .      *** implication
 op _<->_ : Formula Formula -> Formula .     *** equivalence
 op <>_ : Formula -> Formula .               *** eventualy
 op []_ : Formula -> Formula .               *** always
 op _W_ : Formula Formula -> Formula .       *** unless
 op _|->_ : Formula Formula -> Formula .     *** leads-to
 op _=>_ : Formula Formula -> Formula .      *** strong implication
 op _<=>_ : Formula Formula -> Formula .     *** strong equivalence
…
endfm
```

Les opérateurs LTL sont représentés dans Maude en utilisant une forme syntaxique semblable à leur forme d'origine. Par exemple, l'opération [] est défini dans Maude par l'opérateur (*always*). Cet opérateur s'applique sur une formule pour donner une nouvelle formule. Nous avons, par ailleurs, besoin d'un opérateur indiquant si une formule donnée est vraie ou fausse dans un certain état. Nous trouvons un tel opérateur (|=) dans un module prédéfini appelé SATISFACTION :

```
fmod SATISFACTION is
 protecting LTL .
 sort State .
 op _|=_ : State Formula ~> Bool .
```

```
endfm
```

L'état State est générique. Après avoir spécifié le comportement de son système dans un module système de Maude, l'utilisateur peut spécifier plusieurs prédicats exprimant certaines propriétés liées au système. Ces prédicats sont décrits dans un nouveau module qui importe deux modules : celui qui décrit l'aspect dynamique du système et le module SATISFACTION. Soit, par exemple, M-PREDS le nom du module décrivant les prédicats sur les états du système :

```
mod M-PREDS is
  protecting M .
  including SATISFACTION .
  subsort Configuration < State .
  …
endm
```

M est le nom du module décrivant le comportement du système. L'utilisateur doit préciser que l'état choisi (configuration choisie dans cet exemple) pour son propre système est sous type de type State. A la fin, nous trouvons le module MODEL-CHECKER qui offre la fonction model-Check. L'utilisateur peut appeler cette fonction en précisant un état initial donné et une formule. Le Model Checker de Maude vérifie si cette formule est valide (selon la nature de la formule et la procédure du Model Checker adoptée par le système Maude) dans cet état ou l'ensemble de tous les états accessibles depuis l'état initial. Si la formule n'est pas valide, un contre exemple (counterexample) est affiché. Le contre exemple concerne l'état dans lequel la formule n'est pas valide [4]:

```
fmod MODEL-CHECKER is
including SATISFACTION .
…
op counterexample : TransitionList TransitionList -> ModelCheckResult [ctor] .
op modelCheck : State Formula ~> ModelCheckResult .
…
endfm
```

3. ECATNETS

Les ECATNets [1] (Bettaz M., Maouche M., 1993) sont une catégorie des réseaux de Petri algébriques basés sur les réseaux de Petri de haut niveau et les types abstraits algébriques. La sémantique des ECATNets est définie en termes de la logique de réécriture qui offre une base solide et rigoureuse pour toute démarche de vérification des propriétés des systèmes décrits. Les réseaux de Petri sont utilisés pour leur pouvoir d'exprimer la concurrence et l'aspect dynamique et les types abstraits algébriques sont utilisés pour leur puissance de décrire abstraitement les données. Le besoin de spécifier explicitement le comportement des processus et les structures des données motive leur association dans un cadre uni. Autant que des réseaux de Petri, les ECATNets est un modèle formel permet la spécification et la validation complexes. Il fournit un formalisme rapidement compris dû à leur description graphique [2]. En outre, les ECATNets ont une théorie puissante et des outils de développement basés sur une logique ayant une sémantique saine et complète.

Les ECATNets donc combinant les forces des réseaux de Petri avec ceux des types de données abstraits. Les places sont identifiées par des multi-ensembles des termes algébriques. Les arcs entrants de chaque transition t, c.-à-d. (p, t), sont marqués par deux inscriptions IC(p, t) (Input Conditions) et DT(p, t) (Destroyed Tokens). Les arcs sortants de chaque transition t, c.-à-d. (t, p'), sont marqués par CT(t,p') (Created Tokens). Finalement, chaque transition t est marquée par une inscription TC(t) (Transition Condition) (voir la figure 1). IC(p, t) indique une condition qui valide une transition t, DT(p, t) indique le marquage (un multi-ensemble) qui doit être enlevé de p lors du franchissement de t, CT(t, p') indique le marquage (un multi-ensemble) qui doit être ajouté à p' lors du franchissement de t. En conclusion, TC(t) représente un terme booléen qui indique une condition additionnelle pour le franchissement de la transition t. L'état courant d'un ECATNet est donné par l'union des termes ayant la forme (p, M(p)) [3]. Par exemple, soit un réseau contenant une transition t ayant une place d'entrée p marquée par le multi-ensemble $a \oplus b \oplus c$ et une place de sortie vide p', l'état distribué s de ce réseau est donnée par le multi-ensemble : $s = (p, a \oplus b \oplus c)$.

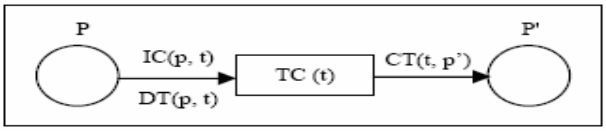


Figure I.3.1 Exemple d'un ECATNet

Une transition t est franchissable quand des diverses conditions sont simultanément vraies. La première condition est que chaque IC(p, t) pour chaque place d'entrée p est tirable. La deuxième condition est que TC(t) est vrai. Enfin l'addition de CT(t, p') à chaque place de sortie p' ne doit pas avoir comme conséquence d'excéder sa capacité quand cette capacité est finie. Si t est franchise alors DT(p, t) est complètement enlevé (cas positif) de la place d'entrée p et simultanément CT(t, p') est ajouté à la place de sortie p'. Notons que dans le cas non positif, on enlève les éléments en commun entre DT(p, t) et M(p) [3]. Le franchissement d'une transition et les conditions de ce franchissement sont formellement exprimés par des règles de la logique de réécriture. Les notions de base de cette logique dans le cadre de leur utilisation dans les ECATNets sont résumées comme suit.

- Les axiomes sont en réalité des règles de réécriture conditionnelles décrivant les effets des transitions comme des types élémentaires de changement [4].

- Les règles de déduction nous permettent d'avoir des conclusions valides des ECATNets à partir des changements [4].

- Une règles de réécriture est de la forme "t: u $\rightarrow$ v if boolexp"; où u et v sont respectivement les côtés gauche et droit de la règle, t est la transition liée à cette règle, et boolexp est un terme booléen. Plus précisément, u et v sont des multi-ensembles des pairs de la forme $(p, [m]_{\oplus})$, telle que p est une place de réseau, $[m]_{\oplus}$ est un multi-ensemble des termes algébriques, et l'union multi-ensemble sur ces termes [3].

- Notons que les termes sont considérés comme singletons. L'union des multi-ensembles sur les pairs $((p, [m]_{\oplus})$ est dénoté par $\otimes$. $[x]_{\otimes}$ dénote la classe d'équivalence de x, selon les axiomes ACI (Associativity Commutativity Identity = ϕ_M) pour $\otimes$. Un état d'un ECATNet lui-même est représenté par un multi-ensemble de telles paires [2].

Les formes des règles de réécriture (c.-à-d., les méta-règles) à associer avec les transitions d'un ECATNet donné sont définies comme suit : Soit **R** un ensemble de règles de réécriture (définissant tous les types élémentaires de changement), **R** valide une séquence **s** $\rightarrow$ **s'** (définissant un changement global d'un état à un autre) ssi **s** $\rightarrow$ **s'** peut être obtenu par des applications finies et concurrentes des règles de déduction suivantes : Réflexivité (Reflexivity), Congruence (Congruence), Remplacement (Replacement), Décomposition (Splitting), Recombinaison (Recombination) et l'Identité (Identity). Nous rappelons que la règle de réflexivité décrite que chaque état se transforme en lui-même. La règle de congruence que les changements élémentaires doivent correctement propagés. La règle de remplacement

est utilisée quand les instanciations de variables deviennent nécessaires. Les règles de décomposition et de recombinaison nous permettent, en décomposant et recombinant "judicieusement" les multi-ensembles des classes d'équivalences des termes, pour détecter les calculs montrant un maximum de concurrence. Nous rappelons maintenant les formes des règles de réécriture (i.e, les méta-règles) à associer avec les transitions d'un ECATNet donné.

