N° 19/ 2010 –M/ Inf

République Algérienne Démocratique et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumedien
Faculté d'Electronique et d'Informatique

Mémoire présenté

Pour l'obtention du diplôme de MAGISTER

En Informatique

Spécialité : Informatique mobile

Par Melle MADI Taous

Sujet :

Formalisme pour la modélisation du temps dans les systèmes mobiles

Soutenu publiquement, le 10 Juin devant le jury composé de :

Mr M.AHMED NACER	Professeur à l'USTHB,	Président
Mme M.BOUKALA	Professeur à l'USTHB,	Directrice de mémoire
Mme N. GHARBI	Maître de conférences / A l'USTHB,	Examinatrice
Mme S.MOUSSAOUI	Maître de conférences / A l'USTHB,	Examinatrice

A la mémoire de ma tante. Que Dieu lui accorde sa sainte miséricorde et l'accueille en son vaste paradis.

Remerciements

Louange à Allah, seigneur des mondes, et que la paix et le salut soient sur notre Prophète Mohammed, l'envoyé en clémence pour l'univers, ainsi que sur sa famille et sur tous ses compagnons.

En premier lieu, j'ai le plaisir d'dresser mes vifs remerciements au Prof. M. BOUKALA, mon directeur de thèse, pour m'avoir donné la possibilité de travailler sur un sujet aussi passionnant. Ses conseils et directives ont été un facteur déterminant pour le bon déroulement de ce travail.

Je tiens à exprimer mes sincères remerciements aux membres de l'équipe MEPQOS pour leur écoute et leurs précieuses indications.

Je remercie vivement les membres du jury : Prof. AHMED NACER, Dr. N.GHARBI et Dr. S. MOUSSAOUI pour le temps et l'effort investis afin de juger ce travail.

Je réserve une attention particulière à mes amies et collègues de travail pour leur encouragement et soutien. J'estime avoir beaucoup de chance de les avoir rencontrés.

Mes plus vifs remerciements à mes parents dont la tendresse et la patience m'a apporté la motivation nécessaire pour aller jusqu'au bout. Je ne leur serai probablement jamais assez reconnaissante.

Sommaire

Table des figures

Résumé

Les avancées technologiques connues ces deux dernières décennies, la prolifération des techniques de communication d'une part et le progrès qu'a connu la microélectronique d'autre part, ont conduit à l'apparition d'environnements étendus hautement dynamiques et manifestant de nouvelles caractéristiques (capacités variables, opérations de déconnexion fréquentes des dispositifs mobiles, …). Du point de vue logiciel, ces évolutions ont amené à l'exploration de nouveaux paradigmes de conception des applications réparties, s'adaptant le mieux au nouveau contexte physique servant de base à l'exécution de celles-ci. Ces paradigmes explorent le concept du *code mobile*. Cette notion de mobilité logique a produit de nouveaux styles d'interaction qui diffèrent significativement du motif d'interaction traditionnel Client/ Serveur, elle a également crée de nouveaux défis dans toutes les phases du cycle de développement d'un système, telles que la modélisation, l'exécution, la vérification, …etc. Dans notre travail, nous nous intéressons aux formalismes de description dédiés à la modélisation des systèmes intégrant ce concept de mobilité logique, les systèmes à base d'agents mobiles en particulier. On qualifiera ces systèmes de *mobiles*.

Plusieurs formalismes ont été proposés pour modéliser les systèmes mobiles et raisonner sur les différents aspects liés à la mobilité. Ces formalismes peuvent être classifiés en deux grandes catégories : Approches basées sur les algèbres de processus et approches basées sur les réseaux de Petri.

Dans notre étude, nous procédons d'abord par une analyse des caractéristiques fondamentales qui doivent être exhibées par les formalismes dédiés à la description des systèmes mobiles. Nous présentons par la suite, une étude des formalismes les plus pertinents qui existent dans la littérature. A l'issue de cette étude et sur la base des caractéristiques présentées, nous choisirons le modèle sur lequel portera notre contribution.

Dans une seconde partie du travail, nous proposons une extension du modèle choisi par le facteur temps. Le formalisme résultant fournit un support pour la modélisation des systèmes mobiles soumis à des contraintes temporelles.

Introduction générale

En 1949, la première ère de l'informatique a fait son apparition suite à la naissance du premier ordinateur. En 1971, la création du premier micro-processeur a offert de grandes perspectives à la miniaturisation du matériel informatique ce qui a conduit à la généralisation de son usage. Dès lors, l'adaptation des technologies de télécommunication à l'informatique fut rapidement incontournable afin de faciliter les échanges de données entre les ordinateurs sans avoir à utiliser des supports de stockage fastidieux (cartes perforées, disquettes, …). Cela a permis donc de créer des interconnexions de machines appelées *réseaux d'ordinateurs.*

Pour tirer partie au mieux de ces environnements connectés, le développement d'applications pouvant prendre en compte un ensemble de machines a été rapidement étudié. C'est ce type d'applications que nous qualifions de *réparties* ou *distribuées*. Celles-ci peuvent s'exécuter sur plusieurs machines, utiliser les ressources distribuées, mettre en place des mécanismes de tolérance aux fautes, …etc, ce qui permet d'aller bien au-delà du simple échange de données. Plusieurs méthodes ont été proposées pour la conception des applications réparties, nous citons à titre d'exemple les paradigmes de conception *pair à pair*, *publisher/ subscriber* et le paradigme de conception *client/ serveur* qui est sans conteste le plus populaire.

Avec les avancées technologiques connues ces deux dernières décennies, la prolifération des techniques de communication d'une part, et le progrès qu'a connue la microélectronique d'autre part, une nouvelle ère de l'informatique s'est révélée. On assiste à la mise en réseau massive d'une grande diversité d'entités communicant sur une échelle planétaire. On parle de l'ère du *global computing,* caractérisée par un fort degré de répartition des ressources d'exécution et des réseaux hétérogènes. Le résultat est un environnement hautement dynamique manifestant des capacités variables.

Cette nouvelle infrastructure a révélé de nouveaux scénarii et caractéristiques qui ont conduit à leur tour à l'apparition de nouveaux problèmes et difficultés liés au développement des applications réparties. Cela a promu l'étude de nouveaux paradigmes de conception et langages de programmation mettant en œuvre des mécanismes permettant de gérer ces caractéristiques nouvelles. Les travaux menés dans ce sens ont engendré un nouvel axe de recherche exploitant la notion du *code mobile*. Cette notion de *mobilité logique,* a produit des

styles d'interaction qui différent significativement du motif d'interaction traditionnel client/ serveur. Parmi ces paradigmes de conception, le *Code à la demande* est probablement le plus utilisé, les Applets dans le langage java en sont un exemple. Le paradigme des agents mobiles est cependant le plus défiant à cause de ses exigences. Il permet par ailleurs d'apporter des améliorations indéniables portant principalement sur le gain de performances.

Depuis les années 70, un travail de recherche considérable a été effectué sur les méthodes de vérification formelle pour garantir la fiabilité et la sécurité des systèmes informatiques généralement centralisés et de taille réduite. Les résultats obtenus ont depuis été appliqués avec succès pour concevoir, implémenter et valider des systèmes critiques dans des domaines comme l'avionique, le contrôle aérien ou les télécommunications. Etant donné un modèle d'un système exprimé dans un formalisme donné, les méthodes de vérification formelle consistent à examiner les propriétés comportementales du modèle établi afin de détecter des erreurs éventuelles et d'exhiber les propriétés du système et ce, en se basant sur des techniques de vérification rigoureuses. Dans notre étude, nous nous intéressons aux modèles permettant la description opérationnelle des systèmes.

Simultanément, les besoins en fiabilité et en sécurité sont devenus immenses avec la complexité grandissante des applications réparties et notamment avec l'introduction du concept de la mobilité logique. Malheureusement, les formalismes déjà développés s'avèrent inadaptés à ce type d'applications modernes du fait qu'ils apparaissent insuffisants pour relever les nouveaux défis posés : primitives d'interaction flexibles, support pour la mobilité, la sécurité, …etc. Le développement de modèles de description, de méthodes et d'outils de vérification et de validation formelle pour la réalisation d'applications exhibant à la fois la répartition et la mobilité, et qu'on qualifiera dans ce mémoire de *systèmes mobiles*, apparait donc toujours comme un problème largement ouvert.

Plusieurs travaux de recherche se sont orientés vers la proposition de nouvelles approches et de techniques permettant d'étendre les formalismes préexistants pour capturer les exigences exposées par les systèmes mobiles. Ces propositions peuvent être classifiées en deux grandes catégories : Approches basées sur les algèbres de processus et approches basées sur les réseaux de Petri.

L'axe de recherche basé sur les algèbres de processus a bénéficié des premiers travaux portant sur la mobilité. En effet, les travaux de Milner, au début des années 90, ont donné naissance au premier calcul qui permet la modélisation de systèmes caractérisés par la

reconfiguration dynamique de l'environnement. Il s'agit du π-calcul [13] qui est une extension du CCS [12] (Concurrent Communicating Systems). Il a été suivi par la proposition d'une foison de calculs, à chacun ses objectifs et techniques pour le traitement de la répartition, la mobilité et les différents aspects de la sécurité. On cite à titre d'exemple, le calcul des ambients mobiles [24], le join calcul [22], Klaim [25], …etc.

L'axe de recherche basé sur les réseaux de Petri est relativement plus récent. Mobile nets [29], extension des réseaux de Petri colorés, est l'un des premiers modèles qui ont été proposés. Il s'inspire largement des techniques du π-calcul. Le paradigme nets within nets [34] a servi de base à plusieurs modèles définis pour la description des systèmes mobiles, Mobile EOS [38] et Petri hypernets [39] en sont des exemples. Le modèle Mobile Synchronizing Petri Nets [41] et son extension Abstract Mobile Synchronizing Petri Nets [42] constituent des variantes des réseaux de Petri colorés étiquetés qui s'inspirant des techniques de calculs de processus et qui se préoccupent en plus de l'aspect sécurité.

L'objectif principal de ce mémoire consiste à présenter une étude sur les formalismes qui existent dans la littérature pour la description des systèmes mobiles. Nous nous intéressons plus particulièrement aux modèles basés sur les réseaux de Petri pour les avantages multiples qu'ils présentent (communauté importante, base théorique solide, représentation graphique attrayante, …).

Ce mémoire tente d'apporter une contribution en s'intéressant à la catégorie des systèmes mobiles soumis à des contraintes temporelles. Cette contribution consiste à enrichir le modèle *Abstract Mobile Synchronizing Petri Nets* par une extension à la dimension temporelle. Le modèle résultant intitulé Timed *Abstract Mobile Synchronizing Petri Nets* est une classe spécale des réseaux de Petri colorés étiquetés temporisés.

Le mémoire est organisé comme suit : Le chapitre 1 est dédié à la présentation des systèmes mobiles. Le chapitre 2 est dévolu à la présentation des formalismes de description des systèmes mobiles et des caractéristiques essentielles qui doivent être supportées par ces modèles. Le chapitre 3 est consacré à la description détaillée du modèle *Abstract Mobile Synchronizing Petri Nets* sur lequel portera notre apport. Et enfin, le chapitre 4 est réservé à la présentation du modèle *Timed Abstract Mobile Synchronizing Petri Nets* que nous proposons. Nous terminons par une conclusion et des perspectives pouvant faire suite à ce travail.

Chapitre 1

Les systèmes mobiles

1.1. Introduction

Les avancées technologiques qu'a connu la microélectronique d'une part, et la prolifération de l'usage des technologies de communication sans fils d'autre part, ont conduit, du point de vue matériel, au changement de la vision statique des réseaux informatiques. Ceux-ci ont connu une croissance vertigineuse tant sur le plan de la taille que du trafic engendré et sont devenus intensivement dynamiques. Du point de vue logiciel, ces évolutions ont amené à l'exploration de nouveaux paradigmes de conception s'adaptant le mieux au nouveau contexte physique servant de base à l'exécution des applications réparties. Ces paradigmes explorent le concept de la mobilité de composants logiciels.

Dans la section 1.2 de ce chapitre, nous présentons les caractéristiques essentielles du nouveau contexte physique. Dans la section 1.3, nous étudions les paradigmes de conception basés sur le code mobile, le schéma des agents mobiles en particulier. Nous illustrons les éléments qui motivent l'adoption de ce paradigme dans les environnements dynamiques et à large échelle. Nous illustrons également les défis qu'il expose. Nous illustrons également les défis qu'il expose.

1.2. Caractéristiques des environnements dynamiques

A leurs débuts, les réseaux informatiques étaient constitués de machines immobiles dotées de puissances comparables et connectées par des liens de communication filaires dont les caractéristiques sont sensiblement équivalentes. Ces environnements sont caractérisés par une topologie statique particulière (anneau, bus, arbre, …) où le dynamisme est un phénomène rare qui se réduit aux pannes de sites et aux erreurs humaines, débranchement de câbles par exemple. Ainsi, ces réseaux filaires offrent aux concepteurs d'applications un environnement homogène et assez stable du point de vue évolution des connexions (apparition et disparition

des hôtes). Cette uniformité et nature stable facilite la gestion transparente des ressources en faisant abstraction de la structure du réseau [01]. Elle permet l'établissement et le maintien des communications distantes entre les différents hôtes. Et enfin, elle assure un comportement prévisible du réseau (temps de communication borné) donc offrant une certaine garantie de fiabilité [02].

Avec l'utilisation de plus en plus courante des technologies sans fil ces dernières années, les réseaux évoluent de plus en plus vers de nouvelles architectures dynamiques à large échelle. Les sites deviennent mobiles, apparaissent et disparaissent fréquemment et les connexions fluctuent de manière continue.

Les environnements dynamiques résultant de cette nouvelle tendance technologique présentent des caractéristiques spécifiques, certaines liées aux nœuds et à leur mobilité et d'autres au support de communication. Les dispositifs mobiles, bien qu'ils soient de plus en plus performants grâce à la technologie de miniaturisation, restent pauvres en ressources par rapport aux dispositifs fixes notamment en termes d'énergie. Les réseaux sont constitués par conséquent de nœuds caractérisés par des puissances et des capacités qui varient fortement. Par ailleurs, la mobilité des nœuds brise l'aspect statique des environnements, cet aspect qui se présente comme une caractéristique essentielle à certains schémas d'organisation classiques tels que le schéma client/ serveur. L'autre catégorie des problèmes est liée au support de communication peu fiable. En effet, les transmissions radio sur lesquelles se basent les communications sans fil sont caractérisées principalement par la bande passante limitée, les interférences et les erreurs de transmission.

Du point de vue logiciel, ces nouvelles architectures ont amené à reconsidérer les schémas organisationnels conçus sur la base des anciennes structures fixes [03]. Les paradigmes du *code mobile* , le schéma des agents mobiles en particulier, apparaissent comme une abstraction bien adaptée à la conception et l'implémentation des applications réparties destinées à être déployées sur les environnements intensivement dynamiques et à large échelle.

1.3. Introduction de la mobilité logique

Pour faciliter la construction des applications réparties, plusieurs styles architecturaux ont été établis. Néanmoins, le style le plus populaire est sans conteste le modèle Client/ Serveur. Dans ce schéma organisationnel, les interactions entre clients et serveurs peuvent

être distantes en mettant en œuvre des mécanismes permettant aux clients d'envoyer leurs requêtes aux serveurs, comme elles peuvent être conçues en introduisant le concept de mobilité auquel cas on aboutit à des variantes du modèle Client/ Serveur classique telles que le paradigme basé sur la migration de processus. Ce dernier est à la base d'un autre style organisationnel qui suscite un intérêt particulier dans le domaine des recherches portant sur les applications réparties, notamment celles destinées à être déployées dans des environnements dynamiques et à grande échelle [04], il s'agit du schéma des agents mobiles.

1.3.1. Paradigmes de conception classiques

Les applications réparties utilisent au cours de leur exécution un ensemble de ressources qui peuvent être distribuées sur différents sites du réseau sous-jacent. Pour l'accès et l'utilisation de ces ressources, plusieurs styles architecturaux ont été envisagés. On cite à titre d'exemple les paradigmes *pair à pair, abonnement / publication, mémoire distribuée partagée et le schéma Client/ Serveur* qui est sans conteste le plus populaire. Dans ce schéma, on sépare une fonctionnalité précise de celui qui l'utilise grâce à une répartition des éléments et à une communication à distance. En d'autres termes, une application rendant une certaine fonctionnalité (un service) est placée sur un nœud du réseau (le serveur) en attente de requêtes émises par d'autres applications (les clients) réparties sur le réseau. Ce schéma définit une manière de communiquer entre deux processus à l'aide d'un protocole préétabli. Lorsque qu'un client souhaite réaliser un service, il envoie une requête au serveur qui exécute le service et lui renvoie le résultat.

1.3.2. Paradigmes de conception à base du code mobile

Dans le cadre du modèle Client/ Serveur, trois composants élémentaires sont impliqués dans la réalisation d'un service : le *savoir faire* (code), les *ressources* nécessaires (applicables au code) et le *composant calculatoire* (responsable de l'exécution du code). L'exécution du service ne peut avoir lieu qu'après la réalisation d'un regroupement de ces trois éléments dans un même site après les avoir identifiés [07].

La construction d'un schéma Client /Serveur peut être fondée sur deux classes de paradigmes pour la réalisation du service convoité par le client : modèles d'exécution à

distance et modèles se basant sur le concept de mobilité grâce au déplacement d'un (ou plus) des composants impliqués pour la réalisation du service.

- Dans le modèle d'exécution à distance, les trois éléments définis pour un service (code, ressources et composant calculatoire) restent concentrés sur le serveur.
- L'autre manière de concevoir les interactions Client/ Serveur consiste à introduire le concept de mobilité. Ce concept peut être vu comme une nouvelle variante du modèle Client/ Serveur classique. Selon les composants du service qui se déplacent, on distingue trois formes de mobilité (voir tableau 1.1):
    - Envoi du savoir faire ou évaluation distante (Remote EValuation REV).
    - Récupération du savoir faire ou code à la demande (Code On Demande COD).
    - Migration de processus.