3.1. Formes des Règles de Réécriture : Cas positif sans Arcs Inhibiteurs

IC(p,t) est de la forme $[m]_{\oplus}$

Cas 1. $[IC(p, t)]_{\oplus} = [DT(p, t)]_{\oplus}$

La forme de cette règle est : $t : (p, [IC(p ,t)]_{\oplus}) \rightarrow (p', [CT(t, p')]_{\oplus})$

Cas 2. $[IC(p, t)]_{\oplus} \cap [DT(p, t)]_{\oplus} = \phi_M$

Cette situation correspond à vérifier que IC(p, t) est inclus dans M(p) et, retirer DT(p,t) de M(p). La forme de cette règle est : $t : (p, [IC(p, t)]_{\oplus}) \otimes (p, [DT(p, t)]_{\oplus}) \rightarrow (p, [IC(p, t)]_{\oplus}) \otimes (p', [CT(t, p')]_{\oplus})$

Cas 3. $[IC(p, t)] \cap [DT(p, t)] \neq \phi_M$

Cette situation correspond au cas le plus général. Elle peut cependant être résolue d'une manière élégante en remarquant qu'elle pourrait être apportée aux deux cas précédents. Ceci est réalisé en remplaçant la transition t de ce cas par deux transitions t1 et t2. Ces transitions, une fois elles sont franchises concurremment, donnent même effet global que notre transition. Nous éclatons IC(p, t) en deux multi-ensembles IC1(p, t1) et IC1(p, t2). Nous éclatons également DT(p, t) en deux multi-ensembles DT1(p, t1) et DT1(p, t2) :

$IC(p, t) = IC1(p, t1) \cup IC1(p, t2)$, $DT(p, t) = DT1(p, t1) \cup DT1(p, t2)$.

Les quatre multi-ensembles obtenus doivent réaliser $IC1(p, t1) = DT1(p, t1)$ et $IC2(p, t2) \cap DT2(p, t2) = \phi_M$. Le franchissement de la transition t est identique au franchissement en parallèle des deux transitions t1 et t2.

3.2. Formes des Règles de Réécriture : Cas Général

IC(p,t) est de la forme $[m]_{\oplus}$

Cas 1. $[IC(p,t)]_{\oplus} = [DT(p,t)]_{\oplus}$

La forme de cette règle est : $t : (p, [IC(p, t)]_{\oplus}) \rightarrow (p', [CT(t, p')]_{\oplus})$

Cas 2. $[IC(p, t)]_{\oplus} \cap [DT(p, t)]_{\oplus} = \phi_M$

Cette situation correspond à vérifier que IC(p, t) est inclus dans M(p) et, retirer DT(p, t) de M(p). La forme de cette règle est : t : (p, $[IC(p, t)]_{\oplus}$) ⊗ (p, $[DT(p, t)]_{\oplus}$) → (p, $[IC(p, t)]_{\oplus}$) ⊗ (p', $[CT(t, p')]_{\oplus}$)

Cas 3. $[IC(p, t)]_{\oplus} \cap [DT(p, t)]_{\oplus} \neq \phi_M$

Ce cas peut être résolu aussi en combinant les deux cas précédents. Ceci peut se faire en remplaçant la transition de ce cas par deux transitions, une fois franchises concurremment, donne le même effet global comme cette transition. En réalité, ce remplacement montre comment spécifier une situation donnée en deux niveaux d'abstraction.

IC(p, t) est de la forme ˜[m]⊕

La forme de cette règle est:

t : (p, $[DT(p, t)]_{\oplus} \cap [M(p)]_{\oplus}$) → (p', $[CT(t, p')]_{\oplus}$) if ($[IC(p, t)]_{\oplus}$ \ ($[IC(p, t)]_{\oplus} \cap [M(p)]_{\oplus}$)) = ϕ_M → [false]

IC(p, t) = empty

La forme de cette règle est la suivante :

t : (p, $[DT(p, t)]_{\oplus} \cap [M(p)]_{\oplus}$) → (p', $[CT(t, p')]_{\oplus}$) if $[M(p)]_{\oplus} \rightarrow \phi_M$

Telle que la capacité C(p) est finie, la partie conditionnelle de règle de réécriture va inclure le composant suivant :

$[CT(p, t)]_{\oplus} \oplus [M(p)]_{\oplus} \cap [C(p)]_{\oplus} \rightarrow [CT(p, t)]_{\oplus} \oplus [M(p)]_{\oplus}$ (**Cap**)

Dans le cas où il y a une condition de transition TC(t), la transition est représentée dans la logique de réécriture par une règle non conditionnelle.

4. IMPLEMENTATION DES ECATNETS DANS MAUDE

Considérons la version textuelle des ECATNets dans Maude. Le module générique qui décrit des opérations de base d'un ECATNet est le suivant [4]:

```
fmod GENERIC-ECATNET is

  sorts Place Marking GenericTerm .

  op mt : -> Marking .

  op <_;_> : Place GenericTerm -> Marking .

  op _._ : Marking Marking -> Marking [assoc comm id: mt] .
```

endfm

Comme illustré dans ce code, mt dénote le marquage vide dans un ECATNet. Nous définissons l'opération "<_;_>" qui permet la construction des marquages élémentaires. Les sous-lignes dans cette définition indiquent les positions des paramètres. Le premier paramètre de cette opération est une place et le second est un terme algébrique quelconque qui peut être dans cette place. Pour mettre un terme 0 par exemple de sorte Nat dans une place p, il faut préciser dans le code subsort Nat < GenericTerm, c.-à-d., que Nat est sous-sorte de GenericTerm. Dans ce cas, le terme < p ; 0 > est valide. Nous n'avons pas défini une opération qui implémente ⊕. L'opération "_._" implémentant l'opération ⊗ est suffisante grâce au concept de décomposition. Si une place contient plusieurs termes, par exemple (p, a ⊕ b), alors on peut l'écrire < p ; a > . < p ; b >.

Pour la méta-représentation des ECATNets, notons que l'utilisateur n'est pas obligé d'écrire son ECATNet dans une méta-représentation. Il peut l'écrire dans le mode ordinaire, puis il utilise la fonction de Maude upModule qui permet de transformer la représentation d'un module à sa méta-représentation. Le passage dans l'autre sens aussi est possible grâce à la fonction downModule [4].

Partie II :

Modélisation d'un système d'agents mobiles par les ECATNets

Chapitre I :

Description et Motivation

- Description de la modélisation
- Motivation

1. DESCRIPTION DE LA MODELISATION DES SYSTEMES D'AGENTS MOBILES PAR LES ECATNETS

Les ECATNets sont un modèle formel qui combine les avantages des réseaux de Petri, des types abstraits données, et de la logique de réécriture dans la modélisation, la spécification et la validation des systèmes complexes et des systèmes répartie. Ils ont aussi la capacité d'exprimer la concurrence, le parallélisme, et l'aspect dynamique des systèmes. La représentation graphique des réseaux de Petri donne une grande puissance d'expression, elle offre une symbolique simple facile à comprendre et une capacité d'exprimer les comportements parallèles et concurrents. La sémantique des ECATNets est définie en terme de la logique de réécriture qui est une logique saine, complète, permet de raisonner de manière concrète, elle unifie plusieurs modèles formels qui expriment la concurrence, et peut expliquer tout situation du monde réel. Les types abstraits algébriques sont utilisés pour leur puissance de décrire abstraitement les données. Ils permettent une compréhension rapide de la structure de données et ils sont faciles à implémenter.

Avec l'accroissement spectaculaire des réseaux, l'émergence des systèmes distribués et l'apparition des nouvelles technologies dans le domaine de l'intelligence artificielle et en particulier la technologie des agents mobiles, et suite aux limites prouvées de la modélisation des systèmes d'agents mobiles par les réseaux de Petri ordinaire et quelque réseaux de Petri de haut niveau, et pour avoir une modélisation formelle, valide et intuitive on a pensé à profiter des avantages des ECATnets (les forces de réseau de Petri, type abstrait de données et logique de réécriture), et donc présenter l'apport des ECATNets dans la modélisation ce type de systèmes.

Un système d'agents mobiles est constitué principalement d'un agent stationnaire qui est considéré comme le gestionnaire du système et d'un ensemble d'agents mobiles qui collaborent pour accomplir des tâches dans le système, dont chacun des agents doit avoir certaines propriétés telles que: l'autonomie, la réactivité, l'initiative, le raisonnement, l'apprentissage et l'habilité sociale et spécialement la capacité de se déplacer pour les agents mobiles.

Ces Agents sont représentés dans notre modèle par des types abstraits algébriques, ce qui facilite la compréhension et l'implémentation du système, dont chaque agent à ces propres données, son propre code et ces propre opérations, cette propriété montre l'autonomie des

agents, où on peut bien remarquer que chaque agent est définie indépendamment des autres agents, et que son fonctionnement ne nécessite aucune intervention des autre agents.

L'habilité social des agents est montrée par les différentes communications établie entre les différents agents, les échanges les plus importants entre les agents peuvent se résumer en messages, requêtes, ordres, questions… ces échanges sont modélisés de la même manière que les agents par des types abstraits algébriques.

Les deux types abstraits de données définie précédemment seront utilisés dans notre modélisation en tant que jetons du réseau de Petri, donc chaque jeton et soit un agent soit un message, une création d'un jeton agent signifie la naissance d'un nouveau agent, la création d'un jeton de type message signifie une communication entre les agents est établie et ainsi de suite.

Pour bien présenter la mobilité dans un système d'agents mobiles il faut avoir un ensemble de différentes locations (des machines) où l'agent peut se trouver, et pour cela l'environnement doit être divisé en plusieurs endroits. Ces différentes locations sont bien illustrées par la composition de notre modèle d'ECATNets en parties dont chaque partie présente une location de l'agent ou le type de location de l'agent, et dont chacune est composée d'un ensemble de places et de transitions chargés à représenter le comportement des agents dans un endroit précis. Les transitions externes sont chargées à modéliser l'action de mobilité des agents, l'apparence de chacune de ces transitions signifie que l'agent se déplace d'un endroit vers un autre. Notre modèle donc peut montrer le parallélisme de l'exécution des taches, l'autonomie des agents dont chacun est dans une location "l" ainsi que la réactivité de l'agent dont il peut s'adapter et changer son comportement selon sa location.

Suite à la définition des ECATNets, les places sont identifiées par des multi-ensembles des termes algébriques, on peut dire donc que chaque place est identifiée par l'ensemble des agents existants à un moment donné dans le système. Mais pour rendre la représentation plus intuitive et pour bien montrer la propriété de mobilité qui est considéré comme une propriété importante dans les systèmes d'agents mobiles, on peut définir les places comme l'ensemble des agents et ses états à un moment donné "t" et à un endroit ou une location "l".

En résumé, la migration est représentée par les transitions entre les différentes locations dont chaque transition entre deux locations différentes signifie que l'agent migre d'une location vers une autre. Les autres transitions ou plus spécialement les transitions situant

dans la même location montrent les activités des différents agents dans la machine, ces activités différent d'un agent à un autre, mais elles sont principalement les activités qui change l'état du système ou plus précisément qui influence sur le nombre, le types et l'état des agents présents dans le système. Les autres activités comme les activités spécifiques et les activités élémentaires de chaque agent sont définie plus précisément dans leur TAD ce qui donne l'avantage de ne pas encombrer le réseau de Petri et de le laisser plus compréhensible et intuitive.

2. MOTIVATION

Pour démontrer l'avantage de l'utilisation des ECATnets et leurs apports dans la modélisation du comportement des agents mobiles, et après l'étude de plusieurs systèmes d'agents mobiles on a choisi d'utiliser le système d'administration à distance des Firewalls NetFilter. Le problème posé est que son administration se fait manuellement par l'administrateur de sécurité qu'il doit se retrouver à proximité de ces consoles à chaque fois qu'il faut appliquer un changement dans les règles de sécurité, ce qui rend l'administration des Firewalls plus difficile surtout dans les réseaux à grande échelle. Le système d'agents mobiles proposé est constitué d'un ensemble d'agents intelligents, autonomes et mobiles ont la capacité de migrer de la machine origine vers le Firewall pour effectuer différentes tâches telles que la configuration, la mise à jour et la surveillance.

Le choix du système a été basé sur plusieurs critères, dont le plus important été d'approfondir les connaissances durant cette thèse dans le domaine de l'informatique et spécialement dans les domaines des systèmes d'exploitation, de l'open source et de la sécurité informatique, cette dernière est considérée actuellement comme la première préoccupation dans tous les systèmes informatique.

Cependant, NetFilter est un Firewall open source qu'il offre plusieurs avantages par rapport à d'autres Firewall, notamment son prix, si on compare son prix avec celui des solutions proposées sur le marché des firewalls on trouve une différence qui dépasse les dix fois le prix, ainsi que la facilité de mise à jour du NetFilter et la possibilité d'implémenter des nouvelles fonctionnalités dans le Firewall, ce qui est considéré comme impossible dans les boites noir acheté du marché, et particulièrement pour consolider la sécurité des entreprises et spécialement dans les domaines où la sécurité des informations est considérée comme une préoccupation critique, dont on ne peut pas implémenter des boites noir qui peuvent nuire à la sécurité, mais on doit utiliser des solutions dont leurs fonctionnement est connue en détails.

Ainsi que la puissance du NetFilter, sa facilité d'installation de configuration et de déploiement, et spécialement la disponibilité de son code m'ont poussés de choisir ce système et non pas d'autres.

Chapitre II :

Les Firewalls et NetFilter

- Introduction
- Définition
- Principales Types de filtrage
- Intérêts et limites du Firewall
- L'open source et la sécurité
- Firewall/NetFilter
- Capacité de filtrage du NetFilter/Iptables

Introduction

Actuellement, l'enjeu principal des réseaux, et en particulier des réseaux connectés à Internet, du point de vue de la sécurité, est de se préserver des attaques externes et internes. Ces attaques peuvent nuire au maintien de la confidentialité, de l'intégrité et de la disponibilité du réseau et/ou des données qui y cheminent. Il existe donc différents systèmes qui permettent d'accroître le niveau de sécurité d'une infrastructure réseau, mais malheureusement aucun système de protection n'est parfait et il existe des limites et des contournements possibles de ces systèmes. Des versions de ces systèmes de protection sont proposées commercialement par différentes sociétés ou organisations, sous forme propriétaire ou libre. Les firewalls sont un de ces systèmes de protection. De nos jours, les recherches sont toujours en cours pour optimiser les systèmes actuels ou trouver de nouvelles solutions de protection.

1 DEFINITION

Un *firewall* (pare-feu en français), est un système physique (*appliance*) ou logique (*logiciel*) servant d'interface entre un ou plusieurs réseaux afin de contrôler circulation des paquets de données, neutraliser les tentatives de pénétration en provenance de l'extérieur et maîtriser les accès vers l'extérieur. Pour cela, les pare-feux analysent les informations contenues dans les couches 3, 4 et 7 du modèle OSI et contrôlent ainsi les informations en transit.

Pour une appliance, il s'agit d'une machine physique comportant au minimum deux interfaces réseau : une interface pour le réseau à protéger (réseau interne) et une interface pour le réseau externe (domaine dangereux : réseau extérieur ou Internet).

Dans le cas d'un ordinateur équipé d'un logiciel pare-feu, celui-ci contrôle tous les accès du système à l'interface réseau externe (on parle de Firewall personnel pour un poste client ou basé hôte pour un serveur). On retrouve les mêmes domaines réseaux dont le réseau interne est réduit au système d'exploitation de l'ordinateur.

Les Firewalls matériels peuvent donc posséder une interface supplémentaire vers un réseau à part, la zone démilitarisée (**DMZ**). Le filtrage du trafic est effectué à travers un ensemble de règles prédéfinies par l'administrateur système. Ces règles sont différentes en fonction des interfaces, des protocoles et des sens de circulation.

On distingue deux principales politiques de sécurité au niveau des Firewalls :

- Interdire tous les échanges qui n'ont pas été explicitement autorisés (solution la plus sûre),

- Interdire les échanges qui ont été explicitement interdits.