On considère un scénario où un composant calculatoire A (client) localisé au niveau d'un site S_A a besoin du résultat d'un service quelconque. On suppose aussi l'existence d'un autre site S_B éventuellement impliqué dans l'accomplissement du service [08] :

- Dans le paradigme Client/ Serveur classique, un composant calculatoire B (serveur) offrant un ensemble de services est placé dans le site S_B. Les ressources et le code requis pour l'exécution du service résident dans le site S_B également. Le composant client A, situé dans S_A, demande l'exécution d'un service par le biais d'une interaction avec le composant serveur B. En réponse, B réalise le service en exécutant le savoir faire adéquat en se servant des ressources nécessaires à la réalisation du service.
- Dans le paradigme REV, le composant calculatoire A dispose du savoir faire propre au service mais pas des ressources nécessaires. Celles-ci sont localisées au niveau du site S_B. Par conséquent, le composant client A envoie le savoir faire du service au composant calculatoire B localisé dans le site S_B. B exécute le service en utilisant les ressources disponibles. Les résultats sont livrés au composant A suite à une interaction.
- Dans le paradigme COD, les ressources requises pour la réalisation du service sont localisées au niveau du site S_A, mais il n'existe pas d'informations sur la manière dont ces ressources doivent être manipulées. Le composant A interagit alors avec B situé au niveau de S_B lui demandant le savoir faire associé au service. Une deuxième

interaction se produit lorsque B délivre le savoir faire à A qui peut par la suite l'exécuter.

- Dans le paradigme *migration de processus,* le composant A qui réside initialement dans le site S_A, dispose du savoir faire propre au service à réaliser, mais certaines des ressources dont il a besoin sont localisées au niveau du site S_B. A migre alors avec le savoir faire et éventuellement certains résultats intermédiaires vers le site S_B où il continue son exécution en se servant des ressources nécessaires. Dans ce paradigme, le processus est déplacé de serveur en serveur en fonction des ressources qu'il souhaite utiliser. L'un des exemples est l'exécution d'une requête SQL sur une suite de bases de données distinctes. La migration de processus est généralement basée sur une *migration forte* (déplacement du code et de l'intégralité de son état d'exécution) et *réactive* (invoquée par le système). Le processus exprime ses besoins en ressources, comme l'accès à un fichier ou à du temps processeur, et c'est le système qui va placer le processus demandeur sur la machine correspondant le mieux à ses besoins. Par exemple, le placer sur la machine où le fichier demandé est stocké.

Paradigme	**Avant**		**Après**	
	S_A	S_B	S_A	S_B
CS	A	Savoir faire Ressources B	A	Savoir faire Ressources **B**
REV	*Savoir faire* A	Ressources B	A	*Savoir faire* Ressources **B**
COD	Ressources A	*Savoir faire* B	*Savoir faire* Ressources **A**	B
Migration de processus	*Savoir faire* *A*	Ressources	-	*Savoir faire* Ressources ***A***

Tableau 1.1 : Le paradigme Client/ Serveur et ses variantes

Le tableau 1.1 montre l'emplacement des composants impliqués par le service à réaliser avant et après son exécution. Pour chaque paradigme, le composant calculatoire en gras est celui qui exécute le code. Les composants en italique sont ceux qui se sont déplacés.

1.3.3. Les agents mobiles

Le schéma à agents mobiles dérive de deux domaines différents : les agents venant des systèmes multi-agents et les systèmes distribués avec la migration de processus. Dans le cadre des systèmes multi-agents qui appartiennent à la base au domaine de l'intelligence artificielle, on désigne par *agent* des programmes qui possèdent une "intelligence" assez développée pour prendre des décisions face à des situations complexes. Ces agents sont répartis au sein d'un système distribué et communiquent entre eux via un medium de communication géré par le système pour réaliser la tâche pour laquelle ils ont été crées. D'une manière générale, le schéma à agents mobiles peut être considéré comme une généralisation de la migration de processus où le déplacement est à l'initiative même du code. On parle alors de migration *proactive* de processus. Ainsi, on définit les agents mobiles comme étant des agents ayant la capacité de se déplacer de site en site tout en ayant conscience de leurs déplacements. C'est cette autonomie de déplacement qui différencie les agents mobiles de la migration de processus [01].

Un agent réalise généralement une fonctionnalité précise et simple. Qu'elle soit cliente ou serveur, une application réalisant des tâches plus complexes sera constituée de plusieurs agents comme dans le cas de la conception orientée objet. Nous pouvons donc abstraire les applications réparties comme un ensemble d'agents se déplaçant de site en site et dialoguant entre eux. Le déplacement s'effectue pour deux raisons principales : Trouver et utiliser les services applicatifs désirés, ou simplement pour mieux s'adapter à leur environnement, pour équilibrer la charge du réseau par exemple (migrer lorsqu'un site est trop chargé). Le dialogue entre agents sert à mettre en commun des fonctionnalités ou simplement à connaître l'avancement de la tâche générale, par exemple, pour en connaître la terminaison. Cette phase de dialogue est nommée *coopération*.

1.3.3.1. Avantages

Le paradigme des agents mobiles a suscité un intérêt particulier dans le domaine de la recherche portant sur les applications réparties à large échelle. Ce schéma d'organisation peut apporter des améliorations indéniables portant notamment sur le gain de performance qui est dû à une meilleure utilisation des ressources disponibles [01]. Cette amélioration intervient à différents niveaux :

- *Diminution de l'utilisation du réseau :* Le déplacement des agents mobiles réduit significativement la communication distante entre clients et serveurs. En privilégiant les interactions locales, l'utilisation du réseau se limitera principalement au transfert des agents. Cette situation présente trois principaux avantages :
  - Elle permet de réduire le temps de latence notamment dans le cas des applications déployées sur des réseaux à large échelle.
  - Elle permet de réduire la consommation de la bande passante, ce qui est particulièrement utile dans les environnements sans fil. Cette diminution est constatée dans les différents types d'applications qui nécessitent d'intenses échanges entre clients et serveurs. On cite à titre d'exemple, la collecte d'informations dans des bases de données réparties. Elle permet également de réduire la consommation d'énergie et de prolonger ainsi la durée de vie des batteries des dispositifs mobiles.
  - Elle permet de réduire les périodes de connexion entre les sites ce qui permet de moins se soucier des opérations de déconnexion fréquentes dues à la connectivité sans fil.
- *Optimisation du traitement :* Le déplacement du savoir faire va permettre de déléguer les calculs à des machines serveurs généralement plus puissantes que celles des clients.
- *Tolérance aux fautes :* En se déplaçant avec leur code et leurs données propres, les agents mobiles peuvent s'adapter facilement aux erreurs systèmes. Ces erreurs peuvent être purement physiques, la défaillance d'un nœud par exemple, ou d'ordre fonctionnel telles que l'arrêt d'un service.
- *Calculs indépendants (l'autonomie) :* Dans le modèle classique d'évaluation à distance, le client et le serveur doivent rester connectés tant que le service est en cours d'exécution. Avec les agents mobiles, un utilisateur nomade peut demander un service, se déconnecter puis récupérer ses résultats ultérieurement lors d'une prochaine session.

1.3.3.2. Limites

Du point de vue conception, les agents mobiles représentent une méthode permettant de mieux caractériser certaines classes d'applications, mais ils apportent aussi une complexité accrue par rapport au paradigme classique Client / Serveur bien maîtrisé. Sur le plan développement, la complexité de mise au point des programmes est un problème fondamental. En effet, la majorité des environnements de programmation disposent d'outils de débogage permettant de suivre l'évolution des différents éléments d'une application et de suivre son exécution afin de détecter les erreurs. Toutefois, ce genre d'outils devient très difficilement applicable lorsque les éléments de l'application perdent leur aspect statique.

Un autre problème en relation avec le développement est la difficulté de mettre en place une technique de vérification des applications à base d'agents mobiles. Parmi les techniques les plus utilisées bien qu'elle ne soit pas totalement satisfaisante est le *test*. Dans cette méthode, une application est mise à l'épreuve en lui injectant des données de ses différents points d'entrée dans le but de simuler le comportement utilisateur et de couvrir une plage de situations la plus large possible. Si le test s'applique assez naturellement dans les programmes classiques, c'est à dire non répartis, il représente un domaine de recherche à part entière dès qu'il s'agit des applications réparties. Dans ce cadre, il est nécessaire de coordonner les différentes injections sur les points d'entrée répartis pour pouvoir garantir que le comportement simulé est bien celui qui était visé. Cette méthode est déjà loin d'être simple et satisfaisante dans le cas des applications réparties statiques, on peut donc facilement imaginer que cela devient encore plus compliqué si les points d'entrées (agents) deviennent mobiles [01].

La vérification formelle est une autre méthode de vérification qui a pour vocation d'assurer le bon fonctionnement et la fiabilité des applications réparties avant leur implémentation et déploiement, et qui permet de se passer de la phase des tests. Une première étape dans cette méthode de vérification consiste à décrire le système à étudier dans un formalisme spécifique. Il s'agira ensuite d'examiner les propriétés comportementales du modèle établi afin de détecter les erreurs éventuelles. Les travaux dans ce sens constituent un axe de recherche encore largement ouvert.

Notre travail s'insère dans ce cadre spécifique et se focalise sur les modèles formels dédiés à la description du comportement des applications réparties conçues sur des styles

architecturaux fondés sur la mobilité logique, les agents mobiles plus précisément. Dans le reste du mémoire, nous référençons ce type d'applications par l'expression *Systèmes mobiles*.

1.4. Conclusion

La mobilité amplifie davantage le degré de complexité des systèmes. En conséquence, la compréhension et l'analyse de leur structure et de leur comportement d'une part, l'expression et la formalisation de leurs propriétés d'autre part, sont devenues des tâches qui constituent de vrais défis. Pour cela, des formalismes doivent être établis à différents niveaux d'abstraction, chacun d'eux traitant des aspects bien particuliers des systèmes mobiles. Les modèles formels dédiés à la description des systèmes mobiles font l'objet du prochain chapitre.

Chapitre 2

Modèles formels pour les systèmes mobiles

2.1. Introduction

Plusieurs formalismes ont été proposés pour modéliser les systèmes mobiles et raisonner sur les différents aspects liés à la mobilité. Ces formalismes peuvent être classifiés en deux grandes catégories : Approches basées sur les algèbres de processus et approches basées sur les réseaux de Petri. Dans notre étude, nous nous intéressons plus particulièrement à cette dernière catégorie. Ce choix est incité par la représentation graphique attrayante des modèles de réseaux de Petri, leur base théorique solide et l'importance de leur communauté.

Dans ce chapitre, nous commençons d'abord par l'identification d'un certain nombre d'exigences fondamentales que doit posséder un formalisme pour la description des systèmes mobiles. Ensuite, nous citons quelques modèles d'algèbres de processus puis nous présentons une étude des formalismes basés sur les réseaux de Petri et des techniques qu'ils adoptent. A l'issue de cette étude, nous choisirons l'un des modèles étudiés pour servir de base à notre contribution qui fera l'objet du chapitre 4.

2.2. Caractéristiques fondamentales d'un modèle pour les systèmes mobiles

L'introduction de la mobilité aux applications réparties a augmenté davantage leur complexité de manière directe (mobilité des processus, création et suppression dynamique des composants, nouveaux modes de communication, …) et indirecte par le besoin devenu stricte et inévitable de la prise en compte de problèmes préexistants (sécurité, contrôle d'accès, …). La définition de modèles formels pouvant supporter les nouvelles caractéristiques exhibées par ces systèmes doit s'articuler sur un ensemble d'abstractions essentielles et d'exigences techniques [02, 09] :

- **Abstractions à base de composants.** Un système mobile doit être vu comme un ensemble de constituants autonomes disposant chacun d'une activité propre. Au niveau de la modélisation, il convient donc de représenter ces unités d'exécution sous forme d'abstractions introduisant la notion d'activité, comme les processus ou les threads. Des *primitives de communication* doivent être prévues pour mettre en œuvre les interactions entre ces composants. Ces derniers peuvent être mobiles, par conséquent, des *primitives de migration* doivent être également prévues pour supporter cette mobilité. En outre, ces composants doivent pouvoir interagir avec leur environnement au moyen d'interfaces bien définies. Et enfin, ils doivent disposer d'une identité les désignant sans ambiguïté et en permettant l'accès et la manipulation.
- **Connectivité dynamique.** La connectivité permet à un composant d'interagir avec un autre par la disposition d'une connexion ou référence. Dans les systèmes mobiles, les composants logiciels peuvent être créés, supprimés ou liés dynamiquement à d'autres composants suite à des communications durant l'évolution du système. Ils peuvent également disparaître et réapparaître suite à leur mobilité (agent mobile migrant dans un domaine administratif n'autorisant pas les communications vers l'extérieur) ou à la mobilité des dispositifs matériels qui les transportent (téléphone portale temporairement déconnecté du réseau car situé dans une zone non couverte). Il est donc indispensable que les modèles dédiés à la description des systèmes mobiles soient en mesure de supporter une *connectivité dynamique* qui évolue au cours de l'exécution des systèmes.
- **Localité.** La notion de localité peut être abordée de différentes manières. Il s'agit d'un concept très délicat à formaliser et intimement lié au domaine d'application considéré. Cette notion se manifeste par la prise de conscience par les composants logiciels de la localisation qu'ils occupent *« network-awarness »*. Il peut s'agir d'une *localisation physique*, par exemple, l'accès à une base de données répliquée se fera en fonction de l'emplacement physique de l'utilisateur qui choisira le serveur miroir le plus proche pour bénéficier du meilleur temps d'accès. Comme il peut s'agir d'une *localisation virtuelle* auquel cas la notion de localité capture le partitionnement du réseau en différentes régions ou *domaines administratifs* pouvant éventuellement se recouvrir. Dans ce cas, chaque localisation, possède des propriétés distinctives et traite de manière indépendante les différents aspects de sécurité, de communication, de tolérance aux défaillances, …etc. Ces localisations sont séparées par des barrières pour les protéger contre les attaques. Les entités calculatoires mobiles doivent donc être

dotées de moyens leurs permettant de traverser ces barrières administratives lors de leurs déplacement d'une localisation à une autre.

- **La sécurité.** l'aspect sécurité au sens large du mot constitue un autre ingrédient qui doit être au centre d'intérêts des formalismes pour pouvoir construire des applications dignes de confiance. En effet, la mobilité amplifie davantage les enjeux qui se présentent en termes de sécurité. Par exemple, les agents mobiles acquièrent les ressources qui leurs sont nécessaires des sites qu'ils visitent ; il est donc nécessaire de prévoir des mécanismes permettant de supporter les méthodes assurant le contrôle d'accès aux ressources et empêchant l'utilisation abusive de celles-ci. Aussi, des primitives doivent être prévues pour protéger les localisations contre les attaques et contrôler la migration des composants entre celles-ci.

Ces caractéristiques regroupent les concepts de base qui doivent être pris en compte par les modèles de description des systèmes mobiles. D'autres caractéristiques et primitives doivent être considérées par les modèles servant de noyaux aux langages concurrents destinés à la programmation répartie et mobile, nous citons à titre d'exemple la coordination [10] et la gestion des défaillances pour permettre la construction de systèmes robustes.

Par ailleurs, les modèles doivent satisfaire les deux exigences générales suivantes :

- **La simplicité.** Le modèle doit rester simple pour que les techniques et outils associés restent tractables et puissent être utilisés pour raisonner sur les systèmes.
- **La complétude.** Cette exigence signifie que le modèle doit être suffisamment expressif pour que ses primitives permettent de décrire de manière assez complète les caractéristiques essentielles du système modélisé.

Notons que ces caractéristiques constituent deux notions rivales. Si un modèle est trop expressif alors, on ne peut pas mathématiquement l'analyser de façon automatique. Le défi est de trouver un bon compromis entre simplicité et complétude et donc, de proposer des modèles qui soient suffisamment expressifs pour décrire la réalité des systèmes tout en restant raisonnablement analysables.

2.3. Modèles à base d'algèbres de processus

Les algèbres de processus ou calculs de processus constituent une catégorie importante des modèles permettant la description des interactions réalisables au sein des systèmes concurrents [31].

Il s'agit de modèles mathématiques définis par une syntaxe et une sémantique : La syntaxe est à l'origine très simple et comprend un nombre réduit d'opérateurs algébriques primitifs (composition séquentielle, parallèle, choix non déterministe, …) qui par assemblage permettent de décrire des comportements complexes. La sémantique est définie formellement de manière axiomatique et opérationnelle. Une sémantique axiomatique consiste en un ensemble de lois algébriques (commutativité, associativité, distributivité, …) permettant de démontrer l'équivalence des termes. La sémantique opérationnelle consiste en une relation de transition $\boldsymbol{P \xrightarrow{a} P'}$ exprimant le fait que le terme *P* peut effectuer l'action *a*, puis évoluer et se transformer en un terme *P'*.