2 PRINCIPALE TYPE DE FILTRAGE

2.1 Le filtrage de paquets

Les premiers firewalls étaient les routeurs eux-mêmes. Ceux-ci s'appuyaient donc sur du filtrage de paquets IP, à l'aide d'une analyse des entêtes des datagrammes en transit. Les firewalls peuvent effectuer du filtrage, sur la couche 3 du modèle OSI, basé sur les adresses IP source ou destination, on parle donc de filtrage par adresse (*address filtering*), ou peuvent effectuer du filtrage sur les protocoles de transmission, on parle alors de filtrage par protocole (*protocol filtering*). Enfin le Firewall peut être configuré pour bloquer tout le trafic arrivant sur certains ports afin d'empêcher des postes extérieurs d'accéder à des services non indispensables de l'extérieur.

Un Firewall possédant plusieurs interfaces, dont une DMZ, peut affiner ces règles de blocage pour les réorienter vers l'interface réseau adéquate (ex. : le port 80 vers le serveur WEB de la DMZ).

2.2 Le filtrage dynamique

Il existe des services comme le FTP qui utilisent un port statique pour la connexion, puis des ports dynamiques (au-dessus des ports réservés pour les services standards <1024) pendant les transferts. Il existe donc un système de filtrage dynamique de paquets (stateful inspection) basé sur l'inspection des couches 3 et 4 du modèle OSI. Il permet d'effectuer un suivi des transactions entre le client et le serveur et donc d'assurer la bonne circulation des données de la session en cours. Un paquet qui n'appartiendrait pas à une session encore active sera soumis au filtrage prédéfini. Le filtrage dynamique est plus performant que le filtrage de paquets standard mais ne protège pas contre les failles applicatives (failles liées aux logiciels).

2.3 Le filtrage applicatif

Le dernier type de filtrage utilisé est au niveau applicatif (couche 7 du modèle OSI). Il permet de contrôler les communications au niveau des applications (analyse application par application). Ce filtrage nécessite une connaissance parfaite de la structure des données échangées par les applications. Un firewall effectuant un filtrage applicatif est appelé *passerelle applicative* car il permet de relayer des informations entre deux réseaux en

effectuant un filtrage fin au niveau du contenu des paquets échangés. Ce type de filtrage est très performant et assure une bonne protection du réseau mais nécessite une grande puissance de calcul au niveau du Firewall. Sur des réseaux de taille importante, l'utilisation d'un tel Firewall est difficile à mettre en place et ralentit les communications.

Enfin, certaines applications propriétaires ou non standards ne permettent pas la mise en place de ce type de firewalls.

3 INTERETS ET LIMITES DU FIREWALL

3.1 Avantages

– Avec une architecture réseau cohérente, on bénéficie d'une centralisation dans la gestion des flux réseaux.

– De plus, avec un plan d'adressage correct, la configuration du Firewall est peu ou pas sensible au facteur d'échelle (règles identiques pour 10 comme 10000 équipements protégés).

– L'utilisation de la journalisation offre une capacité d'audit du trafic réseau et peut donc fournir des traces robustes en cas d'incident, si le Firewall n'est pas lui-même une des cibles.

– Le Firewall permet de relâcher les contraintes de mise à jour rapide de l'ensemble d'un parc en cas de vulnérabilité sur un service réseau : il est possible de maintenir une certaine protection des équipements non vitaux au prix de la dégradation du service avec la mise en place d'un filtrage.

3.2 Inconvénients

– La capacité de filtrage d'un équipement dépend de son intégration dans le réseau mais le transforme en goulet d'étranglement (capacité réseau et ressources du firewall).

– De par sa fonction, le Firewall est un point névralgique de l'architecture de sécurité avec de fortes contraintes de disponibilité. Il existe des solutions permettant la synchronisation de l'état des firewalls, mais beaucoup de configurations reposent encore sur un équipement unique.

– Enfin, une bonne gestion d'un pare-feu nécessite la compréhension des protocoles filtrés surtout lorsque les interactions deviennent complexes comme dans les cas FTP, H323,. . . avec le transport de paramètres de connexion dans le segment de données. De plus il

apparaît bien souvent des effets de bord liés aux diverses fonctions (couches réseaux filtrées, traduction d'adresses) et influencées par l'ordre d'application des règles.

4 L'OPEN SOURCE ET LA SECURITE

L'Open Source est un logiciel libre de droit et son code source est dans tous les cas disponible et même modifiable en fonction de la licence du logiciel. Les entreprises, y compris les plus grandes, n'hésitent plus, en effet, à recourir à des solutions issues de la communauté *open source* pour sécuriser leur système d'information.

Les logiciels libres présentent beaucoup d'avantages on peut citer les suivants :

• Les caractéristiques du logiciel libre sont un atout en sécurité.

• Le code source des logiciels libre est bien supérieur aux logiciels propriétaire tant au niveau de sa modularité, de sa lisibilité et de sa qualité.

• Il est également moins complexe et bien documenté.

• Il est réalisé sans contraintes de temps et financières et il respecte les standards.

• Le fait que le code source soit ouvert permet bien sur de corriger les failles de sécurité éventuelles du logiciel qu'un développeur externe aura détecté.

• Il est très facile de contacter l'auteur du logiciel afin de lui demander d'implémenter certaines fonctionnalités de sécurité qui pourraient être utiles.

• Le code source ouvert nous donne la possibilité de savoir comment se déroule le fonctionnement de ce logiciel et ainsi d'évaluer son efficacité.

• le logiciel libre et de par son code source ouvert permet facilement de détecter ses failles de sécurité, cela provoque les pirates à les exploiter mais aussi les développeurs de ces logiciels de les corriger plus rapidement.

• Le faible coût pour la mise en place de ce type de logiciel par rapport aux logiciels propriétaires ou aux architectures matérielles pousse les entreprises de moyens limités à se recourir à ces solutions.

5 FIREWALL/NETFILTER

Le Firewall est un dispositif de filtrage des paquets réseau qui fonctionne au niveau de la couche 3 et 4 (réseau et transport) et parfois la couche 7 (application) du modèle OSI. Ce dispositif, matériel ou logiciel, sert d'interface entre le réseau interne (à protéger) avec le

réseau externe, typiquement sur la passerelle, afin de contrôler le trafic entrant et sortant selon des règles définies par la politique de sécurité de l'entreprise. Les systèmes de règle se définissent par deux grands principes s'excluant mutuellement [14] :

- soit tout autoriser sauf ce qui est explicitement interdit,
- soit tout interdit sauf ce qui est explicitement autorisé.

NetFilter est une solution complète de Firewall implémentée dans le noyau Linux à partir de la version 2.4 destinée à contrôler tous les échanges de données réseaux, sa tâche est de faire le "Filtrage des Paquets Réseau", il se place entre la couche réseau du kernel Linux et la couche applicative [11,12]. Ce Firewall filtre les paquets de manière dynamique en tenant compte de l'état des sessions, et gère la translation d'adresses et de ports ainsi que le marquage de paquets…etc. L'interface de commande "iptables" est le moyen de paramétrer et configurer NetFilter. Elle se lance dans "user space" avec les droits root [13].

Iptables manipule trois tables : la table *filter*, la table *nat* et la table *mangle*. Chaque table est formée de chaînes par défaut, auxquelles il faut rajouter celles que vous créez. Pour chaque chaîne, il faut définir une politique par défaut, puis rajouter des règles pour gérer les cas particuliers [13]. Le tableau suivant récapitule les trois principales tables avec ses chaînes:[12]

Filter : cette table sert à filtrer les paquets réseaux	**INPUT**	c'est la chaîne par laquelle passent tous les paquets entrant par une interface.
	FORWARD	c'est la chaîne par laquelle transitent les paquets qui traversent la machine d'une interface à une autre.
	OUTPUT	c'est la chaîne par laquelle passent les paquets qui sortent des applications.
Nat (Network Adress Translation, ou Traduction d'Adresses Réseau)	**PREROUTING**	chaîne qui permet de faire de la translation d'adresse de destination.