L'un des premiers modèles, CCS (Calculus of Communicating Systems) [11,12], est né des travaux de Milner sur les systèmes concurrents. Dans ce modèle, chaque processus possède un ensemble de canaux de communication, référencés par des noms, sur lesquels il peut échanger des données avec les autres processus. Toutefois, le nombre de canaux que possède un processus est *fixe*, c'est-à-dire que sa connaissance sur son environnement n'évolue pas durant l'exécution du système. Cet aspect statique de la topologie de communication constitue la limitation majeure du modèle. La connectivité dynamique étant l'une des caractéristiques cruciales des systèmes mobiles, il s'est avéré nécessaire de faire évoluer ce formalisme en vue de pouvoir la capturer. Ce constat a incité Milner à définir un nouveau calcul de processus basé sur CCS mais dont le principe consiste à permettre l'échange de noms de canaux entre processus, assurant ainsi la représentation d'une topologie de communication dynamique. Il s'agit du π-calcul [13] qui apparaît donc comme le premier formalisme qui permet de gérer la mobilité. Ce dernier a servi de base pour une foison de calculs qui ont été proposés par la suite, chacun d'eux dédié au traitement d'un aspect bien particulier des systèmes mobiles. Ces formalismes, y compris le π-calcul, sont réunis sous l'appellation de c*alculs nominaux* [15][1]. Leur principe de base repose sur une technique d'abstraction simple [16] qui consiste à référencer des objets de natures diverses présents dans un système

[1] Un calcul nominal est un formalisme qui dispose d'un ensemble abstrait de noms purs et d'un opérateur pour la génération locale de noms frais. Ces calculs possèdent des caractéristiques essentielles pour la description de la mobilité et des protocoles de sécurité.

distribué par des *noms purs*. Ces noms abstraits sont simplement des entités atomiques sur lesquelles la seule opération possible est le test d'égalité. Malgré sa simplicité, cette technique assure aux calculs nominaux un support puissant pour le traitement de la mobilité et l'aspect sécurité en particulier [15]. Dans cette section, nous passons en survol les calculs nominaux les plus pertinents de la littérature qui s'inscrivent dans le cadre de la mobilité.

2.3.1. π-calcul

Le π-calcul est un modèle mathématique qui offre un moyen pour décrire et analyser des systèmes qui consistent en des processus dont la configuration de la topologie de communication change de manière continue. Le π-calcul permet de créer dynamiquement des noms de canaux et de limiter leur connaissance à certains processus grâce à un opérateur spécial dit *de restriction*. Ensuite ces noms de canaux peuvent être transférés entre les processus à travers d'autres canaux, ce qui permet l'établissement de nouvelles connexions. L'ensemble des noms des canaux dont un processus a la connaissance forme le contexte perçu par celui-ci. Le π-calcul a fait l'objet d'un grand nombre d'études. Il constitue pour cela un calcul référence pour plusieurs formalismes dédiés à la mobilité.

Il existe plusieurs versions du π-calcul. Le Monadic π-calcul est la forme basique, qui permet l'émission d'un seul nom de canal à la fois dans un message. Dans ce formalisme, les noms des canaux sont les seules valeurs qui peuvent être manipulées, ce qui permet de focaliser sur l'aspect communication dans la modélisation.

Le Polyadic π-calcul [17] étend le monadic π-calcul pour permettre l'émission d'un tuple de noms de canaux en un seul message. Le High Order π-calcul (HO π-calcul) [18] permet cependant le transfert des processus sur des canaux au même titre que les noms ce qui offre la possibilité de modéliser la mobilité directement. Une fois le processus transmis, celui-ci peut commencer son exécution.

Le Nomadic π-calcul [19] est une extension du π-calcul avec une notion explicite de sites, d'agents et de mobilité. Un système, dans ce calcul, consiste en une collection de sites (localisations) nommés et d'agents qui migrent entre ces sites grâce à des primitives de migration. Cette extension opère sur deux niveaux d'abstraction selon que la communication

soit dépendante de la localisation ou pas : Un niveau *haut* où les primitives sont indépendantes des sites et un niveau *bas* où les primitives dépendent des sites.

Le π-calcul existe également en deux versions : Synchrone et asynchrone. L'asynchronisme est simplement exprimé par le fait que l'envoi d'une donnée (aussi appelée message) sur un canal n'est pas bloquant pour le processus émetteur, ce qui intuitivement, implique qu'on ne peut pas contrôler le moment où un message sera effectivement consommé.

2.3.2. Le join-calcul

Le join-calcul [22] est un calcul de processus fondé sur la communication asynchrone des canaux. Il est considéré comme une progéniture du π-calcul asynchrone. Il modélise la mobilité par l'échange des noms de canaux de la même manière que le π-calcul et a la même expressivité que celui-ci. Le join-calcul distribué [23] est une extension du join-calcul avec une notion de localisation et de migration explicite. Il propose en plus des primitives simples pour la détection et la gestion des erreurs au niveau des localisations. La sémantique du join-calcul et du join-calcul distribué est donnée en termes de machines chimiques abstraites (CHAM) [21] spécifiques.[2]

2.3.3. Le calcul des ambients mobiles

Le calcul des ambients mobiles [24] est un calcul nominal dont l'abstraction de base qui est *l'ambient*, représente des environnements calculatoires mobiles, éventuellement imbriqués et permettant des communications locales. Ce calcul vise à fournir un cadre général pour la description uniformisée de la mobilité dans ses deux formes (physique et logique) tout en focalisant sur l'aspect sécurité.

Le problème majeur traité dans ce formalisme est l'aspect gestion des domaines administratifs. Un ambient doit non seulement avoir une autorisation pour s'exécuter ou accéder à une ressource au niveau d'un domaine, mais aussi pour sortir d'un domaine ou pour y accéder, traverser un par feux par exemple.

[2] Une machine chimique abstraite CHAM est un modèle général qui illustre comment les calculs concurrents peuvent être exécutés. Les mécanismes de la CHAM ressemblent à ceux rencontrés dans les solutions chimiques, c'est la raison pour laquelle ils sont décris par des termes tirés de la chimie.

Un ambient peut être défini comme étant un environnement calculatoire délimité par une frontière et contenant des données, du code et des processus. Un ambient peut contenir d'autres ambients, ce qui conduit à une structure hiérarchique. Les ambients peuvent se déplacer dans cette hiérarchie sous le contrôle des processus qu'ils contiennent, comme ils peuvent être dissous pour révéler leur contenu.

Chaque ambient doit avoir un nom. Connaitre le nom d'un ambient par un processus lui donne la capacité d'accéder à cet ambient, de le quitter ou de le dissoudre. Autrement dit, la connaissance des noms des ambients est utilisée comme technique pour le contrôle l'accès. Pour cela, le nom d'un ambient est une information privée qui ne doit pas être distribuée à tous les processus.

Pour qu'un ambient puisse être déplacé par l'un de ses processus, le processus doit avoir la capacité de quitter l'ambient qui l'encapsule ou la capacité d'accéder à un ambient frère (contenu dans le même ambient). Ces capacités correspondent à des autorisations de quitter où d'accéder à des domaines administratifs.

Dans certaines situations, un ambient ne peut pas se déplacer directement (en un seul pas de calcul) vers l'ambient ou la localisation destination ; il doit plutôt passer par un certain nombre d'étapes. Ceci est dû au fait qu'un ambient ne peut changer de position qu'à l'intérieur de son hiérarchie avec un seul niveau hiérarchique par déplacement. Considérons le cas de la figure 2.1. Un composant au niveau de la localisation *l1* souhaite se déplacer vers la localisation *l2* située dans un autre domaine administratif. Cela peut être réalisé d'abord en se déplaçant en dehors de la localisation *l1*, puis en dehors de son domaine administratif, puis accéder au domaine administratif de *l2*, en suite à la localisation *l2*. Il en résulte que la simple connaissance du nom de l'ambient destination dans certains cas n'est pas suffisante pour y accéder.

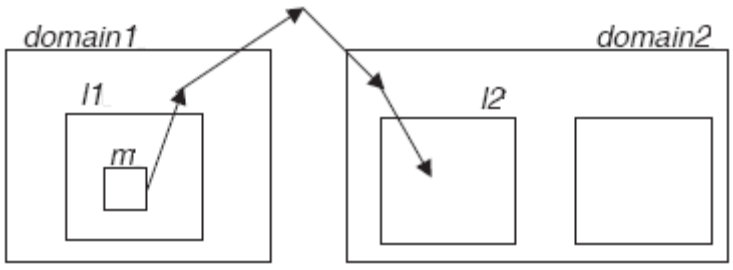

Fig.2.1- Migration d'un ambient

Le calcul des ambients restreint la communication entre processus résidant dans le même ambient. La communication se fait par passage de messages asynchrones. Cependant, le transport des messages entre ambients peut être modélisé en enveloppant le message dans un ambient qui doit avoir les capacités nécessaires pour voyager entre les autres ambients. Ainsi, les communications distantes sont modélisées par une combinaison de mobilité et de communications locales.

2.3.4. Klaim

Klaim (Kernel Language for Agent Interaction and Mobility) est un cœur d'un langage de programmation visant à coordonner un ensemble de processus partageant des ressources dans un contexte réparti supportant la mobilité du code. C'est un noyau du langage de coordination LINDA[3] [05, 06] à espaces de tuples multiples, enrichi par des opérateurs de l'algèbre de processus CCS de Milner.

Plusieurs extensions de Klaim existent. Plus particulièrement, cKlaim (core Klaim) [25] constitue une algèbre de processus qui résume les caractéristiques fondamentales de Klaim et qui peut être considérée comme une variante de π-calcul avec communication asynchrone de noms à travers des mémoires de stockage partagées et localisées. μKlaim [26] est une extension de cKlaim qui permet de contrôler la mobilité et l'accès aux ressources par l'attribution dynamique de privilèges aux processus. StocKlaim (Stochastique Klaim) [27] est une autre extension qui intègre la modélisation des aspects quantitatifs dans les systèmes mobiles.

Ce qui distingue Klaim des autres approches est l'infrastructure réseau qui est explicitement modélisée et nettement distinguée des processus utilisateur ; cela permet une séparation claire entre le niveau calculatoire et le niveau coordination ce qui garantit un meilleur traitement des applications mobiles.

[3] LINDA est un langage de coordination basé sur un mécanisme de communication mettant en œuvre une mémoire globale partagée appelée espace de tuples. Un langage de coordination fournit le support nécessaire aux activités calculatoires pour communiquer et interagir afin de réaliser un objectif.

2.3.5. Bilan sur les modèles à base d'algèbres de processus

On observe dans les modèles, dits calculs nominaux, décris précédemment qu'il existe une grande diversité de techniques et de primitives permettant de supporter les différentes caractéristiques exhibées par les systèmes mobiles. La notion de connectivité dynamique est généralement capturée grâce à la technique des noms abstraits. La notion de localité est rendue transparente dans certains formalismes tels que le Monadic π-calcul et le join-calcul, et elle est manipulée explicitement par d'autres tels que le Nomadic π-calcul et Klaim. Plusieurs extensions de calculs ont été établies afin de gérer les différents aspects de sécurité tels que l'attribution dynamique des privilèges aux processus, la restriction de la migration entre les localités et le contrôle d'accès aux ressources.

La vérification de systèmes d'une algèbre de processus consiste généralement à écrire deux spécifications, d'un coté une spécification *Sys* décrivant le comportement effectif du système, de l'autre une spécification *Spec* décrivant une abstraction donnée de celui-ci, puis à établir la correction de *Sys* par rapport à *Spec* en montrant que *Sys* se comporte de la même manière que *Spec*. Pour montrer cette correction, il existe deux procédés différents : le procédé orienté syntaxe et celui orienté sémantique. Dans le premier cas, on cherche à établir des relations d'équivalence entre les termes en se basant sur leur structure statique. Dans le second cas par contre, on définit des relations d'équivalence entre les expressions en fonction de l'ensemble des réductions possibles, c'est-à-dire par rapport aux comportements [14].

Dans certains cas, par exemple lorsque les spécifications sont finies, la vérification basée sur la syntaxe ou sur la sémantique peut être réalisée automatiquement. Dans les calculs nominaux, cependant, bien que la technique de génération dynamique des noms frais les pourvoit du support nécessaire pour la gestion de la mobilité et de la sécurité des systèmes, celle-ci rend la vérification automatique une tâche fastidieuse et défiante, voire impossible. Les recherches dans ce cadre constituent encore des études prospectives largement ouvertes.

Les algèbres de processus, grâce à leur puissance d'expression et à la diversité éminente des primitives qu'ils mettent en œuvre, sont des modèles bien adaptés pour servir de noyaux aux langages concurrents notamment ceux destinés à la programmation mobile et répartie à large échelle [02].

Néanmoins, ces formalismes constituent des modèles purement textuels qui ne facilitent pas le raisonnement direct sur la concurrence et la causalité exhibées par les systèmes modélisés, ce qui favorise l'adoption d'un modèle graphique tel que les réseaux de Petri pour une meilleure compréhension et maîtrise du fonctionnement des systèmes modélisés.

2.4. Modèles à base de réseaux de Petri

Les réseaux de Petri sont un formalisme graphique largement répandu pour la modélisation du comportement des systèmes concurrents. Depuis leur introduction par C.A. Petri en 1962 dans sa thèse intitulée *Communication avec des Automates* [46], les réseaux de Petri ont fait l'objet d'un très grand nombre de travaux qui ont révélés une base théorique solide et qui ont donné naissance à une variété éminente de techniques et d'outils pour l'analyse de plusieurs propriétés et problèmes associés aux systèmes concurrents, nous citons à titre d'exemple l'accessibilité, la vivacité, la couverture, … (voir annexe A).

Avec l'introduction de la mobilité, les systèmes modernes ont connu une dynamique intense, le nombre de composants impliqués ainsi que les liens entre ceux-ci fluctuent continuellement durant le calcul de façon imprédictible. Il s'avère que les réseaux de Petri souffrent de leur nature statique, et sont peu adaptés pour être utilisés directement pour la modélisation et la spécification de ce genre de systèmes. Pour pouvoir profiter de tous les avantages qu'apporte l'utilisation des réseaux de Petri (représentation graphique attrayante, base théorique solide, outils d'analyse et de simulation, possibilité de raisonner directement sur la concurrence et la causalité exhibées par les systèmes), plusieurs modèles ont été proposés, certain reposent sur le paradigme *nets within nets* et d'autres s'inspirent fortement des modèles d'algèbre de processus et emploient la technique de gestion des noms pour combler les limite des réseaux de Petri. Nous présentons dans la section courante certains de ces modèles.

2.4.1. Nets within nets pour la mobilité

Le paradigme *nets within nets* (voir annexe B) formalise l'idée que les jetons d'un réseau de Petri peuvent eux même être des réseaux de Petri. Dans ce paradigme, on modélise les objets d'un système par des réseaux qu'on appelle *réseaux objets (object nets).* A partir de ces réseaux, sont instanciés les *jetons réseaux* qui font partie du marquage du réseau qui modélise le système sous-jacent, appelé *réseau système (system net)*. Ces modèles sont dits *systèmes à objets (Object systems).* Plusieurs niveaux hiérarchiques peuvent être envisagés. Dans le cas où l'on se limite à un seul niveau d'imbrication, on parle de *systèmes à objets élémentaires EOS*. Dans les *EOS,* les jetons réseaux se déplacent dans le réseau système comme des jetons ordinaires tout en ayant la possibilité de changer leur marquage local mais pas leur structure.

En adoptant ce point de vue, il est possible de capturer de manière intuitive et naturelle la notion de mobilité : Une entité mobile est décrite par un *jeton réseau* qui peut se déplacer dans le réseau système qui modélise le système environnant. Pour cela, plusieurs modèles basés sur ce paradigme ont été proposés pour traiter la mobilité. Deux de ces modèles seront présentés dans les deux sections suivantes : Les *mobile EOS* et les *Petri hypernets*.

2.4.1.1. Mobile EOS

Dans le modèle des systèmes à objets élémentaires EOS, deux sémantiques ont été définies : Une sémantique par référence et une sémantique par valeur (voir annexe B). Dans la sémantique par référence, toutes les places du réseau système appartiennent à un *espace de noms global unique*. Cette sémantique fournit aux objets une connectivité maximale au détriment de la notion de localité, ce qui contredit l'intuition lorsqu'il s'agit de la modélisation des systèmes distribués. En effet, dans le cas d'un espace distribué, un objet peut être directement accessible via un pointeur à l'intérieur de sa localisation courante ce qui est bien modélisé par la sémantique par référence. Par contre, lorsque cet objet doit être transféré à une autre localisation du réseau, celui-ci doit être traité comme étant une valeur qui peut être copiée sur des messages pour être transmise sur le réseau, ce qui ne peut pas être exprimé par la sémantique par référence.

La sémantique par valeur met en œuvre *des espaces de noms locaux*, chacun correspond à une place spécifique du réseau qui modélise le système. Cette sémantique s'avère mieux adaptée à la modélisation des systèmes distribués du moment qu'elle permet de supporter la notion de localité, mais encore trop rigide. Un réseau objet qui réside dans une place du réseau système ne peut se synchroniser (interagir) qu'avec les transitions adjacentes à cette place, ce qui réduit significativement la connectivité des objets.

Le modèle des Mobile EOS a été présenté dans [38]. Il résulte de l'enrichissement du modèle des systèmes à objets élémentaires EOS afin de couvrir les exigences de modélisation des systèmes mobiles et distribués. Des modifications ont été apportées sur deux niveaux : Sur le plan structure par la définition d'*une infrastructure de localité*, et sur le plan sémantique par la définition d'une nouvelle sémantique dite *de mobilité*. L'infrastructure de localité consiste à définir sur le réseau système un ensemble de localisations disjointes et un ensemble de transitions dites de *mouvement* permettant aux agents de se déplacer entre ces localisations.

Dans la nouvelle sémantique de mobilité, on suppose que chaque localisation constitue un espace de noms. Alors, à l'intérieur d'une même localisation, un objet est directement accessible via un pointeur (sémantique par référence), mais pour être transmis sur le réseau vers une autre localisation, il doit être traité comme étant une valeur (sémantique par valeur).

Ainsi, la sémantique de mobilité permet de définir un espace de noms plus flexible, c'est-à-dire, plus large que les places (sémantique par valeur) mais qui n'implique pas nécessairement toutes les places du réseau système (sémantique par référence).

2.4.1.1.1. L'infrastructure de localité

L'infrastructure de localité permet de définir à partir d'un système à objets élémentaires EOS, un *système de mobilité* avec une notion d'espace de noms flexible. Etant donné un réseau système $\widehat{N}$ du EOS, celui-ci est décomposé en un ensemble de localisations disjointes, chacune constituant un espace de noms, connectées par des éléments de mobilité dits transitions de mouvement, permettant aux objets mobiles de se déplacer entre les localisations. L'ensemble des transitions de mouvement constitue une *infrastructure de mobilité.*

Exemple 2.1 [38]

Le EOS de la figure 2.2 est constitué du réseau système $\widehat{N}$ tel que $\hat{T} = \{t_1, \dots, t_8\}$ et $\hat{P} = \{p_1 \dots, p_{10}\}$ et d'un seul objet réseau $\mathcal{N} = \{N\}$ avec $T_N = \{t_{20}, t_{21}, t_{23}, t_{24}\}$ et $P_N = \{p_{20}, p_{21}, p_{22}, p_{23}, p_{24}\}$. Les transitions annotées par la même étiquette doivent obligatoirement être franchies simultanément. Elles forment une structure de synchronisation (voir annexe B).

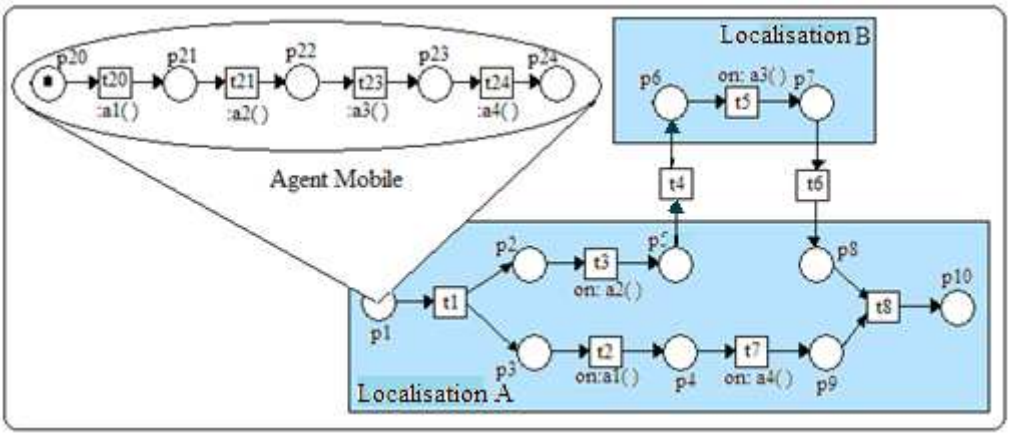

Fig.2.2- Système de mobilité.

L'infrastructure de localité définit sur le réseau système $\widehat{N}$ un système de mobilité constitué de :

- Une infrastructure de mobilité constituée des deux transitions t_4 et t_6 ;
- Un ensemble de deux localisations A et B avec :
  - $T_A = \{t_1, t_2, t_3, t_7, t_8\}$;
  - $P_A = \{p_1, p_2, p_3, p_4, p_5, p_8, p_9, p_{10}\}$;
  - $T_B = \{t_5\}$;
  - $P_B = \{p_6, p_7\}$.

2.4.1.1.2. La sémantique de mobilité

Dans un système de mobilité, chaque localisation doit correspondre à un espace de noms distinct. Cela revient à dire qu'à l'intérieur d'une même localisation, les objets sont manipulés suivant la sémantique par référence tandis qu'entre les différentes localisations, c'est la sémantique par valeur qui est mise en œuvre. Cette sémantique mixte est dite sémantique de mobilité.

2.4.1.1.3. Bilan

Le formalisme mobile EOS est une extension des réseaux de Petri à objets basés sur le paradigme *nets within nets*. Ce formalisme repose sur la définition d'une sémantique généralisée appropriée aux concepts de distribution et de mobilité. Cette nouvelle sémantique est mise en œuvre par la définition d'un système de mobilité. Ce dernier introduit le concept de localités par la définition d'un ensemble de localisations. Il introduit également le concept de mobilité en prévoyant une infrastructure permettant le mouvement entre ces localisations.

Du fait qu'il soit basé sur une approche de réseaux de Petri orientée objet, le formalisme mobile EOS arrive à fournir une abstraction des systèmes à base de composants structurée en deux niveaux : un niveau représentant les composants actifs qui constituent le système et un autre niveau pour modéliser l'environnement dans lequel évoluent les composants. Ces derniers parviennent à interagir avec leur localisation courante à travers des transitions synchronisées, mais il n'existe pas de primitives leur permettant d'interagir entre eux. Le modèle capture la notion de localité de manière explicite mais n'est pas assez flexible pour capturer le concept de connectivité dynamique ce qui réduit significativement son expressivité.

Il a été démontré que le formalise Mobile EOS simule les réseaux de Petri à arcs inhibiteurs [38]. Par conséquent, toutes les propriétés intéressantes sont indécidables (accessibilité, couverture, bornitude, vivacité, …).

2.4.1.2. Petri hyper nets

Petri hyper nets [39] est une autre branche du paradigme nets within nets qui opère sur plusieurs niveaux d'imbrication. Dans ce formalisme, les agents mobiles sont modélisés par des réseaux de Petri et peuvent être manipulés en tant que jetons par d'autres agents mobiles de niveau hiérarchique supérieur.

Les Petri hyper nets fournissent une description *hiérarchique* et *modulaire* des systèmes à agents mobiles. Hiérarchique, car les agents mobiles sont typés et modélisés par des composants appelés *réseaux ouverts*, chaque agent doit être d'un niveau hiérarchique supérieur à celui des agents qu'il manipule comme jetons. La modularité provient du fait que les agents sont constitués de la composition synchrone d'un ensemble de *modules* typés également. Chacun de ces modules a la structure d'une machine à états[4] et est destiné à gérer les agents de son type. Un Petri hyper net est une collection de réseaux ouverts structurés de manière hiérarchique, avec un *hyper marquage* qui constitue une assignation des jetons réseaux aux places.

Les modules d'un même agent peuvent communiquer localement par synchronisation de transitions. De plus, ils sont tolérés à communiquer avec un module du parent immédiat dans

[4] Une machine à états est un réseau de Petri tel que chaque transition a exactement une pré-condition et une post-condition. Par conséquent, les machines à états sont purement séquentielles, c'est à dire que deux transitions ne peuvent jamais être tirées de manière concurrente à partir d'un marquage accessible.

la hiérarchie ou avec un module d'un fils immédiat, c'est à dire un jeton réseau. Cette communication verticale se fait par synchronisation de transitions également et elle est assimilée à un échange de messages. Ces messages sont eux même des agents mobiles migrants, ce qui conduit à un changement dans la hiérarchie.

L'utilisation des machines à états comme modules dans le formalisme résout le problème du choix de la sémantique à employer lors du tir des transitions. Du moment qu'il est impossible d'avoir plus d'une référence à un réseau objet, on pense intuitivement à la sémantique par valeur.

Par ailleurs, le choix des machines à états empêche la création ou la destruction des agents. Ainsi, les hyper nets permettent d'aboutir à des espaces d'états finis malgré la mobilité éventuelle des agents.

Dans le reste de cette section, nous nous limitons à illustrer concrètement la structure du modèle et à décrire informellement son fonctionnement à travers l'exemple suivant.

Exemple 2.2 [39]

Cet exemple illustratif constitue une étude de cas d'un système impliquant trois types d'agents : aéroports, avions (agents de type π) et voyageurs (agents de type τ). La classification des agents en types (canaux) est l'une des caractéristiques principales du modèle. L'agent aéroport est constitué de deux modules : un π-module qui gère les avions, c'est-à-dire qui manipule des jetons correspondant aux avions, et un τ-module qui gère les voyageurs.

La figure 2.3 illustre le π-module de l'aéroport. Une fois l'avion atterri (action *land*), les voyageurs descendent l'un après l'autre (action *deplane*), l'avion est ensuite ravitaillé (action *refuel*) et déplacé vers une nouvelle porte (action *to-gate*) pour recevoir de nouveaux voyageurs (action *board*) et décoller (action *take-off*). Le schéma montré dans la figure 2.3 représente un réseau de Petri ordinaire à l'exception de la pré-condition de la transition *land* et de la post condition de la transition *take-off* qui sont représentées par des demi cercles hachés appelés *places virtuelles*. Ces places virtuelles reflètent l'idée que pour réaliser avec succès les deux actions *land* et *take-off*, l'aéroport doit coopérer avec une autorité de contrôle de

niveau supérieur. Elles constituent le moyen de synchronisation des transitions entre des niveaux adjacents dans la hiérarchie.

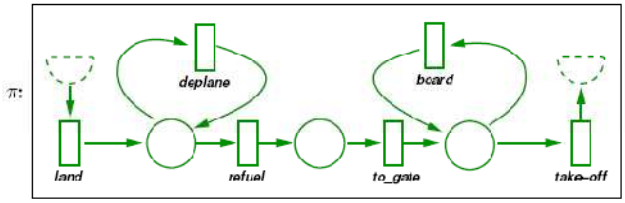

Fig.2.3- Aéroport : module qui gère les avions

La figure 2.4 illustre le τ-module de l'aéroport. Intuitivement, les voyageurs arrivent à l'aéroport suite à l'atterrissage d'un avion (action *land*), puis au déplacement des voyageurs de l'avion vers l'aéroport (action *deplane*). Ceci est formalisé par le demi-cercle marqué par le type π qui représente la pré-condition de la transition *deplane.*

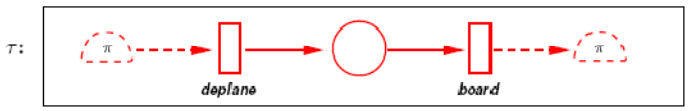

Fig.2.4- Aéroport : module qui gère les voyageurs

En synchronisant les deux modules de l'aéroport, on obtient le réseau ouvert illustré dans la figure 2.5. Les transitions portant le même nom dans les deux modules sont synchronisées ce qui permet les communications locales à l'intérieur d'un agent.

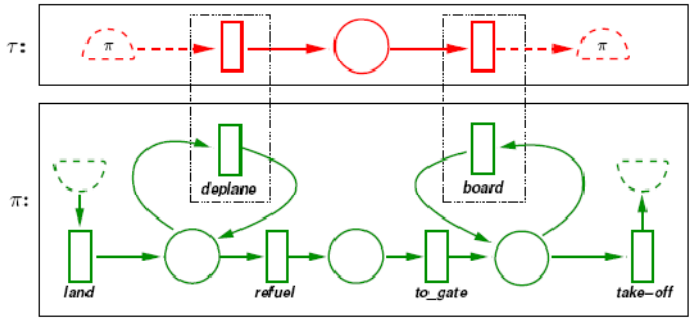

Fig.2.5- Réseau ouvert associé à l'agent aéroport.

Un modèle simple d'un avion est présenté dans la figure 2.6. Il est constitué d'un τ-module permettant de gérer les voyageurs. Les voyageurs montent à bord de l'avion (action *board*) et descendent (action *deplane*) suite à la coopération avec un agent de niveau supérieur

qui est l'aéroport dans notre cas, d'où les deux demi-cercles pré et post-condition des transitions *board* et *deplane*.

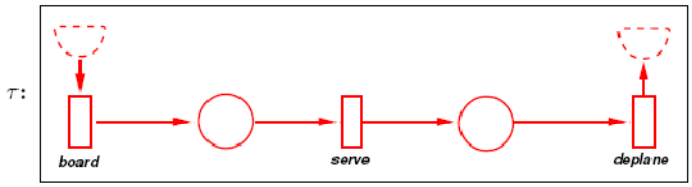

Fig.2.6- Avion : module qui gère les voyageurs

Le hyper net ouvert correspondant au système, illustré dans la figure 2.7 est constitué d'un aéroport, un avion et deux voyageurs (représentés par des jetons ordinaires). Le hyper net est dit ouvert du fait que le niveau le plus élevé (aéroport) est prêt à réaliser des interactions avec un module d'un agent de niveau supérieur.

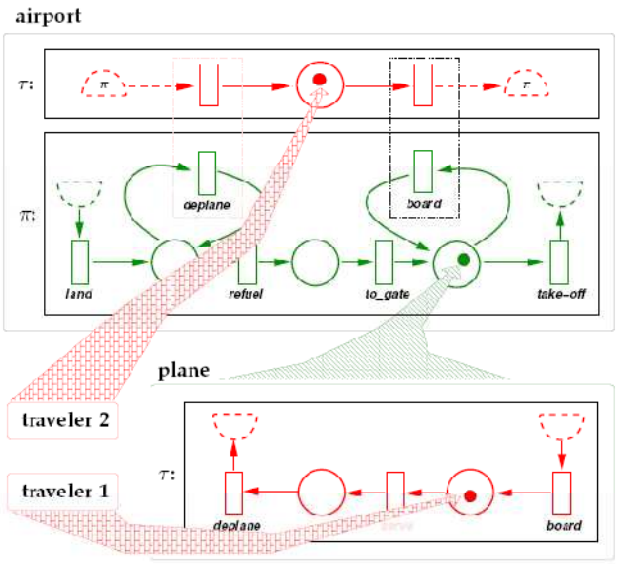

Fig.2.7- Hypernet ouvert

Ainsi, un hyper net est un ensemble de réseaux ouverts $\mathcal{N}$ en plus d'un hyper-marquage $M : \mathcal{N} \rightarrow \cup_{N \in \mathcal{N}} P_N$ qui décrit la distribution des éléments de $\mathcal{N}$ en tant que jetons dans les places appartenant à d'autres réseaux ouverts de $\mathcal{N}$.

Dans l'hyper-marquage présenté dans la figure 2.7, un voyageur peut monter à bord de l'avion (ce qui correspond au tir simultané des transitions *board*). La façon dont cette action complexe est effectuée et le résultat produit sont présentés dans la figure 2.8.

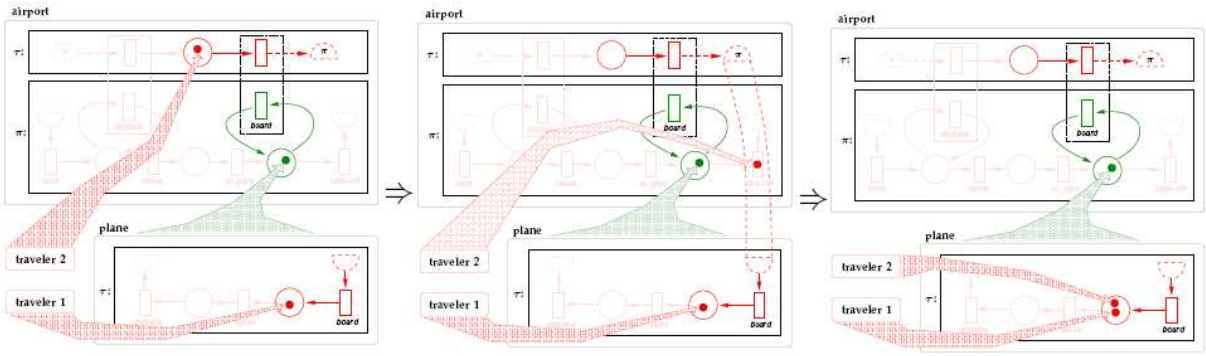

Fig.2.8- Tir de la transition *board*

Le jeton *traveler 2* est envoyé à travers un *canal virtuel* du τ-module associé à l'aéroport vers celui associé à l'avion, ce qui conduit à un changement dans la hiérarchie. Un jeton ne peut pas être observé lorsqu'il traverse le canal virtuel ; l'étape intermédiaire illustrée dans la figure 2.8 ne représente donc qu'une explication intuitive du processus de franchissement.

L'action qu'on décrit implique les transitions *board* à deux niveaux différents : deux instances dans le réseau associé à l'aéroport et une troisième instance dans le réseau associé à l'avion. Elle implique également la manipulation de deux types de jetons : avion et voyageur. Nous déduisons donc que dans un hyper net, le franchissement apparait comme une transition complexe qui implique la synchronisation des activités à différents niveaux.

2.4.1.2.1. Bilan

Petri hyper nets est un autre modèle basé sur le paradigme nets within nets qui a été proposé pour représenter des systèmes à agents mobiles en interaction. Dans ce modèle, on perçoit un système comme une collection d'agents mobiles typés définis suivant une vision modulaire et obéissant à une hiérarchie flexible. La hiérarchie permet d'appliquer des contrôles sur les différents types d'agents mobiles alors que la flexibilité permet leur migration inter niveaux.

Ce formalisme fournit une représentation des systèmes à base de composants actifs structurés en plusieurs niveaux organisés de manière hiérarchique, ce qui augmente la complexité des modèles établis. Les composants peuvent communiquer entre eux et avec leur environnement grâce à des actions complexes impliquant la synchronisation de plusieurs

transitions à la fois. La notion de localité n'est pas traitée explicitement dans le modèle et la connectivité dynamique n'est pas supportée non plus : un sous-système n'a pas la possibilité d'acquérir de nouvelles connexions durant l'exécution.

Les hyper nets fermés, c'est-à-dire qui n'interagissent pas avec un environnement externe, sont caractérisés par leur espace d'états fini grâce à la restriction qui impose que les modules soient des machines à états, ce qui permet de capturer les propriétés essentielles des systèmes.

2.4.2. Modèles basés sur la technique de gestion des noms

Ces modèles s'inspirent des calculs de processus dans leur façon de traiter la mobilité et la dynamique des systèmes. Ils adoptent l'abstraction qui consiste à référencer des entités qui peuvent être de nature diverse (processus, canaux, localisations) par des noms. Ces noms peuvent être générés de manière dynamique comme ils peuvent être transmis suite à des communications à la π-calcul.