	POSTROUTING	C'est grâce cette chaîne que vous pourrez faire du masquerading (partage de connexion) et faire croire à tous sur Internet que votre réseau n'a

		qu'une unique IP, celle de la passerelle.
	OUTPUT	Celle-ci va permettre de modifier la destination de paquets générés localement (par la passerelle elle-même).
Mangle : permet de marquer et/ou modifier des paquets à la volée	**PREROUTING**	chaîne qui permet de faire de la translation d'adresse de destination.
	INPUT	c'est la chaîne par laquelle passent tous les paquets entrant par une interface.
	FORWARD	c'est la chaîne par laquelle transitent les paquets qui traversent la machine d'une interface à une autre
	OUTPUT	c'est la chaîne par laquelle passent les paquets qui sortent des applications.
	POSTROUTING	C'est grâce cette chaîne que vous pourrez faire du masquerading (partage de connexion) et faire croire à tous sur Internet que votre réseau n'a qu'une unique IP, celle de la passerelle.

La syntaxe générale d'une commande iptables est la suivante :

iptables -t <*table*> <*action*> <*chaine*> <*option*s> -j <*cible*>

où les cibles correspondent aux actions qu'on va appliquer à un paquet lorsque ce-ci satisfait les conditions d'une règle [12].

Cibles	Action
DROP	Le paquet est détruit purement et simplement
REJECT	Le paquet est détruit, mais l'expéditeur en est informé.
ACCEPT	Le paquet est autorisé à continuer dans le traitement de la pile IP. Mais une autre règle située après la règle qui a accepté ce paquet peut très bien finalement décider de le supprimer.
LOG / ULOG	Le paquet est autorisé à continuer de passer, mais ses

	caractéristiques sont notées au passage.
MASQUERADE	Le paquet va être modifié, afin de dissimuler (de masquer en fait) certaines informations concernant son origine.
SNAT	L'adresse IP source est modifiée.
DNAT	L'adresse IP de destination est modifiée.
MARK	Le paquet est marqué en y attachant une information
Une chaîne utilisateur	Le paquet sera passé à une autre chaîne, définie par l'utilisateur.

Les paquets qui passent à travers le Firewall sont examinés par le module NetFilter selon le diagramme suivant [16]:

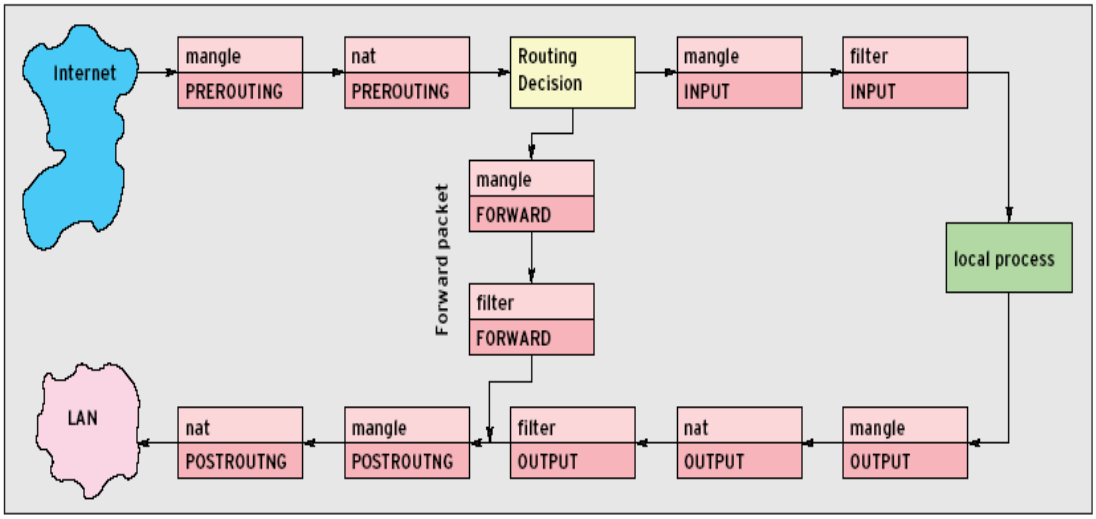


Figure II.1: diagramme présente le cheminement des paquets

Lorsque un paquet arrive, peu importe par quelle interface il entre, il passe d'abord par la fonction de routage qui détermine si le paquet est destiné aux processus locaux de cette machine ou à un hôte d'un autre réseau. Ensuite il est soumis à un ensemble de chaînes contenant elle-même des règles permettant de décider du sort du paquet selon la stratégie suivante :

• Si le paquet est destiné à l'hôte local :

- Il traverse la chaîne INPUT des tables Mangle et Filter.

- S'il est accepté, il est transmis aux processus locaux qui l'attendent

- Les paquets générés localement et avant leur transmission ils traversent la chaîne OUTPUT des tables Mangle, Nat, et Filter

- S'il n'est pas rejeté il est transmis vers la sortie en passant d'abord par la chaîne POSTROUTING des tables Mangle et Nat.

• Si le paquet est destiné à une machine d'un autre réseau :

- Il traverse la chaîne FORWARD des tables Mange et Filter

- S'il n'est pas rejeté il poursuit sa route selon sa destination

6 CAPACITE DE FILTRAGE DU NETFILER/IPTABLES

• Filtrage par adresse IP source ou destination et adresse MAC.

• Filtrage par protocole des niveaux 2 et 3 (tcp, udp, icmp). • Filtrage par service ou numéro de port (ftp, http, smtp…..).

• Filtrage par interface réseau.

• Filtrage des messages ICMP par type et code (echo-request, echo-reply….).

• Filtrage par flags ou drapeaux.

• Filtrage des paquets fragmentés.

• Filtrage S*tatefull* de paquets en fonction de l'état de la connexion NEW, ESTABLISHED ou RELATED (nouvelle, déjà établie ou relié) ce qui permet de faire un filtrage très fine.

• Limiter de manière efficace certains types d'attaques comme le SYN-flood, smurfing, ping of death…..etc

• La translation d'adresse NAT et le partage de connexion Internet.

• Marquage de paquets.

Chapitre III :

Etude de Cas

- Structure de l'environnement réseau
- Présentation du système d'agents mobiles
- Modélisation du système

1 STRUCTURE DE L'ENVIRONNEMENT RESEAU

Dans ce qui suit, on présente d'abord la structure de l'environnement réseau sur lequel on va appliquer notre solution. L'élément de protection est une machine linux 2.4 sur laquelle est installé le couple logiciel NetFilter/iptables, cette machine est connecté à l'Internet soit via un modem ADSL ou par autres équipements d'interconnexions et joue le rôle de passerelle.

Si une entreprise souhaite rendre certains services publics, ou d'héberger localement son site web, ces services sont hébergés sur des machines appelées serveurs semi-publics qui sont disposés dans une zone appelée DMZ ou (Zone **D**é**M**ilitarisée). Ces serveurs doivent être protégés derrière le Firewall.

Le réseau privé local est constitué des stations de travail clients (Windows ou Linux) et des serveurs privés (serveur d'imprimante ou de fichiers par exemple). Les clients accèdent donc à Internet de manière transparente, tout en étant protégés par le Firewall, ceci doit être configuré de telle sorte à respecter la politique de sécurité définie par l'administrateur de sécurité, la politique la plus conseillée, est de tout interdire sans exception, et ensuite d'accepter quelques connexions.