2.4.2.1. Mobile Petri Nets

Mobile Petri nets [29] est l'un des premiers formalismes basés sur les réseaux de Petri qui ont été proposés pour la modélisation des systèmes mobiles. Il s'inspire du join-calcul et capture la mobilité en mettant en œuvre le mécanisme de gestion des noms au sens de Milner, à savoir, la possibilité de passer le nom d'un canal à un processus suite à une communication. Du point de vue réseaux de Petri, les canaux ou ports sont assimilés à des places, et les communications correspondent au tir des transitions. Une transition dans mobile net est décrite sous la forme $c \triangleright p$ où c est la pré-condition appelée aussi *motif* et p indique la post-condition.

Le modèle Dynamic nets [29] est une extension de Mobile nets qui capture la notion de réflexivité. Dans ce modèle, l'ensemble des places et des transitions du réseau augmente durant l'exécution du système. Au lieu de produire de nouveaux jetons seulement, les transitions à leur tir peuvent produire de nouveaux sous réseaux. Une transition dans dynamic net est donnée sous la forme $c \triangleright N$, où N spécifie le sous réseau produit par le tir de la transition.

Ces nouvelles caractéristiques introduites aux réseaux de Petri ont permis de briser leur l'aspect statique, elles sont cependant, à l'origine d'un inconvénient, car il devient difficile, voire impossible, de bénéficier d'une représentation graphique des réseaux.

Exemple 2.3 (Serveur d'impression) [29]

Afin d'illustrer le fonctionnement des Mobile nets, prenons l'exemple d'un système constitué d'un ensemble d'imprimantes, d'utilisateurs et d'un serveur d'impression. Les imprimantes disponibles envoient leurs noms et types (noir et blanc : *nb* ou couleur : *c*) à une place (un port) nommée *ready*. Les utilisateurs émettent leurs requêtes avec le nom du fichier et le type de l'imprimante à une place nommée *job.*

Prenons la configuration du système où une imprimante de type noir et blanc nommée *laser* est disponible et deux requêtes d'impression sont en attente : une requête d'impression du ficher *file1* sur une imprimante noir et blanc et une requête d'impression d'un fichier *file2* sur une imprimante couleur. Cette configuration est décrite par le marquage :

$$ready(laser, nb), job(file1, nb), job(file2, c)$$

Ce marquage correspond à un ensemble de *messages en attente*. Le rôle du serveur d'impression consiste à effectuer une opération d'appariement de motifs (pattern matching) afin d'associer l'imprimante disponible à la requête adéquate (en se basant sur le type de l'imprimante), puis à envoyer le fichier à l'impression. Ceci est décrit par la transition suivante :

$$ready(PRINTER, TYPE), job(FILE, TYPE) \triangleright PRINTER(FILE)$$

Dans cette expression, les noms écrits en majuscules représentent des *variables liées* qui seront remplacées par les paramètres effectifs récupérés à l'exécution. Ainsi, le tir de cette transition conduit à une nouvelle configuration décrite par le marquage :

$$job(file2, c), laser(file1)$$

2.4.2.1.1. MAGNETs

MAGNETS [30] est une variante des mobile nets adaptée à l'implémentation distribuée. Dans ce formalisme un système est modélisé par un ensemble de *systèmes réseaux* (net systems) qui sont des Mobile nets constitués de places locales, de transitions locales et d'un marquage. Le tir d'une transition peut conduire non seulement à la génération d'un nouveau marquage mais aussi à la génération de nouveaux composants dans le système. Ces composants nouvellement produits peuvent constituer eux même de nouveaux systèmes réseaux, comme ils peuvent être insérés dans un système réseau déjà existant pour étendre ses fonctionnalités. Ainsi, la configuration (structure et état) du système global évolue de manière continue durant le calcul.

Ce modèle permet de raisonner sur des abstractions de très bas niveau. Notons finalement qu'une machine virtuelle a été développée dans [30] pour interpréter les configurations définies dans le modèle des MAGNETs.

2.4.2.1.2. Bilan

Mobile Petri nets constitue l'un des premiers modèles qui ont été proposés pour supporter la mobilité en permettant le passage des noms des places dans un réseau de Petri. En se basant sur cette technique, le modèle parvient à capturer la connectivité dynamique des systèmes. Par ailleurs, cette technique nécessite des mécanismes pour la gestion des variables liées dans la définition formelle du modèle et de son fonctionnement. L'introduction de ces mécanismes, cependant, affecte le concept fondamental sur lequel se base la théorie des réseaux de Petri et qui suppose que les interactions entre les places et les transitions doivent être finies, locales et prévisibles.

Il a été démontré dans [29] que les mobile nets peuvent être simulés par des réseaux de Petri P/T. Cela signifie que les résultats de décidabilité des propriétés (accessibilité, couverture, …) restent valables pour les Mobile nets. En outre, lorsque l'ensemble des places et des transitions d'un mobile net est fini, alors celui-ci est également fini pour le réseau de Petri ordinaire associé.

2.4.2.2. Ubiquitous Nets et leurs extensions

Une autre famille de modèles basés sur les réseaux de Petri et dédiés au traitement de la mobilité a été récemment proposée par F. Rosa Verlardo et al [40- 43]. En réalité, les auteurs de ces formalismes insèrent leurs travaux dans le cadre général de l'informatique ubiquitaire (ubiquitous computing). Ils se positionnent cependant sur les axes liés à la mobilité, la sensibilité au contexte, la coordination, les approches orientées agents et se penchent tout particulièrement sur l'aspect sécurité.

Le modèle de base intitulé *ubiquitous nets* [40], est une variante des réseaux de Petri ordinaires dans laquelle a été introduite la notion de localité de manière explicite. Il se charge de l'aspect coordination, plus précisément l'offre et la demande de ressources et/ ou de services dans les systèmes mobiles. Ce modèle a été enrichi par l'introduction des couleurs pour constituer un support de mobilité flexible, et ce par l'adoption de la technique de passage de référence. Le modèle résultant, nommé *Mobile Synchronizing Petri Nets (MSPNs*) fournit déjà un premier mécanisme de sécurité en permettant de contrôler l'accès aux ressources. Afin de renforcer ce support de sécurité dans le modèle, une extension des MSPNs a été mise au point, il s'agit des *Abstract Mobile Synchronizing Petri Nets (AMSPNs)*. Cette extension est fondée sur le mécanisme de génération des noms abstraits qui est à la base des calculs nominaux et qui constitue un outil clé pour la modélisation de concepts liés à la sécurité.

2.4.2.2.1. Le modèle de base

Ubiquitous nets est un formalisme basé sur les réseaux de Petri ordinaires. Il met l'accent sur les aspects de mobilité et coordination, en particulier, l'offre et la demande des services et/ ou ressources. Ce modèle perçoit un système comme étant une collection de processus localisés modélisés par des *composants réseaux nommés* isolés. Certains de ces processus fournissent des services et d'autres en demandent l'utilisation. Les processus ont la faculté de se déplacer de localisation en localisation en fonction des ressources ou services qu'ils convoitent. Ce déplacement se fait grâce à des transitions dites de mouvement, étiquetées généralement par *go k* où *k* désigne le nom de la localisation destination (voir figure 2.9).

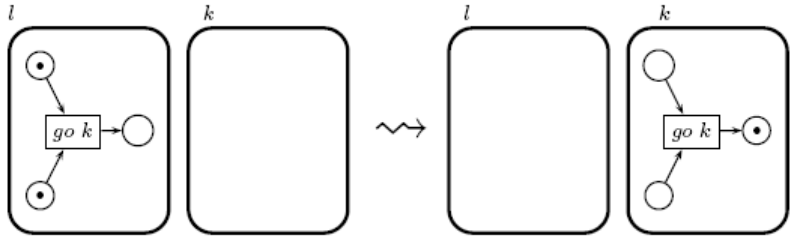

Fig.2.9 - Effet du tir de la transition de mouvement *go k*.

L'offre d'un service ne peut avoir lieu que si le processus fournisseur et le processus demandeur sont situés au niveau de la même localisation. Cette offre est réalisée par le tir simultané de deux transitions : Une *transition d'offre* dont l'étiquette est de la forme *s !* , où s est le nom du service en question, et une *transition de demande* dont l'étiquette est de la forme *s ?*. Les transitions *s !* et *s ?* ne peuvent en aucun cas être tirées individuellement et elles constituent lors du tir une transition unique qu'on nomme *transition synchronisée* ou encore *conjuguée*. Les opérations internes aux processus sont modélisées par des transitions dites *autonomes*. Les conditions du tir de ces transitions sont internes au composant réseau allant effectuer le tir (voir figure 2.10).

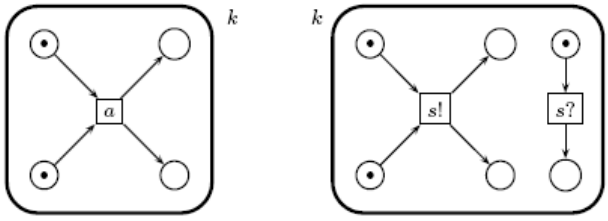

Fig.2.10 – Transition autonome (à gauche) et transitions de synchronisation (à droite).

Afin de pouvoir capturer la position courante des composants réseaux, une fonction de localisation *loc* est définie. Etant donné un composant réseau (modélisant un processus donné) cette fonction retourne le nom de sa localisation courante.

Ce formalisme de base a été enrichi dans [43] par une primitive inspirée de l'opérateur de réplication[5] employé dans les modèles d'algèbres de processus. L'effet de cette primitive

[5] Un opérateur de réplication exprime l'itération ou la récurrence et permet de modéliser des calculs infinis

consiste à créer un nouveau composant réseau dans le système ayant la même structure que le composant qui a invoqué la primitive et avec un certain marquage initial. Cette primitive permet de modéliser la technique de clonage des processus [07] largement utilisée dans le cadre de la mobilité du code.

Afin d'introduire de la flexibilité au modèle quant à la gestion des noms de localité, une amélioration lui a été apportée dans [41] par l'introduction de la notion de couleurs. Le modèle résultant est présenté dans la section suivante.

2.4.2.2.2. Mobile Synchronizing Petri Nets

Le modèle *Mobile Synchronizing Petri Nets MSPNs* [41] est une extension du modèle *Ubiqutous nets* dans laquelle a été introduite la notion de couleurs. Un MSPN est donc un type spécial des réseaux de Petri colorés avec seulement deux types de couleurs : Un type pour représenter des localités et l'autre pour représenter le jeton noir ordinaire. Contrairement aux réseaux de Petri colorés ordinaires où les arcs peuvent être annotés par des expressions ayant une syntaxe donnée [51], dans les MSPNs les arcs ne peuvent être étiquetés que par des variables pour spécifier le flux des jetons des pré-conditions et des post-conditions.

La représentation des noms de localité par des jetons (jetons de localités) permet d'enrichir significativement le modèle de base. En effet, les noms de localité peuvent désormais être communiqués entre les processus en mouvement à la π-calcul suite à des synchronisations. Cette acquisition dynamique des connaissances sur les destinations possibles forme un support pour la modélisation du *contrôle d'accès aux ressources*. En effet, un processus ne peut accéder à une ressource que s'il dispose du nom de la localité privée hébergeant cette ressource. Nous présentons ci-dessous un exemple dans lequel nous illustrons son fonctionnement.

Un *MSPN system* est défini comme étant un ensemble de composants réseaux MSPNs disjoints localisés. Ainsi, l'état d'un MSPN system est défini non seulement par les marquages des différents composants réseaux mais aussi par la fonction de localisation *loc* qui permet de déterminer l'emplacement actuel de chaque composant.

Exemple 2.4 [41]

Afin d'illustrer le fonctionnement du formalisme présenté ci-dessus, prenons l'exemple d'un système permettant d'organiser l'accès à un service SERV situé au niveau d'un ensemble de localités k∈ K. L'accès au service est autorisé à un groupe de processus pour lesquels la localité H est rendue publique.

Le système est constitué d'un processus *forwarder* situé initialement au niveau de la localité nommée H et d'un ensemble de processus fournisseurs du service, chacun situé au niveau d'une localité différente k∈ K.

Le schéma de la figure 2.11 illustre le fonctionnement du processus forwarder. Ce dernier a pour objectif d'orienter les processus clients, le souhaitant, vers une localité où ils recevront le service convoité.

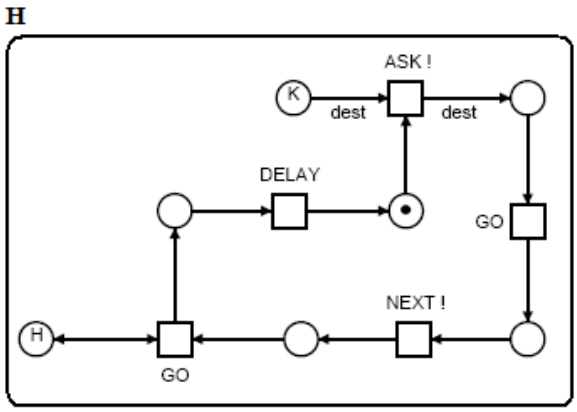

Fig.2.11- Processus forwarder.

Le schéma de la figure 2.12 capture une structure éventuelle d'un processus serveur pouvant fournir le service un nombre non limité de fois.

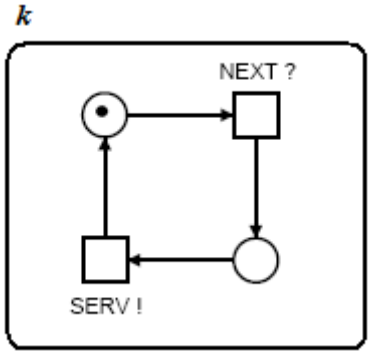

Fig.2.12- Processus serveur.

Pour accéder au service, le processus client doit d'abord se déplacer vers la localité H (qu'il doit connaître préalablement). A ce niveau, il va se synchroniser avec le processus forwarder via la transition synchronisée étiquetée *Ask*. A travers cette synchronisation, le forwarder va extraire un nom de localité de l'ensemble K qu'il va communiquer au client. L'information communiquée à travers cette synchronisation est considérée comme une permission attribuée au client pour pouvoir accéder au service voulu.

A chaque fois que le forwarder communique un nom de localité à un client, il se déplace vers cette localité pour informer le serveur de l'arrivée d'un nouveau client. Cela est réalisé grâce à une synchronisation via la transition conjuguée étiquetée *Next*. Une fois que le serveur est mis à jour par le forwarder, le client peut recevoir le service après son déplacement vers la localité du serveur en question. L'obtention du service est exprimée par la transition synchronisée étiquetée *SERV*.

La figure 2.13 montre la forme d'un processus client idéale, c'est-à-dire, qui interagit avec le système de la manière attendue. Les clients, cependant, peuvent avoir des comportements différents qui peuvent être illégaux dans certains cas.

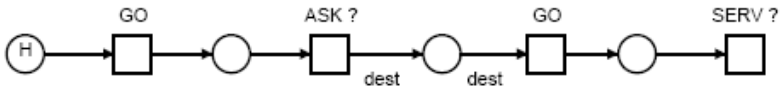

Fig.2.13- Client ayant un fonctionnement idéal.

Celui de la figure 2.14.a, par exemple, accède une première fois au service de façon légale (avec la permission du forwarder). Il se déplace, ensuite, vers une autre localité où il effectue une tâche quelconque, puis retourne à la localité du serveur où il attendra que celui-ci soit mis à jour par le forwarder (pour recevoir un autre client) pour accéder au service une autre fois.

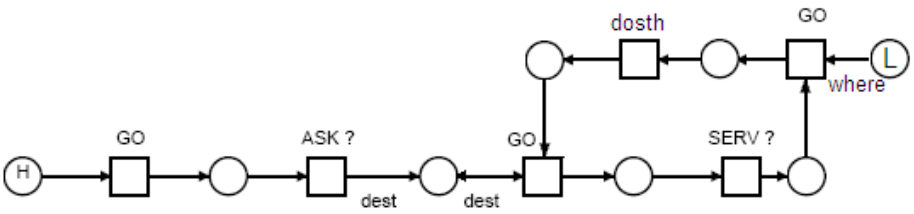

Fig.2.14.a - Autre forme de clients.

Le client de la figure 2.14.b est un autre cas d'utilisation abusive des permissions. Dans ce cas, le client acquière des permissions du forwarder qu'il va déléguer à des clients illégitimes (qui n'ont pas la connaissance préalable de la localité H) qui pourront par conséquent accéder au service.

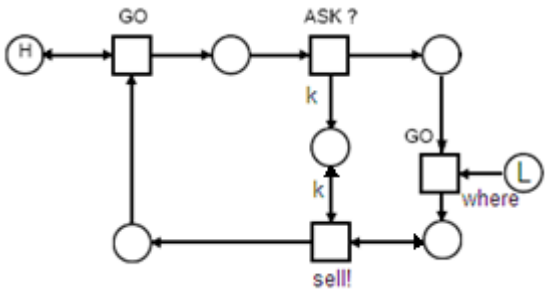

Fig.2.14.b- Autre forme de clients.

Si on suppose que le système est sûr lorsqu'à chaque action SERV correspond une occurrence différente de l'action ASK et que les partis qui se synchronisent doivent être les mêmes pour les deux actions, alors le modèle défini ne permet pas de garantir le fonctionnement correct du système.

Afin de pouvoir adresser des mécanismes d'authentification et de rendre, ainsi, le modèle plus robuste dans le sens de la sécurité, une autre extension à été proposée. Il s'agit des Abstract Mobile Synchronizing Petri Nets AMSPNs.

2.4.2.2.3. Abstract Mobile Synchronizing Petri Nets

Le modèle Abstract Mobile Synchronizing Petri Nets *AMSPNs* [42] étend le modèle précédent par la capacité de *génération de noms frais*. Ces noms peuvent être crées, communiqués entre composants et utilisés pour restreindre les synchronisations seulement entre des partis ayant la connaissance d'un nom particulier. Le formalisme AMSPNs est obtenu à partir des MSPNs en introduisant un troisième type de couleurs qui est le type des *identifiants*, et une transition spéciale étiquetée *new*. Le tir de cette transition permet la génération d'un identifiant frais dans le réseau.