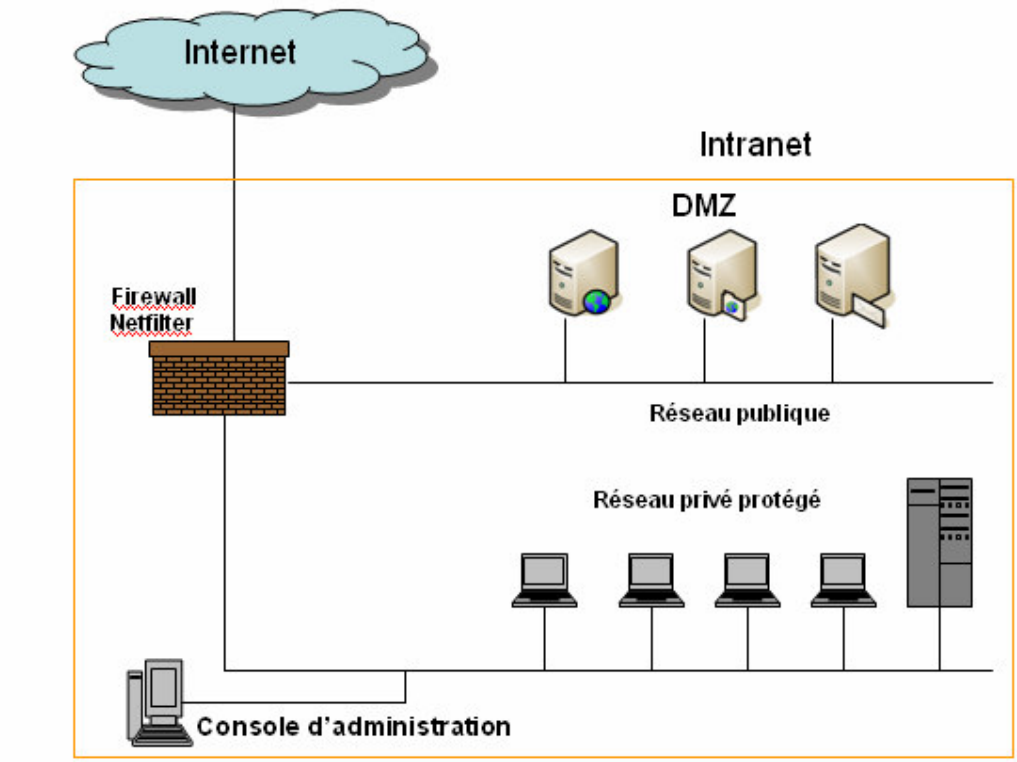


Figure II.2: présentation de structure de l'environnement

L'administrateur ne se trouve pas forcément à la proximité de la station qui fait office de Firewall lorsque il doit appliquer la politique de sécurité au Firewall, en effet si de nouveaux besoins de sécurité apparaîtrent et qu'on doit périodiquement changer la politique de Firewall, cela nécessite une action urgente de la part de l'administrateur de la sécurité.

Grâce à la technologie d'agents mobiles, l'administrateur n'a pas à se soucier de se déplacer à l'endroit où se trouve la machine du Firewall, en effet la tâche d'administration des Firewall est déléguée aux agents mobiles qui se déplacent à la place de l'administrateur pour la mise en place des Firewalls désirés. Peu importe l'endroit où se trouve l'administrateur dans son bureau ou son domicile, son poste de travail doit être connecté au réseau concerné par diverse manière : réseau de télécommunication, réseau sans fil, réseau local.

2 PRESENTATION DU SYSTEME D'AGENTS MOBILES

L'architecture d'une application de mise en place et d'administration à distance des Firewalls sous Linux, fondée sur la technologie d'agents mobiles, tend à réduire la difficulté et les inconvénients de la méthode traditionnelle. Elle décompose l'application en fonctions affectées à plusieurs agents qui travaillent en collaboration pour assurer le fonctionnement global. Ces agents sont: l'agent maître, l'agent générateur, l'agent de configuration, l'agent de mise à jour, l'agent de surveillance, l'agent de journalisation.

2.1 Architecture détaillée de système d'agent

Agent maître: c'est un agent stationnaire qui s'exécute sur son environnement d'origine et ne se déplace pas, son rôle est de créer et contrôler les autres agents esclaves et leurs spécifie les tâche à effectuer, ainsi il joue le rôle d'intermédiaire entre la GUI (interface utilisateur) et les autres agents.

Agent générateur: son rôle est de générer toutes les règles iptables correspondantes aux paramètres entrés et les stocker ensuite dans une base de données locale.

Agent de configuration : c'est lui qui se charge réellement de la mise en place du firewall, il récupère les règles de filtrage de la base de données, introduire les règles dans un script et lancer son exécution.

Agent de mise à jour: son rôle est de modifier la politique d'un firewall existant en ajoutant ou en supprimant quelques règles iptables.

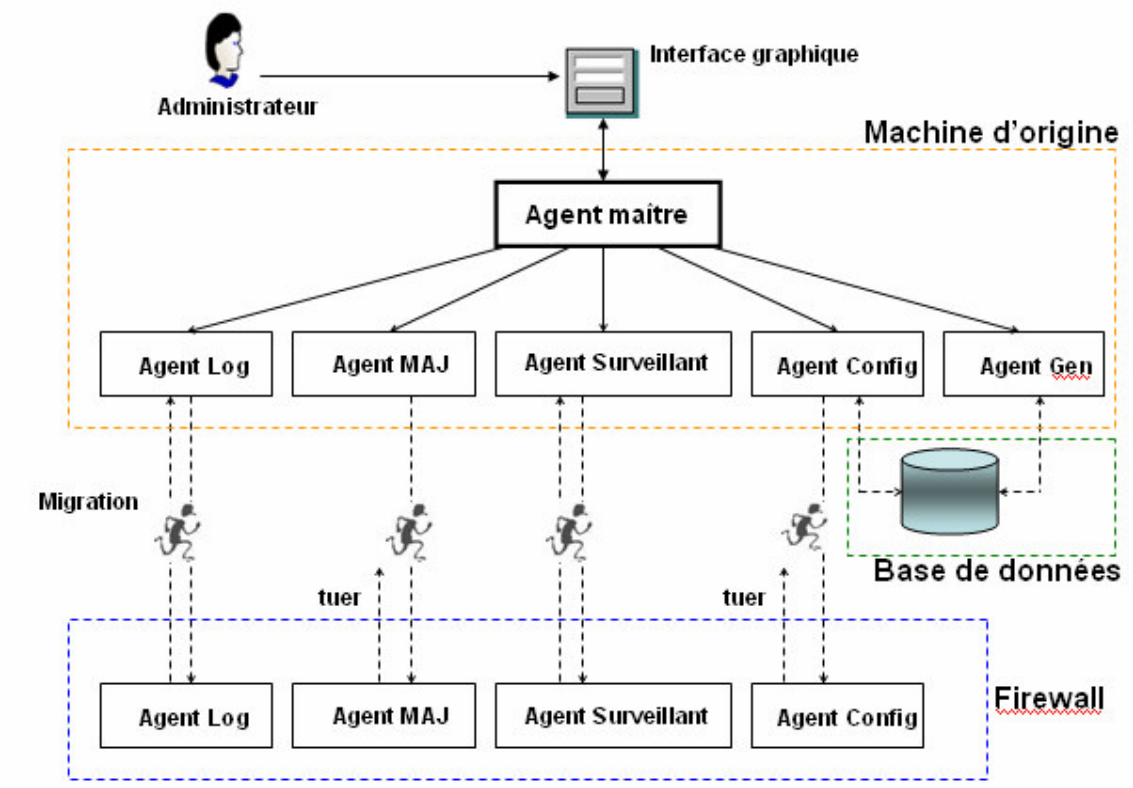


Figure II.3: structure de systèmes d'agents mobiles

Agent de surveillance : comme son nom l'indique il a pour rôle de surveiller en permanence le firewall contre les éventuelles attaques, il vérifie si le script du firewall a été modifié par un intrus et alerte l'administrateur.

Agent de journalisation: Cet agent se déplace vers la machine firewall et récupère les informations du journal, à son retour les informations collectées sont affichées à l'administrateur.

2.2 Principe de fonctionnement

A travers une interface graphique l'administrateur définit l'ensemble de paramètres nécessaires pour définir l'ensemble de règles à implémenter sur le Firewall. Après avoir validé ces paramètres l'agent maître qui est statique déclanche la création de l'agent générateur et l'initialise, ce dernier génère toutes les règles *iptables* et migre au serveur de base de données pour les enregistrer, ensuite il retourne à sa machine d'origine et en suite il se met en sommeil.

Lors de l'arrivée de l'agent générateur, l'agent maître reçoit un message de confirmation de génération de règles et leurs enregistrements celui-ci crée un agent de configuration. Cet agent récupère les règles et migre à la machine où on veut placer le

Firewall. A la destination il met ces règles dans un script qui sera par la suite exécuté par le biais de service *iptables*.