Le modèle proposé peut être utilisé comme support pour la modélisation d'un mécanisme d'authentification qui consiste à prouver la possession d'une clé secrète provenant d'une certaine autorité et ce, en faisant abstraction des techniques cryptographiques employées.

2.4.2.2.4. Bilan

Les modèles présentés constituent trois versions d'un formalisme basé sur les réseaux de Petri dédié à la modélisation des systèmes mobiles, en particulier les aspects liés à la coordination et la sécurité.

Ces modèles se basent sur une approche compositionnelle pour la modélisation des systèmes. Chaque sous-système ou composant actif est représenté par un sous-réseau du réseau global, capable de communiquer avec le reste du système à travers des synchronisations de transitions. Le concept de localité dans ces modèles est manipulé explicitement. Ceci permet d'exprimer les évolutions qui affectent la dimension spatiale dans l'état d'un système et de raisonner sur les propriétés associées.

L'adoption de la technique de gestion des noms permet d'enrichir le modèle dans deux sens différents : d'une part elle permet au modèle de supporter la connectivité dynamique d'une manière très simple et intuitive, et d'autre part, elle permet de fournir un support pour le mécanisme d'authentification qui consiste à prouver la possession d'une information secrète.

Nous estimons que ces modèles, malgré leur simplicité, constituent un outil flexible et facile à manipuler pour la modélisation des systèmes mobiles. D'une part, on constate qu'il existe une nette séparation entre les différents acteurs qui constituent le système modélisé.

D'autre part, les localités jouent un rôle majeur en encapsulant la notion de synchronisation locale des transitions de façon similaire aux calculs destinés aux systèmes mobiles et distribués tels que l'algèbre des ambients mobiles. De plus, il existe une séparation explicite entre l'aspect sécurité (connaissance des localités et / ou identifiants) et l'aspect coordination (connaissance des noms de transitions de synchronisation qui peuvent être vues comme des primitives de coordination).

Il a été prouvé que les modèles Ubiquitous nets et MSPNs peuvent être simulés par les réseaux de Petri ordinaires [41]. Par conséquent, toutes les propriétés décidables dans les réseaux de Petri le sont également dans ces deux modèles. Dans le formalisme AMSPNs par contre, une source infinie de couleurs est introduite ce qui rend naturellement l'accessibilité indécidable. La couverture cependant reste décidable ce qui est suffisant pour vérifier les propriétés de sûreté usuelles [42, 44].

2.4.3. Bilan sur les modèles à base de réseaux de Petri

Les formalismes de réseaux de Petri dotés de techniques appropriées peuvent constituer un bon support pour la modélisation des systèmes mobiles, ceci tient aux avantages multiples qu'ils présentent. Depuis leur apparition, les réseaux de Petri ont fait l'objet d'un nombre important de travaux de recherche, ce qui a engendré une base théorique solide et riche de techniques et d'outils d'analyse. Par ailleurs, les réseaux de Petri constituent un formalisme basé à la fois évènement et états. Ils permettent par conséquent, de raisonner aussi bien sur les configurations successives d'un système que sur l'ensemble des évènements produits. En outre, les réseaux de Petri disposent d'une représentation graphique attrayante qui accroit la lisibilité et facilite la compréhension des modèles.

Le tableau comparatif 2.1 présenté ci-dessous, résume les caractéristiques essentielles supportées par les modèles étudiés ci-dessus selon la technique qu'ils adoptent.

Les modèles basés sur le paradigme nets within nets, de par leur nature, permettent de supporter des abstractions à base de composants et éventuellement de représenter la notion de localité. Cependant, la connectivité dynamique n'est pas prise en charge. Cette caractéristique fondamentale est par contre bien supportée par les formalismes basés sur la technique de génération des noms. Abstract Mobile Synchronizing Petri Nets est l'un de ces formalismes qui permet en plus de couvrir d'une manière très simple et intuitive la notion de localité et

permet de supporter quelques aspects de sécurité, plus particulièrement, le contrôle d'accès et l'authentification.

Techniques/ Modèles		**Abstraction à base de composants**	**Connectivité dynamique**	**Localité**	**Sécurité**
Nets within nets	Mobile EOS	+	-	+	-
	Petri Hyper nets	+	-	-	-
Gestion des noms	Mobile nets	+	+	-	-
	AMSPNs	+	+	+	+

Tableau 2.1 : Comparaison des modèles à base de réseaux de Petri.

A l'issue de l'étude que nous avons présentée, nous estimons que Ubiquitous nets et leurs extensions, les AMSPNs en particulier, malgré leur simplicité, parviennent à remplir l'ensemble des caractéristiques fondamentales et d'exigences présentées dans la section 2.2 du chapitre courant, ils constituent donc de bons modèles de description des systèmes mobiles basés sur les réseaux de Petri.

2.5. Autres approches

D'autres approches existent dans la littérature pour exprimer le fonctionnement des systèmes mobiles et raisonner sur leurs propriétés. Nous citons à titre d'exemple le modèle formel Mobile UNITY [32, 33] qui permet d'exprimer la migration des composants, leurs localisations et leurs interactions transitoires. Codeweave [52] est un autre formalisme qui est une spécialisation de Mobile UNITY qui permet de représenter la mobilité des fragments de code (variables, expressions, …).

D'autres travaux de recherche se sont orientés vers la définition de passerelles permettant le passage des modèles d'algèbres de processus vers des modélisations graphiques à base de réseaux de Petri. Ces traductions permettent d'une part de raisonner directement sur la concurrence et la causalité des systèmes, et permettent d'autre part de rendre le capital des

techniques et d'outils développés au sein des réseaux de Petri accessible aux concepteurs se basant sur les algèbres de processus. Nous citons dans ce sens, les approches de traduction compositionnelle présentées dans [20] et [28] permettant respectivement le passage des expressions du π-calcul et de Klaim vers des extensions de réseaux de Petri de haut niveau basées sur le modèle des M-nets[6] [53].

2.6. Conclusion

Dans ce chapitre, nous nous sommes intéressés aux formalismes dédiés à la description opérationnelle des systèmes mobiles. Nous avons d'abord décrit quelques formalismes basés sur les algèbres de processus, puis nous avons présenté une étude sur les formalismes basés sur les réseaux de Petri. A l'issue de cette étude, nous avons choisi le modèle Abstract Mobile Synchronizing Petri Nets sur lequel portera la contribution de ce travail. Le prochain chapitre est dévolu donc à la présentation détaillée de ce modèle.

[6] M-nets est une classe de réseaux de Petri colorés étiquetés et composables au moyen d'opérateurs similaires à ceux des algèbres de processus. L'un des objectifs de ce modèle est de faciliter la définition de la sémantique compositionnelle des langages de programmation concurrents.

Chapitre 3

Le modèle AMSPN

3.1. Introduction

AMSPN [42] est un type spécial des réseaux de Petri colorés munis de la capacité de gestion des noms. En effet, dans ce modèle, des noms peuvent être crées et communiqués entre des composants, comme ils peuvent être utilisés pour restreindre l'occurrence de la synchronisation seulement entre des entités ayant la connaissance d'un nom particulier. Ce modèle peut fournir un support pour la modélisation des mécanismes d'authentification faibles [42, 44, 45] tels que la politique Resurrecting Duckling[7] [56, 57]. Le chapitre courant est dévolu à la description de ce modèle.

3.2. Notations

Nous définissons pour un AMSPN les ensembles et les notations suivantes :

- *A* : L'ensemble des étiquettes des transitions autonomes. Cet ensemble inclut les étiquettes spéciales *go* et *new* associées respectivement aux transitions de mouvement et à la transition distinguée de génération d'identifiants frais.
- *L* : L'ensemble des noms de localité.
- *Id* : Un ensemble potentiellement infini d'identifiants.
- $Tokens = \{\bullet\} \cup L \cup Id$.
- *Var* : L'ensemble des variables tel que $Var = Var_{\bullet} \cup Var_L \cup Var_{Id}$ où $Var_{\bullet} = \{\varepsilon\}$, Var_Lest l'ensemble des *variables de localité et* Var_{Id} l'ensemble des variables d'identifiants. De plus, on identifie l'ensemble des variables distinguées $Var_{aut} \subseteq Var_{Id}$ dont l'utilité sera clarifiée plus loin.
- *S* : L'ensemble des noms de services.
- *S ! = {s ! / s∈ S}* : L'ensemble des étiquettes des transitions d'*offre de services.*
- *S ?={s ? / s∈S}* : L'ensemble des étiquettes des transitions de *demande de services*.

[7] Resurrecting Duckling est une implémentation d'un mécanisme d'authentification faible qui consiste à créer des liens asymétriques entre les entités, constituant une relation maître-esclave. Ces associations doivent être sécurisées et transitoires, c'est à dire qu'elles peuvent être rompues à tout instant.

- $\overline{\bullet}$: S ! ∪ S ? → S ! ∪ S ? : Une fonction de conjugaison telle que : $\overline{s!} = s?$ et $\overline{s?} = s!$.

3.3. Structure

AMSPN [42] est un type spécial des réseaux de Petri colorés constitué d'un ensemble de places, de transitions étiquetées, d'arcs étiquetés par des variables et seulement par celles-ci et d'une fonction *couleur* qui associe à chaque place un type de couleurs.

Définition 3.1 (AMSPN)

Un AMSPN est un réseau de Petri coloré étiqueté $N = (P, T, F, \lambda, C)$ *où :*

- *P et T sont respectivement les ensembles de places et de transitions du réseau avec* $P \cap T = \emptyset$.
- $F: (P \times T) \cup (T \times P) \rightarrow Var$ *: Une fonction définie de l'ensemble des arcs vers l'ensemble des variables.*
- $\lambda : T \rightarrow A \cup S! \cup S?$ *: Une fonction d'étiquetage des transitions.*
- $C: P \rightarrow \{\bullet, L, Id\}$ *: La fonction couleur définie sur les places et étendue aux arcs :* $C\big((t,p)\big) = C\big((p,t)\big) = C(p)$.

Les conditions suivantes doivent être respectées :

(i) *Pour chaque arc* $a \epsilon Dom(F), C(a) = x \Leftrightarrow F(a) \epsilon\, Var_x$ *pour* $x \epsilon \{\bullet, L, Id\}$.

(ii) *Pour chaque transition* $t \in T$ *telle que* $\lambda(t) \epsilon A \setminus \{New\} :\ post(t) \subseteq pre(t)$.

(iii) *Pour chaque transition* $t \epsilon T$ *telle que* $\lambda(t) = go : |\, \bullet t_L| = 1$.

(iv) $|\{t/\lambda(t) = New\}| \leq 1$ *et* $|t^{\bullet}_{Id}| \geq 1$.

Notons que :

- $\bullet t = \{p \ / \ (p,t) \epsilon F\}$
- $t\bullet = \{p \ / \ (t,p) \epsilon F\}$.
- $P_x = \{p / C(p) = x\}$ pour $x \epsilon \{\bullet, L, Id\}$.
- $\bullet t_x = \bullet t \cap P_x$.
- $t\bullet_x = t\bullet \cap P_x$.
- $post(t) = \{F(t,p)/p \epsilon t^{\bullet}_x \ où\ x \epsilon \{L, Id\}\}$.
- $pre(t) = \{F(p,t)/p \epsilon \bullet t_x \ où\ x \epsilon \{L, Id\}\}$.
- $Var(t) = post(t) \cup pre(t)$.

La condition (i) exprime que les arcs sont étiquetés par des variables du type des places qui leurs sont adjacentes. Pour des raisons d'homogénéité, on suppose dans la définition que chaque arc est étiqueté par une variable bien que nous n'avons besoin de variables que pour distinguer les occurrences des différents jetons d'identifiants et de localité . Pour cela, la variable distinguée ε a été introduite pour étiqueter les arcs adjacents aux places du type jeton ordinaire ($C(p) = \{\bullet\}$). Graphiquement, cette variable est omise.

La condition (ii) exprime le fait que les variables qui apparaissent dans les arcs sortants d'une transition autonome (à l'exclusion de la transition *New*) doivent également apparaître dans l'un des arcs entrants de cette transition. Cela revient à dire que les jetons de localité et les identifiants peuvent être consommés ou déplacés mais ne peuvent jamais être générés.

La condition (iii) exprime l'existence d'une place de localité distinguée unique associée à chaque transition de mouvement comme une place d'entrée.

La condition (iv) impose que chaque réseau AMSPN doit avoir au plus une transition *New*. Cette transition doit avoir au moins une place de sortie de type Id dans laquelle sera déposé l'identifiant nouvellement généré.

Définition 3.2 (Marquage d'un AMSPN)

Un marquage M d'un AMSPN $N = (P, T, F, \lambda, C)$ *est une fonction* $M: P \rightarrow MS(Tokens)$. *On dira qu'un marquage est légal si* $M(p_x) \epsilon MS(x) pour\ x \epsilon \{\bullet, L, Id\}$.

On note par Id(M) l'ensemble des identifiants dans M où $Id(M) = \bigcup_{p \epsilon P_{Id}} Id(M(p))$.

On définit un *AMSPN system* comme étant un ensemble de composants réseaux AMSPNs disjoints localisés. Ainsi, le marquage ou l'état d'un AMSPN system est défini non seulement par les marquages des différents composants réseaux mais aussi par la fonction de localisation *loc* qui permet de déterminer l'emplacement actuel de chaque composant.

Définition 3.3 (Etat d'un AMSPN system)

Un AMSPN system est un ensemble $N = \{N_i\}_{i=1}^{n}$ *de AMSPNs disjoints* $N_i = (P_i, T_i, F_i, \lambda_i, C_i)$. *Un état S d'un AMSPN system est une paire* (M, loc) *où*

- $M = (M_1, \dots, M_n)$ *tel que* M_i *est le marquage du AMSPN* N_i.

- loc : *N* $\rightarrow$*L est la fonction de localisation.*

On écrira :

- $P = \bigcup_{i=1}^{n} P_i$.
- $M(p)$ pour noter $M_i(p)$ lorsque $p \epsilon P_i$.
- $F(a)$pour noter $F_i(p)$ lorsque $a \epsilon F_i$.
- $\lambda(t)$ pour noter $\lambda_i(t)$ lorsque $t \epsilon T_i$.

Avec ces notations, nous pouvons considérer l'état d'un *AMSPN system* comme étant une paire (M, loc) avec $M: P \rightarrow MS(Tokens)$.

3.4. Fonctionnement

Comme pour tout réseau de Petri de haut niveau, dans les AMSPNs, une transition peut être tirée sous différents modes. Un mode spécifie les jetons qui constituent la pré-condition de la transition. Un mode σ est une fonction de l'ensemble des variables *Var* dans l'ensemble des couleurs *Tokens* (les valeurs que les jetons peuvent avoir) tel que $\sigma(x) = \bullet \Leftrightarrow x = \varepsilon$.

On présente dans ce qui suit les règles de tir des transitions autonomes, des transitions de synchronisation et de la transition étiquetée *New* [42].

Définition 3.4 (Tir d'une transition autonome)

Soit N un AMSPN system, $t \epsilon T_i$ *avec* $\lambda(t) = A \backslash \{new\}$ *et* $S = (M, loc)$ *un état de N. La transition t est sensibilisée par le marquage M sous le mode* σ *si et seulement si pour chaque place* $p \epsilon \bullet t : \sigma\big(F(p,t)\big) \epsilon M(p)$. *Son tir conduit à l'état S'* $= (M', loc)$ *qu'on définit comme suit :*

(i) *Pour chaque place p* : $M(p) = (M(p) \setminus \sigma(F(p,t))) \cup \sigma(F(t,p))$.

(ii) *Si* $\lambda(t) \neq go$ *alors* $loc = loc$ *sinon* $loc\big(N_j\big) = loc(N_i)$*pour tout* $j \neq i$ *et* $loc(N_i) = \sigma(F(p,t))$ *où* $p \epsilon \bullet t_L$.

Définition 3.5 (Tir de la transition new)

Soit N un AMSPN system, $t \epsilon T$ *avec* $\lambda(t) = New$, *Q est l'ensemble des places d'identifiants de la post-condition de la transition t.* $S = (M, loc)$ *est l'état de N. La transition t est*

sensibilisée par le marquage M sous le mode σ si pour chaque place $p \epsilon {\bullet}t$: $\sigma\big(F(p,t)\big) \epsilon M(p)$. Son tir conduit à l'état $S' = (M', loc)$ qu'on définit comme suit :

(i) *Pour chaque place* $p \epsilon P \setminus Q$: $M(p) = (M(p) \backslash \sigma(F(p,t))) \cup \sigma(F(t,p))$.

(ii) *Pout chaque place* $q \epsilon Q : M(q) = (M(p) \backslash \sigma(F(p,t))) \cup \{\eta\}$ *tel que* $\eta \notin Id(M)$.

(iii) $loc' = loc$.

Notons que même si les variables d'identifiants associées aux arcs de sortie sont différentes, les valeurs qui leurs sont attribuées sont identiques et correspondent au nom fraîchement généré η.

Etant donnée une paire de transitions $t_1 \epsilon T_1 et\ t_2 \epsilon T_2$, on note :

- $post(t_1, t_2) = post(t_1) \cup post(t_2)$.
- $pre(t_1, t_2) = pre(t_1) \cup pre(t_2)$.
- $Var(t_1, t_2) = post(t_1, t_2) \cup pre(t_1, t_2)$.

On utilise ces notations pour définir les conditions et les règles de tir des transitions de synchronisation lorsque celles-ci vérifient *la condition de compatibilité*.

Définition 3.6 (Condition de compatibilité)

On dit que deux transitions t_1, t_2 sont compatibles si :

(i) $\lambda(t_1) = \overline{\lambda(t_2)}$.

(ii) $post(t_1, t_2) \subseteq pre(t_1, t_2)$.