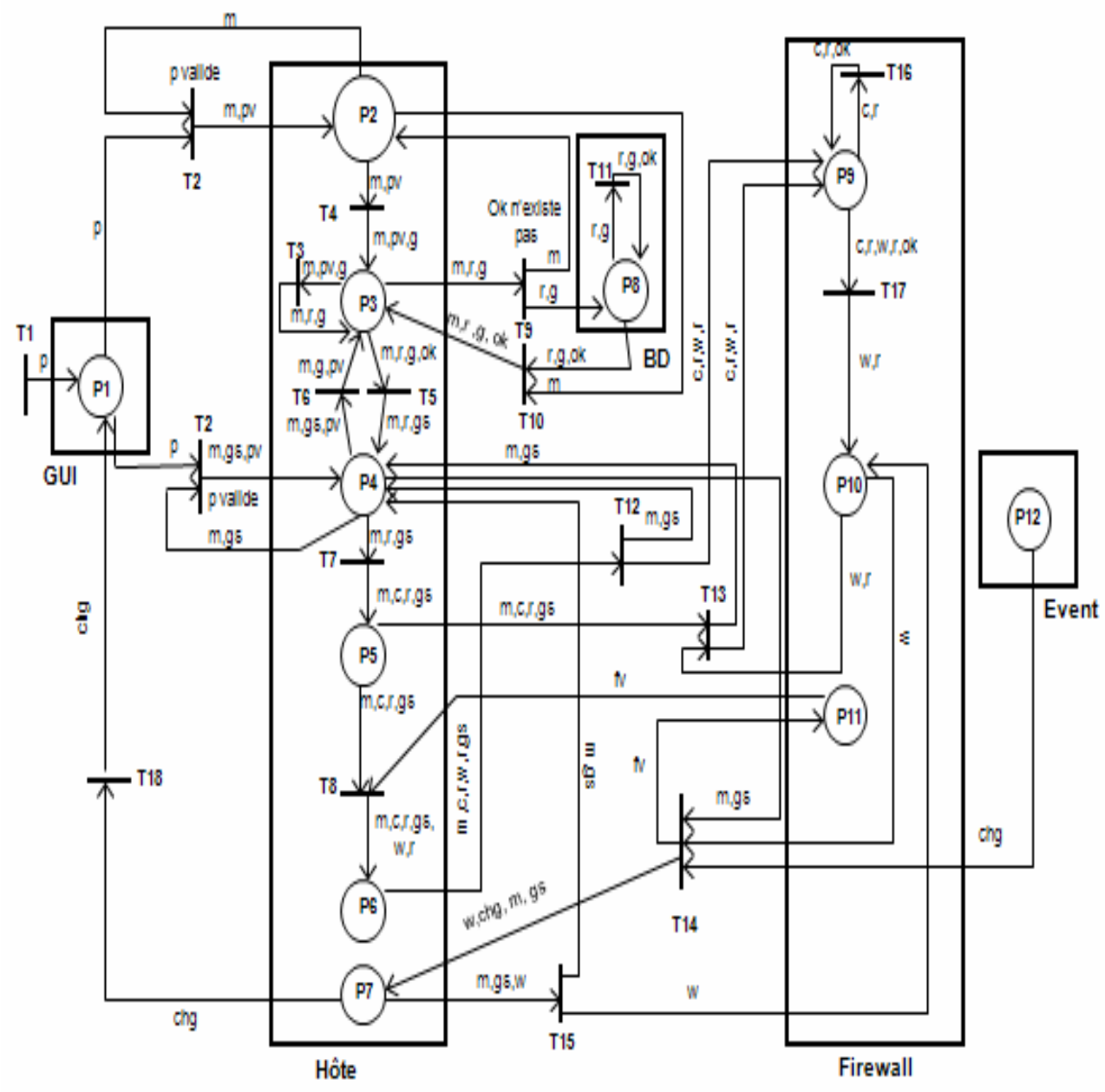
Lorsque le Firewall est configuré, l'agent de configuration sera tué et un autre agent dit agent de surveillance sera créé et déplacé au Firewall pour vérifier si le script a été modifié par un intrus, si c'est le cas il retourne à son hôte d'origine pour alerter l'administrateur.

Souvent les paquets louches sont ignorés et éventuellement enregistrés par le système de journalisation. A la demande de l'administrateur des informations de journal, l'agent maître crée un agent qui se déplace à la machine Firewall, récupère les journaux et à son retour les affiche à l'administrateur.

3 MODELISATION DU SYSTEME

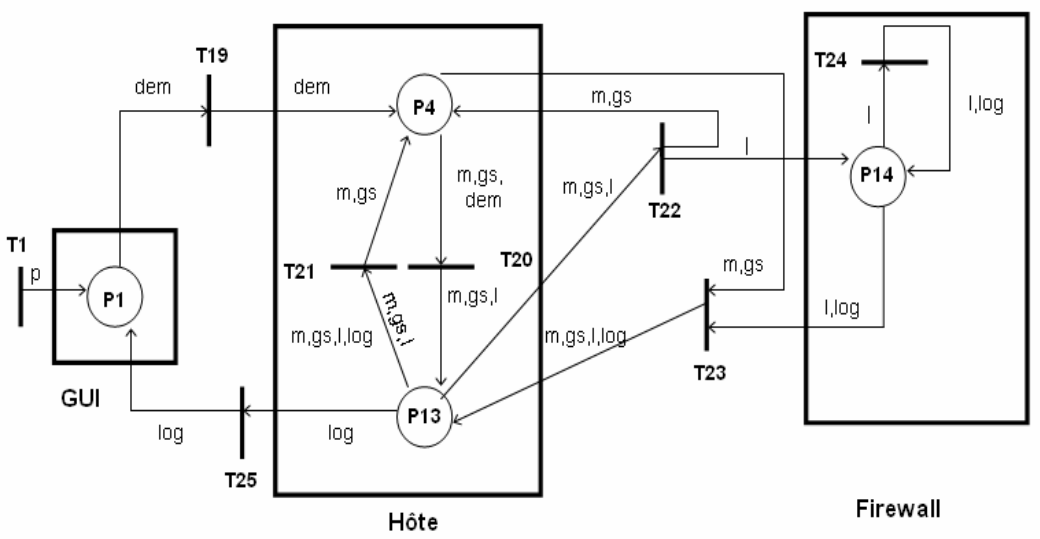
Étant donné qu'on a modélisé la mise en place et l'administration à distance d'un seul Firewall, notre système est constitué donc de trois machines seulement: la machine origine des agents, la machine Firewall, et la machine serveur de base de données. Chaque machine est modélisée par un ensemble de places et de transitions, où les places représentent l'état de la machine (les agents existants dans chaque machine et leurs états: en sommeils ou actifs), et les transitions représentent le comportement d'agents dont la mobilité est bien illustrée par les transitions de migration entre machines (T9, T10, T12, T13, T14, T15 dans la figure ci-dessous). On note que les jetons du réseau de Petri représentent les agents du système d'agents mobiles où chaque agent est un type abstrait de données.

On note aussi que si les inscriptions IC(p, t) et DT(p, t) sont identiques, alors uniquement IC(p, t) sera présentée sur l'arc (p, t) pour plus de lisibilité.



FigureII.4: ECATNet modélisant le fonctionnement de système d'agents mobiles

Pour plus de lisibilité de réseau du système, l'agent de configuration est modélisé à part



FigureII.5: ECATNet modélisant l'agent de journalisation

Les Places de l'ECATNet

Les places représentent l'état du système : quels sont les agents existants à chaque machine (origine, Firewall, et la machine interface) ainsi l'état de chaque agent (s'il est actif ou en sommeil).

P1 : Une place représente la machine interface qu'elle sert à valider les paramètres entrés par l'administrateur.

P2 : Indique qu'on est dans la machine origine et on a seulement l'agent maître en état actif avec ou sans paramètres valides.

P3 : La machine origine, les agents existants sont l'agent maître et l'agent générateur en état actif.

P4 : la machine origine, l'agent maître actif et l'agent générateur en sommeil.

P5 : la machine origine, l'agent maître actif, l'agent générateur en sommeil, avec l'agent de configuration actif.

P6 : la machine origine, l'agent maître actif, l'agent générateur en sommeil, l'agent de configuration actif, et l'agent de surveillance actif.