(iii) *Si* $z \epsilon pre(t_i) \cap Var_{Aut}$ *alors* $z \epsilon\ pre\big(t_j\big)$ *pour* $i, j\ \epsilon \{1,2\}, i \neq j$.

La condition de compatibilité est purement syntaxique : D'une part, leurs étiquettes doivent être conjuguées (condition i). D'autre part, les deux transitions réunies doivent obéir aux mêmes conditions de tir d'une transition autonome (condition ii). Cela signifie que chaque variable qui apparaît dans les arcs de sortie doit également apparaître dans les arcs d'entrée d'une des deux transitions synchronisées. La condition (iii) justifie la définition du sous ensemble Var_{Aut} des variables d'identifiants dites d'authentification. Lorsqu'une variable d'authentification étiquette un arc de pré-condition d'une transition de synchronisation, celle-ci doit également étiqueter un arc de pré-condition de la transition compatible associée. De

cette manière, on arrive à forcer l'appariement (matching) entre les valeurs associées à ces variables lors du tir synchronisé des transitions.

Le mode de tir σ d'une paire de transitions synchronisée est défini de l'ensemble $Var(t_1, t_2)$ dans l'ensemble des jetons *Tokens* avec $\sigma(x) = \bullet \Leftrightarrow x = \varepsilon$.

Définition 3.7 (Tir d'une transition synchronisée)

Soit N un MSPN system, t_i, t_j *deux transitions compatibles et* $S = (M, loc)$*un état de N. La paire de transitions* (t_i, t_j) *est sensibilisée par le marquage M sous le mode* σ *si pour chaque place* $p \epsilon \bullet t_h, \sigma\big(F(p, t_h)\big) \epsilon M(p)$ *pour* $h \epsilon \{i, j\}$*, son tir conduit à l'état* $S' = (M', loc)$ *qu'on définit comme suit :*

(i) $M(p) = (M(p) \setminus \cup_{h\epsilon\{i,j\}} \sigma(F(p, t_h))) \cup \cup_{h\epsilon\{i,j\}} \sigma(F(t_h, p))$.

(ii) $loc = loc$.

On constate qu'à travers la synchronisation, on parvient d'une part à réaliser la communication des noms, et d'autre part à restreindre des communications grâce à l'opération de matching. Si on suppose que t_1 et t_2 sont deux transitions compatibles, alors la communication est obtenue en utilisant une variable $z \epsilon\ post(t_1) \setminus pre(t_1) \cap pre(t_2)$ ou vice versa (voir figure 3.1).

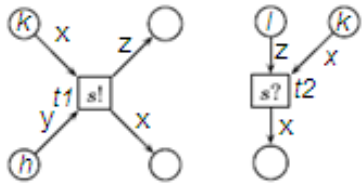

Fig.3.1- Transition de synchronisation.

La restriction des communications est obtenue en utilisant une variable $x \epsilon Var_{aut}$. La définition de la compatibilité impose que $x \epsilon\ pre(t_1) \cap pre(t_2)$ (voir figure 3.1). Ainsi, cette variable permettra de forcer le matching entre les jetons correspondants à cette variable lors

du tir simultané des transitions t_1, t_2. Cette technique peut être utilisée comme support pour la modélisation des mécanismes d'authentification comme illustré dans l'exemple qui suit.

Exemple 3.1 [42]

Dans cet exemple, on modélise un simple protocole d'exclusion mutuelle dans un contexte distribué. Le système est composé d'un fournisseur de tickets, un agent forwarder et un serveur critique illustrés respectivement dans les figures 3.2 et 3.3. Le fournisseur de tickets et l'agent forwarder sont situés initialement au niveau de la localité *L1*. Le serveur critique est un processus figé situé au niveau de la localité *l*.

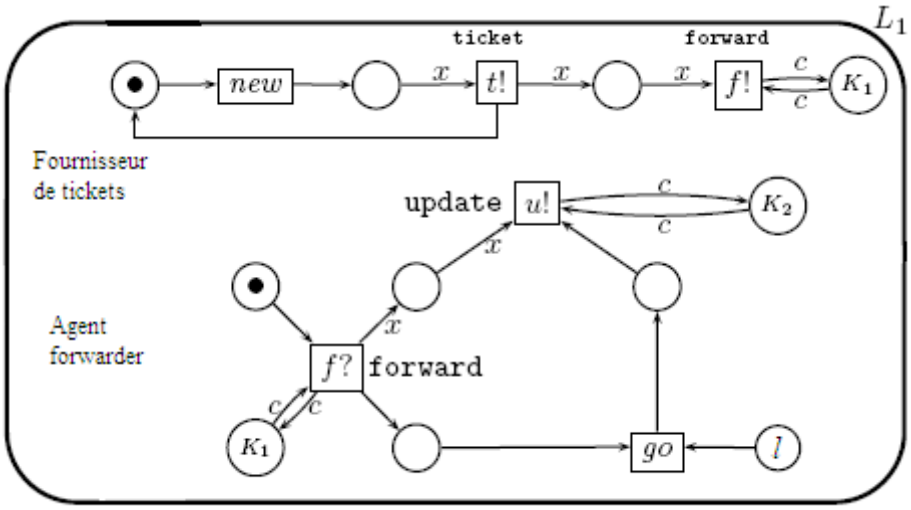

Fig.3.2- Fonctionnement du fournisseur de tickets et de l'agent forwarder.

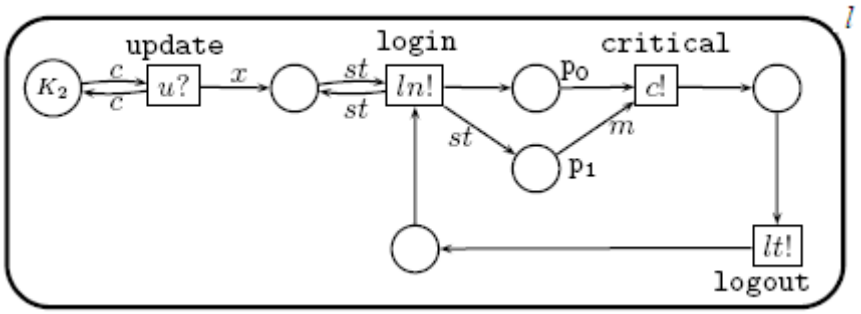

Fig.3.3- Fonctionnement du serveur critique.

Le protocole est amorcé par l'arrivée d'un client (qui peut avoir le fonctionnement illustré dans la figure 3.4) à la localité *L1* pour demander au fournisseur de tickets l'accès à la section critique. A ce moment la procédure ci-dessous est suivie :

- Le fournisseur de tickets génère un ticket (identifiant frais) suite au tir de la transition *New*. Il communique ce nom au client via la transition *ticket.*
- Le fournisseur de tickets se synchronise avec l'agent forwarder via la transition *forward* pour lui demander de mettre à jour le serveur critique en lui communicant le nouveau ticket valide. Le fournisseur de tickets et l'agent forwarder communiquent en utilisant la clé partagée *K1*. Ceci est formalisé en supposant simplement que la variable *c* appartient à l'ensemble Var_{Aut}.
- L'agent forwarder se déplace vers la localité *l* pour mettre à jour le serveur critique via la transition *update*. L'agent forwarder et le serveur critique communiquent en utilisant la clé partagée *K2*.
- Après avoir reçu le ticket, le client se déplace vers la localité *l* où il effectuera un *log in* en utilisant le ticket qui lui a été livré par le fournisseur de tickets, puis, il accède à la section critique en présentant une deuxième fois son ticket. Et enfin, il exécute une opération de *log out.*

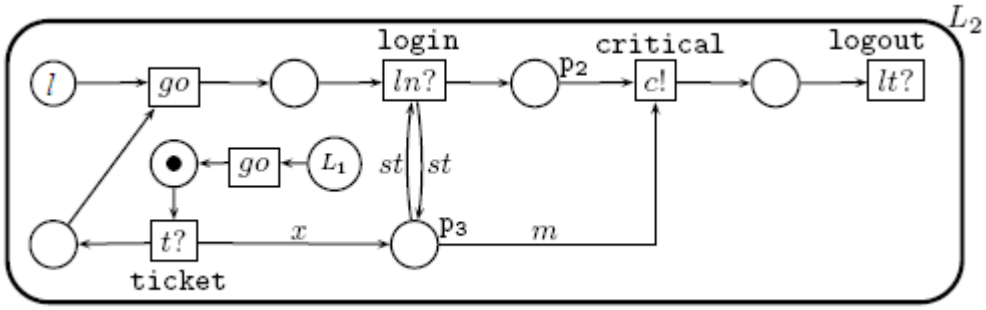

Fig.3.4- Fonctionnement d'un client.

Le système critique est dit sûr lorsque deux clients différents ne peuvent jamais accéder à la section critique en même temps. Autrement dit, le marquage $M(p_0) = 2\ et\ M(q) = \emptyset\ pour\ le\ reste\ des\ places$ ne doit pas être couvert. Comme la couverture est décidable dans le modèle AMSPN [42], la sûreté du système peut être prouvée automatiquement.

Grâce aux variables d'authentification Var_{Aut} et à la condition de compatibilité entre les transitions synchronisées, le modèle permet de garantir que le client qui accède à la section critique est bien celui qui a obtenu la permission.

3.5. Conclusion

Dans ce chapitre, nous avons présenté une description détaillée du formalisme Abstract Mobile Synchronizing Petri nets, sa structure et son fonctionnement. Ce modèle rassemble à la fois la simplicité et le pouvoir d'expression. Il perçoit un système mobile comme une collection de processus localisés et permet, entre autres, l'établissement de communications locales entre les processus et la restriction de ces communications, pour des raisons de sécurité, seulement à des partis ayant la connaissance d'une information secrète. Les communications étant souvent soumises à des délais stricts, il serait intéressant d'étendre le modèle présenté à la dimension temporelle afin de capturer ce genre de contraintes, ce qui fera l'objet du prochain chapitre.

Chapitre 4

Le modèle Timed AMSPN

4.1. Introduction

Dans les systèmes du monde réel, souvent les communications entre les composants sont contraintes par des délais stricts préalablement définis, par exemple pour des raisons de performance, pour éviter des situations de blocage (en faisant recours à la notion de Time-out) ou pour renforcer les mécanismes de sécurité (en imposant la fraicheur des clés d'authentification).

La contribution de ce travail s'insère dans ce cadre spécifique et consiste à proposer un formalisme pour la modélisation de systèmes mobiles qui soient en plus soumis à des contraintes temporelles. Le formalisme que nous proposons, intitulé *Timed Abstract Mobile Synchronizing Petri Nets (Timed AMSPN)*, est une extension du modèle AMSPN à la dimension temporelle. Nous adoptons pour cela, la sémantique du modèle de Van der Aalst [54], nommé *Interval Timed Colored Petri nets (ITCPN*), qui associe à chaque jeton crée un intervalle de temps indiquant son *domaine de validité*.

Dans un Timed AMSPN, un système est perçu comme une collection de composants actifs (processus) localisés. Le temps du système progresse d'une manière synchrone au niveau de tous les processus. Ce *temps abstrait* qu'on considère permet d'exprimer des contraintes sur l'instant d'occurrence des actions qui se produisent au sein du système. Cette notion de temps abstrait, adoptée généralement dans les formalismes de description [55], permet d'apporter des simplifications au niveau conceptuel où l'on s'intéresse à des représentations mettant en avant les aspects essentiels des systèmes et éliminant tout ce qui relève de la technique pure. Lors de l'implémentation de ces contraintes il est, cependant, nécessaire de prendre en considération la nature du contexte physique distribué des systèmes étudiés, caractérisés par l'absence de référentiel temporel global et la diversité des vitesses et des temps de réponse des processeurs.

Dans le modèle qu'on propose, on associe aux arcs en sortie des transitions, des intervalles de temps dits de *disponibilité*. Ainsi, lorsqu'un jeton est crée, un intervalle de disponibilité lui est associé. La borne inférieure de cet intervalle spécifie la date au plutôt à partir de laquelle le jeton pourra être consommé alors que la borne supérieure spécifie la date au plus tard. On associe également à chaque jeton nouvellement crée, une variable appelée *horloge*. Cette variable mesure le temps qui s'écoule depuis la création du jeton. Sa valeur est initialisée à zéro et elle augmente avec le temps jusqu'à ce que le jeton correspondant soit consommé ou devient *invalide* lorsque cette valeur excède la borne supérieure de l'intervalle qui lui est associé. Le jeton dans ce dernier cas ne pourra plus contribuer à l'évolution du système.

Le présent chapitre est dédié à la présentation du modèle Timed AMSPN, ses aspects statiques et son comportement dynamique.

4.2. Définition formelle du modèle Timed AMSPN

Un *Timed AMSPN* est un *AMSPN* tel que présenté dans le chapitre précédent, augmenté par la dimension temporelle. Comme dans le modèle AMSPN, on rencontre dans notre modèle trois types de couleurs : Le type des jetons noirs, le type des identifiants *Id* et le type *L* qui permet de représenter les localités. On distingue également deux catégories de transitions : Transitions autonomes et transitions de synchronisation. Parmi les transitions autonomes, on identifie les deux transitions particulières *go* et *new*. Dans le modèle Timed AMSPN on introduit la notion de *marquages temporels* constitués de *jetons colorés temporisés*. Ceux-ci sont obtenus à partir des jetons colorés ordinaires en leur affectant à leur création deux paramètres temporels : Une variable appelée *horloge* et un intervalle de temps dit *de disponibilité*. Nous reprenons pour notre modèle les notations suivantes :

- A : L'ensemble des étiquettes des transitions autonomes. Cet ensemble inclut les deux étiquettes spéciales *go* et *new*.
- L : L'ensemble des noms de localité.
- Id : L'ensemble des identifiants.
- $Tokens = \{\bullet\} \cup L \cup Id$.
- $Var = Var_{\bullet} \cup Var_L \cup Var_{Id}$ où $Var_{\bullet} = \{\varepsilon\}$.
- $Var_{aut} \subseteq Var_{Id}$.
- S : L'ensemble des noms de services.

- $S! = \{s! / s \in S\}$: L'ensemble des étiquettes des transitions d'*offre de services.*
- $S? = \{s? / s \in S\}$: L'ensemble des étiquettes des transitions de *demande de services*.
- $\bar{\bullet}: S! \cup S? \rightarrow S! \cup S?$: Une fonction de conjugaison telle que : $\bar{s!} = s?$*et* $\bar{s?} = s!$.

Définition 4.1 (Timed AMSPN)

Un Timed AMSPN est un réseau de Petri coloré étiqueté temporisé défini par le tuple $N_T = \langle \sum, P, T, C, \lambda, F, I, IM_T \rangle$ *tel que :*

- $\sum = \{\{\bullet\}, L, Id\}$ *est l'ensemble des domaines de couleurs.*
- *P est l'ensemble des places du réseau.*
- *T est l'ensemble des transitions du réseau tel que* $P \cap T = \emptyset$.
- $C : P \rightarrow \sum$ *est la fonction couleur définie sur les places et étendue aux arcs.* $C((t,p)) = C((p,t')) = C(p)$.
- $\lambda : T \rightarrow A \cup S! \cup S?$ *est la fonction d'étiquetage des transitions tel que* $\{go, new\} \subseteq A$.
- $F : P \times T \cup T \times P \rightarrow Var$ *est une fonction de l'ensemble des arcs dans l'ensemble des variables.*
- $INT = \{[y,z] \epsilon\, Q^+ \times (Q^+ \cup \{\infty\}) / y \leq z\}$ *est l'ensemble des intervalles où les bornes sont des nombres rationnels.*
- $I : T \times P \rightarrow INT$ *est une fonction qui associe un intervalle à chaque arc sortant d'une transition.*
- IM_T *est le marquage temporel initial du réseau.*

Les conditions suivantes doivent être respectées :

(i) *Pour chaque arc* $a \epsilon dom(F), C(a) = x \Leftrightarrow F(a) \epsilon\, Var_x\ pour\ x \epsilon \sum$.

(ii) *Pour chaque transition* $t \in T$ *telle que* $\lambda(t) \epsilon A \setminus \{New\} : post(t) \setminus \{\varepsilon\} \subseteq pre(t)$.

(iii) *Pour chaque transition* $t \epsilon T$ *telle que* $\lambda(t) = go : |\bullet t_L| = 1$.

(iv) $|\{t / \lambda(t) = New\}| \leq 1\ et\ |t^{\bullet}_{Id}| \geq 1$.

Ces conditions admettent les mêmes explications que celles présentées pour les AMSPNs dans la section 3.3 du chapitre précédent.

Dans le modèle Timed AMSPN, on associe à chaque arc $a = (t, p)$ un intervalle de temps. Cet intervalle sera affecté à tout jeton crée suite au tir de la transition *t* et transporté par l'arc *a*.

Notons que :

- $post(t) = \{F(t,p)/p \epsilon t\bullet\}$.
- $pre(t) = \{F(p,t)/p \epsilon \bullet t\}$.
- $Var(t) = post(t) \cup pre(t)$.
- $Var(t_i, t_j) = Var(t_i) \cup Var(t_j)$.
- $Var_x(t) = \{v \epsilon Var(t)\} \cap Var_x$.

4.2.1. Elément jeton temporisé

Dans un réseau de Petri coloré [51], un élément jeton (p, c) est une paire constituée de la position *p* du jeton dans le réseau tel que $p \epsilon P$, et de sa valeur *c* tel que $c \epsilon C(p)$. Cette représentation est introduite afin de pouvoir manipuler les marquages sous forme de multi ensembles au lieu de les manipuler en tant que fonctions définies sur l'ensemble des places. Dans la définition qui suit, nous adoptons cette représentation.

Définition 4.2 (élément jeton temporisé) [58]

Un élément jeton temporisé est un tuple $\langle (p, c), h, [a, b] \rangle$ *tel que :*

- $(p, c) \epsilon\ TE$ tel que $TE = (P \times Tokens)$;
- $h \epsilon\ R^+$;
- $[a, b] \epsilon\ INT$.