P7 : la machine origine, l'agent maître actif, l'agent générateur en sommeil, et l'agent de surveillance actif.

P8 : Représente la machine base de données.

P9 : La machine Firewall, l'agent de configuration actif, l'agent de surveillance actif.

P10 : La machine Firewall avec l'agent de surveillance actif.

P11 : Une place représente la machine Firewall où aucun agent n'est existant.

P12 : Une place représente une machine extérieure la source d'une attaque.

P13 : La machine origine avec l'agent maître actif, l'agent générateur en sommeil, et l'agent de journalisation.

P14 : La machine Firewall avec l'agent de journalisation en état actif.

Les transitions de l'ECATNet

T1 : Entrer les paramètres.

T2 : Envoyer les paramètres à l'agent maître.

T3: Générer les règles**.**

T4 : Créer l'agent générateur.

T5 : Mise en sommeil de l'agent générateur.

T6 : Réveiller l'agent générateur.

T7 : Créer l'agent configurateur.

T8 : Créer l'agent de surveillance.

T9 : Migrer au serveur de base de données.

T10 Retour à la machine origine.

T11 : Enregistrer les règles dans la base de données.

T12 : Migrer le configurateur et le surveillant au Firewall.

T13 : Migrer le configurateur et le surveillant au Firewall

T14 : Migrer l'agent de surveillance du Firewall vers l'hôte.

T15 : Retour de l'agent de surveillance au Firewall.

T16 : Configurer le Firewall.

T17 : Tuer l'agent configurateur.

T18 : Alerter l'administrateur.

T19 : réception de la demande du journal.

T20 : Créer l'agent de journalisation.

T21 : Tuer l'agent de journalisation.

T22 : Migrer l'agent de journalisation au firewall.

T23 : Retour de l'agent de journalisation.

T24 : Récupérer le journal.

T25 : Afficher le journal**.**

Les Jetons

m : L'agent maître.

g : L'agent générateur de règles.

c : L'agent configurateur.

w : L'agent de surveillance.

l : L'agent de journalisation.

r : Règles générées.

gs : L'agent générateur en sommeil.

ok : Message de confirmation.

chg : Message indique un changement dans les règles.

p : Paramètres en entrée.

pv : Paramètres valides.

log : le journal.

dem : Un message de demande du journal.

fv : Firewall vide.

CONCLUSION GENERALE

Le paradigme agent mobile est apparu comme une extension du code mobile et de la migration de processus. Il marque plusieurs apports dans la gestion des systèmes complexes distribués, le plus important et de réduire la quantité de données transmises à travers un réseau pour une même tâche effectuée à distance. Cependant ces systèmes sont en effet difficiles à représenter d'une manière formelle, ce mémoire vise à donner une approche permet de représenter ces systèmes formellement dans le but de les bien analyser, valider et programmer. Notre approche se base sur le modèle de réseau de Petri dit ECATNets qui offre une symbolique simple facile à comprendre et une capacité d'exprimer les comportements parallèles et concurrents. IL se base sur les types abstraits algébrique qui présente une capacité de décrire abstraitement les données ainsi que la logique de réécriture qui définie la sémantique de réseau de Petri utilisé.

Pour rendre le réseau de Petri plus intuitive, on a divisé le réseau en plusieurs parties, dont chacune présente un endroit précis du système. En plus ce partage permettra de montrer les propriétés des agents mobiles les plus essentiels tel que l'autonomie, la réactivité et particulièrement la mobilité. Cependant l'utilisation des types abstraits algébriques pour la définition des agents donne une représentation abstraite aux agents ce qui facilite la compréhension et l'implémentation du système.

A la fin, pour bien illustrer les apports des ECATNets dans la modélisation des agents mobiles, on a démontré l'approche proposée par la modélisation d'un système d'administration à distance des Firewalls basée sur le paradigme d'agents mobiles

REFERENCES

[1] M. Bettaz, M. Maouche. "How to specify Non Determinism and True Concurrency with Algebraic Term Nets" Lecture Notes in Computer Science, N 655, Spring Verlag, Berlin, p. 11-30, 1992.

[2] Noura Boudiaf, Kamel Barkaoui, Allaoua Chaoui, Implémentation des règles de réduction des ECATNets dans MAUDE, 6e Conférence Francophone de MOdélisation et SIMulation - MOSIM'06 - du 3 au 5 avril 2006 –Rabat – Maroc « Modélisation, Optimisation et Simulation des Systèmes : Défis et Opportunités ».

[3] K,Djemame, ECATNets with Limited Capacity Places : a Distributed Simulation Study, 18th UK PerformanceEngineeringWorkshop-UKPEW'02-, University of Glasgow, 10-11July2002.

[4] Noura Boudiaf, Développement des Outils Basés Maude pour les ECATNets Domaine d'Application : Analyse des Programmes Ada, soutenue le 10 avril 2006, Constantine.

[5] Sylvain Goutet, Conception d'une architecture Multi-Agents supportant des agents mobiles intelligents, Mémoire de maîtrise (M.Sc.A.) en Laborataoire de Recherche en Réseautique et Informatique Mobile-LARIM-, AVRIL 2001

[6] Elie Fute Tagne, Emmanuel Tonye, César Viho, Alain Akono, Modélisation d'une Architecture Multi-agents d'un Système de Télécommunication par Réseaux de Pétri, The Third International Conference On Gignal-Image Technologie& Internet-Based Systems (SITIS' 2007), Session SIPW1: Mobile signal processing, December 16-19 2007.

[7] Ali A. Pouyan, Steve Reeves, Behavioral Modeling for Mobile Agent Systems Using Petri Nets, 2004 IEEE International Conference on Systems, Man and Cybernetics.

[8] H. Tran Viet, Gestion De La Mobilité Dans L'ORB Flexible Jonathan, Conférence RIVF 03, Recherche informatique, Vietnam et Francophonie, Institut de la Francophonie pour l'Informatique (IFI), 10-13 février 2003, Hanoï.

[9] Steve Bernier, Conception et Implémentation basées sur des composants répartis d'une station terrestre virtuelle de communication satellite, Mémoire présentée en de l'obtention du grade de Maitre des Sciences (M.Sc), University Sherbrooke, Québeq, Canada, septembre 2000.

[10] Christophe CUBAT DIT CROS, Agents Mobiles Coopérants pour les Environnements Dynamiques, Thèse de doctorat, INPT, soutenue décembre 2005.

[11] József Kadlecsik KFKI RMKI, Gyrgy Pֽsztor SZTE EK, Netfilter Performance Testing.

[12] Olivier ALLARD-JACQUIN**,** Atelier NetFilter : Le Firewall Linux en action, support de cours publié sous la licence de Libre Diffusion de Documents (LLDD), 10 avril 2004.

[13] Hristo Valtchanov, Filtrage des paquets (Netfilter), Atelier ISOC de Formation Aux Réseaux Varna, 15 décembre - 18 décembre 2003.

[14] Jean Baptiste Favre, Firewall : Architecture et déploiement (NetFilter), support de cours, 26 mars 2006.

[15] Antony Stone, Stateful packet filter firewalling with Linux, UKUUG(the UK's Unix & Open Systems User Group) Thursday 5th to Sunday 8th August Leeds, West Yorkshire 2004.

[16] NetFilter L7, Article de LINUX-MAGAZINE issue 64, march 2006, page 66.

[17] Calvin Yun-Long Shenm, Behavior Modeling of Mobile Agent, University of California, Irvine, http://netresearch.ics.uci.edu/Previous_research_projects/agentos/doc/uci_journal.ps.

[18] Marek A. Bednarczyk, Luca Bernardinello, Wiesław Pawłowski and Lucia Pomello, Modelling mobility with Petri hypernets. Congrès Recent trends in algebraic development techniques : (Barcelona, 27-29 March 2004, revised selected papers) WADT : international workshop on recent trends in algebraic development techniques N°17, Barcelona , ESPAGNE (27/03/2004)

[19] Dianxiang Xu, Jianwen Yin, Yi Deng, and Junhua Ding, A Formal Architectural Model for Logical Agent Mobility, Transaction on Software Engineering, Vol. 29, No. 1, January 2003.