L'ensemble de tous les éléments jetons temporisés est donné par $(TE \times R^+ \times INT)$.

Un élément jeton temporisé est un élément jeton augmenté par deux paramètres temporels :

- La variable *horloge h* qui mesure le temps qui s'écoule depuis la création du jeton. Sa valeur est initialisée à zéro et elle augmente avec la *progression du temps*.

- L'intervalle de disponibilité dont la borne inférieure spécifie la date à partir de laquelle le jeton pourra être consommé et la borne supérieure spécifie la date au plus tard.

Lorsque l'horloge h d'un jeton excède la borne supérieure de son intervalle de disponibilité le jeton est dit *Invalide*.

4.2.2. Marquage temporel

Le marquage temporel M_T d'un Timed AMPSN est un multi ensemble d'éléments jetons temporisés : $M_T \epsilon MS(TE \times R^+ \times INT)$ [58]. Le marquage ordinaire correspondant M est obtenu en éliminant les paramètres temporels du marquage temporel M_T.

$$M = \sum_{((p,c),h,[a,b])\epsilon TE\times R^+\times INT} M_T\big((p,c),h,[a,b]\big) . (p,c).$$

Le marquage temporel initial IM_T est obtenu en associant à chaque élément jeton du marquage initial un intervalle de disponibilité et une horloge dont la valeur est égale à zéro $IM_T \epsilon MS(TE, \{0\}, INT)$.

4.3. Définition d'un Timed AMSPN System

On définit un *Timed AMSPN System* comme étant un ensemble de composants réseaux Timed AMSPNs disjoints localisés. Ainsi, l'état d'un Timed AMSPN System est défini, d'une part, par l'union des marquages temporels des différents composants réseaux qui le constitue, et d'autre part, par une fonction de localisation *loc* qui permet de déterminer l'emplacement actuel des composants.

Définition 4.3 (Etat d'un TAMSPN System)

Un Timed AMSPN System $\mathcal{N} = \{N_{Ti}\}_{i=1}^{n}$ *est une collection de Timed AMSPNs disjoints* $N_{Ti} = \langle \sum, P_i, T_i, C_i, \lambda_i, F_i, I_i, IM_{Ti} \rangle$.

Un état d'un Timed AMSPN System est une paire $s = (M_T, loc)$ *où :*

- $M_T = (M_{T1}, M_{T2}, \dots, M_{Tn})$ *avec* M_{Ti} *le marquage temporel de* N_{Ti} *pour* $1 \leq i \leq n$.
- $loc : \mathcal{N} \rightarrow L$ *est la fonction de localisation.*

De plus, le marquage initial du système est donné par : $IM_T = (IM_{T1}, IM_{T2}, \dots, IM_{Tn})$.

Comme nous considérons que les composants réseaux qui constituent le Timed AMSPN System sont deux à deux disjoints, alors il est possible d'adopter les notations suivantes sans qu'il y ait des ambiguïtés :

- $P = \bigcup_{i=1}^{n} P_i$.
- $T = \bigcup_{i=1}^{n} T_i$.
- $F(a)$ pour noter $F_i(a)$ lorsque $a \epsilon dom(F)$.
- $\lambda(t)$ pour noter $\lambda_i(t)$ lorsque $t \epsilon T_i$.
- $I(t,p)$ pour noter $I_i(t,p)$ lorsque $t \epsilon T_i \; et \; p \epsilon P_i$.

Avec ces notations, nous pouvons considérer le marquage M_T d'un Timed AMSPN System comme étant un élément de l'ensemble $MS\big((P \times Tokens) \times R^+, INT\big)$ ce qui simplifie sa manipulation.

4.4. Représentation graphique

Comme tout réseau de Petri, un Timed AMSPN est constitué de transitions représentées graphiquement par des rectangles, de places représentées par des cercles et d'arcs reliant entre les deux entités précédentes. Une transition est annotée par son nom suivi par son étiquette entre parenthèses. Une place est annotée par son nom suivi par son type entre parenthèses et les arcs sont annotés par des variables. Ceux en sortie des transitions sont annotés en plus par des intervalles de temps. Il est à noter que la variable ε et les intervalles $[0, \infty[$ sont omis graphiquement.

Un Timed AMSPN System est constitué d'un ensemble de localités nommées et de composants réseaux Timed AMSPNs pouvant se déplacer entre celles-ci. Une localité est représentée graphiquement par un rectangle à angles arrondis étiqueté par son nom.

Exemple 4.1

Dans cet exemple, nous considérons une abstraction d'un système qui permet d'organiser l'accès à un service SERV qui peut être fourni par un ensemble de serveurs situés au niveau d'un ensemble de localités $k_1, \dots, k_j \epsilon K$. On distingue les acteurs suivants :

- Un agent fournisseur de tickets AFT situés initialement au niveau de la localité nommée *H* (voir figure 4.1).
- Un ensemble de serveurs situés au niveau des localités de l'ensemble *K*. La figure 4.2 illustre le fonctionnement d'un serveur situé au niveau de la localité k_1.
- Des clients souhaitant l'accès au service SERV. La figure 4.3 illustre un exemple typique d'un client situé initialement au niveau de la localité nommée *l*.

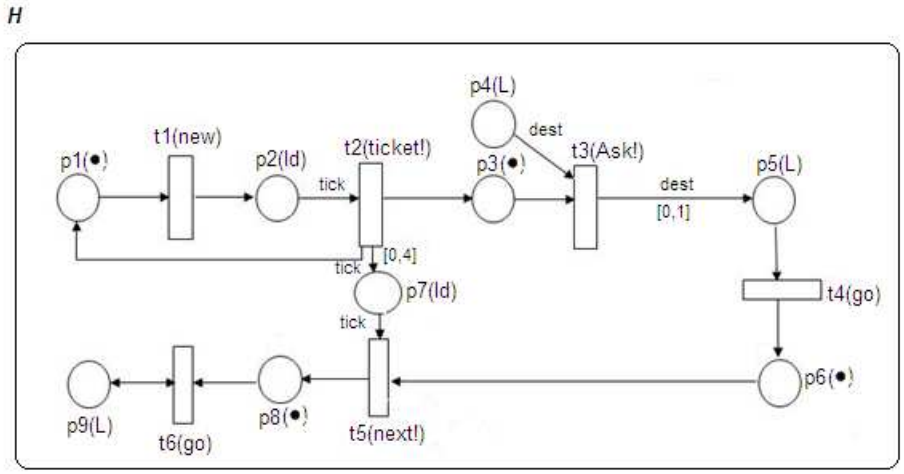

Fig.4.1- Fonctionnement de l'agent fournisseur de tickets.

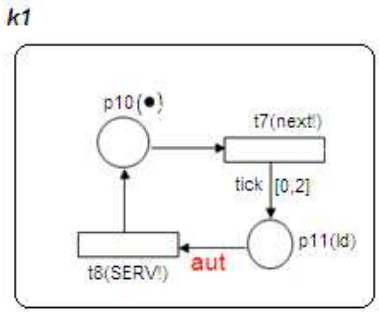

Fig.4.2-Fonctionnement du serveur.

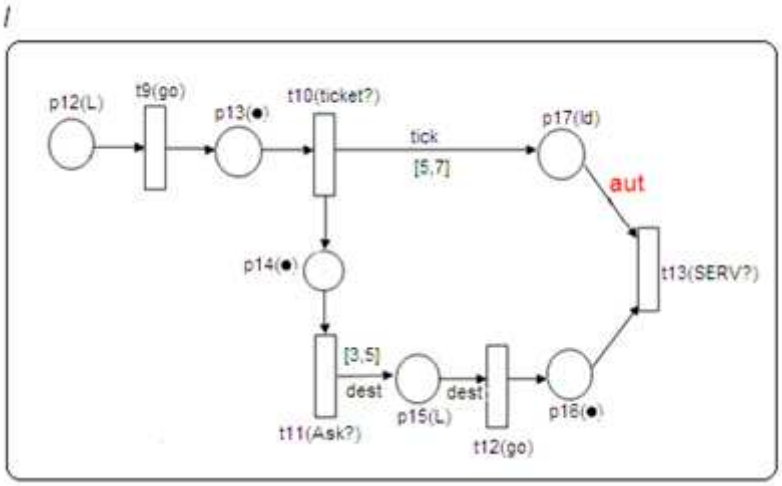

Fig.4.3- Fonctionnement d'un client.

Le système fonctionne comme suit : Lorsqu'un client souhaite accéder au service, il se déplace vers la localité *H*. A ce niveau, l'AFT va lui communiquer un ticket (nouvel identifiant) qu'il aurait généré par le tir de la transition *new*. Cette communication est accomplie grâce à la transition synchronisée étiquetée *ticket*. Ensuite, il se synchronise une deuxième fois avec le client à travers la transition étiquetée *Ask*. A travers cette synchronisation, l'AFT communique au client le nom de la localité du serveur qui lui fournira le service. L'AFT, migre ensuite vers cette localité pour mettre à jour le serveur en lui communiquant le ticket qui doit être présenté par le prochain client qui arrivera. Cela est réalisé grâce à l'action synchronisée *next*. Pour recevoir le service, le client doit d'abord migrer vers la localité du serveur, puis, il doit lui présenter le ticket qui lui a été fourni par l'AFT. Ceci est formalisé grâce à la variable *aut*. En effet, cette variable qui annote les arcs adjacents aux transitions d'offre et de livraison du service va permettre de forcer l'appariement (matching) entre les valeurs des éléments jetons de la pré-condition de ces transitions. Ainsi, le service ne peut avoir lieu que si le ticket que possède le client est identique à celui que possède le serveur. La variable *aut* est une variable d'authentification.

Dans cet exemple le Timed AMSPN System est constitué de trois composants réseaux :

- Le composant N_{T1} modélise l'agent fournisseur de tickets.
- Le composant N_{T2} modélise le serveur.
- Le composant N_{T3} modélise Le client.

On distingue dans le système les variables suivantes :

- $Var_L = \{dest\}$.
- $Var_{Id} = \{tick, aut\}$ avec $Var_{aut} = \{aut\}$.

Dans le système, il existe quatre transitions de mouvement (t_4, t_6, t_9, t_{12}) et quatre couples de transitions synchronisées :$(t_2, t_{10}), (t_3, t_{11}), (t_5, t_7)\ et\ (t_8, t_{13})$ qui correspondent respectivement aux actions synchronisées étiquetées *ticket*, *Ask*, *next*, *SERV*.

On associe aux arcs $(t_2, p_7), (t_3, p_5), (t_7, p_{11}), (t_{10}, p_{17}), (t_{11}, p_{15})$ respectivement les intervalles de temps $[0, 4], [0, 1], [0, 2], [5, 7], [3, 5]$.

L'état initial du système $s_0 = (IM_T, loc_0)$ est obtenu en considérant le marquage temporel initial du système et la position initiale de chacun des composants réseaux. Le marquage initial peut être donné comme suit :

$IM_T =$
$((p_2, d), 0, [0, \infty[) + ((p_4, k_1), 0, [0, \infty[) + ((p_4, k_2), 0, [0, \infty[) + ((p_9, H), 0, [0, \infty[) +$
$((p_{10}, \bullet), 0, [0, \infty[) + ((p_{12}, H), 0, [0, \infty[)$.

Pour des raisons de simplification, nous omettons les paramètres temporels des éléments jeton pour lesquels l'intervalle de tir est égal à $[0, \infty[$. Le marquage initial dans notre cas aura donc la forme suivante :

$IM_T = (p_2, d) + (p_4, k_1) + (p_4, k_2) + (p_9, H) + (p_{10}, \bullet) + (p_{12}, H)$.

De plus, nous avons : $loc_0(N_{T1}) = H,\ loc_0(N_{T2}) = k_1\ \ et\ \ loc_O(N_{T3}) = l$.

4.5. Comportement d'un Timed AMSPN System

Dans le modèle qu'on présente, l'état d'un système peut évoluer de deux manières : En faisant progresser le temps de façon synchrone au niveau de tous les composants réseaux ou en exécutant un évènement. L'exécution d'un évènement correspond au tir d'une transition autonome ou synchronisée sous un mode donné et sous certaines contraintes temporelles. Dans les deux sections suivantes, nous présentons la notion d'évènement, d'évènements valides, puis nous décrivons le fonctionnement du modèle.

4.5.1. Evènements d'un état

Comme dans les AMSPNs, une transition (qu'elle soit autonome ou synchronisée) peut être franchie sous différents modes. Un mode de tir est une fonction qui associe à chacune des variables d'une transition une valeur de son domaine de couleurs. Un évènement est défini par une transition *u* (autonome ou synchronisée), un mode de tir σ et respectivement les multi ensemble des éléments jetons temporisés consommés et produits suite au tir de la transition *u* sous le mode σ.

Définition 4.4 (Evènement)

Un évènement est un tuple $e = \langle u, \sigma, {}^{\bullet}e, e^{\bullet} \rangle$ *où*

- *u correspond à une transition autonome ou synchronisée.*
- *σ est définie sur l'ensemble des variables de u tel que :* $\forall\, v \epsilon Var_x(u)\ \sigma(v) = c$ *tel que* $x \epsilon \sum et\ \ c \epsilon x$. *Pour la variable distinguée* ε *nous avons :* $v = \varepsilon \Leftrightarrow \sigma(v) = \bullet$.
- ${}^{\bullet}e \epsilon MS(TE \times R^{+} \times INT)$ *est un multi ensemble qui indique les éléments jetons temporisés de la pré-condition de l'événement e ;*
- $e^{\bullet} \epsilon MS(TE \times \{0\} \times INT)$ *est le multi ensemble qui indique les éléments jetons temporisés de la post-condition.*

Si $u = t$ alors *u* est une transition autonome ($\lambda(u) \epsilon A$) et l'évènement *e* est dit *évènement autonome*. On distingue un évènement autonome particulier qu'on note e_{new} pour lequel $\lambda(t) = new$.

Si $u = (t_i, t_j)$ alors t_i et t_j sont deux transitions de synchronisation qui vérifient la *condition de compatibilité*. L'évènement *e* dans ce cas est dit *évènement synchronisé*.

Les deux multi-ensembles ${}^{\bullet}e$ et $e^{\bullet}$ d'un évènement *e* sont calculés en fonction du mode de tir σ et du type de l'évènement tel que présenté dans la section 4.5.1.1.

Un évènement est sensibilisé à l'état s=(M_T, loc) si ${}^{\bullet}e \leq M_T$. On note par E(s), l'ensemble de tous les évènements sensibilisés à l'état s. Cependant, un évènement sensibilisé ne peut être franchi que s'il est valide. Cette notion de validité d'évènements est expliquée dans la section 4.5.1.2.

4.5.1.1. Calcul de $\bullet e$ et $e\bullet$ en fonction du type de l'évènement e

On calcule la pré-condition $\bullet e$ et la post-condition $e\bullet$ d'un évènement e selon que celui-ci soit un évènement autonome ou synchronisé. On identifie le cas de l'évènement autonome distingué e_{new}.

a) Evènement Autonome

Si $e = \langle t, \sigma, \bullet e, e\bullet \rangle$ est un évènement autonome avec $\lambda(t) \epsilon A \setminus \{new\}$ alors :

- $\bullet e = \sum_{p \epsilon \bullet t} \Big(\big(p, \sigma(F(p,t)) \big), h, [a,b] \Big)$.
- $e\bullet = \sum_{p \epsilon t \bullet} \Big(\big(p, \sigma(F(t,p)) \big), 0, I(t,p) \Big)$.

Pour l'évènement distingué $e_{new} = \langle t, \sigma, \bullet e, e\bullet \rangle$ avec $\lambda(t) = new$, la pré-condition est calculée de la même manière que pour un événement autonome ordinaire. Cependant, la post-condition est calculée de la manière suivante :

On définit d'abord l'ensemble $Q = \{p / p \epsilon post_{Id}(t)\}$ et l'ensemble $R = P \setminus Q$.

- $e\bullet = e_R\bullet + e_Q\bullet$ tel que :
  - $e_R\bullet = \sum_{p \epsilon t \bullet \cap R} \Big(\big(p, \sigma(F(t,p)) \big), 0, I(t,p) \Big)$.
  - $e_Q\bullet = \sum_{p \epsilon Q} \big((p, \eta), 0, I(t,p) \big)$ tel que $\eta \notin Id(M_T)$.

$Id(M_T)$ est l'ensemble constitué des identifiants $d \epsilon Id$ tels que $(p,d) \epsilon M$ avec M le marquage ordinaire correspondant au marquage temporel M_T. Cela signifie que l'identifiant η n'apparait pas dans le marquage courant du réseau, c'est-à-dire qu'il est fraichement généré suite au franchissement de l'évènement e_{new}.

b) Evènement synchronisé

Si $e = \langle u, \sigma, \bullet e, e\bullet \rangle$ est un évènement synchronisé avec $u = (t_i, t_j)$ une paire de transitions qui vérifient la *condition de compatibilité* (voir la définition 3.6 du chapitre précédent) alors :

- $\bullet e = \sum_{p \epsilon \bullet t_k} \Big(\big(p, \sigma(F(p,t_k)) \big), h, [a,b] \Big)$ $avec\ k \epsilon \{i,j\}$.
- $e\bullet = \sum_{p \epsilon t_k \bullet} \Big(\big(p, \sigma(F(t_k,p)) \big), 0, I(t_k,p) \Big)$ $avec\ k \epsilon \{i,j\}$.

4.5.1.2. Evènements valides d'un état

Soit $< (p, v), h, [a, b] >$ un jeton temporisé de l'état s. Le *domaine de validité* qui lui est associé est défini par l'intervalle $[max(0, a–h), max\ (0, b–h)]$. Ce jeton n'est valide que si $(h \leq b)$. Autrement, le jeton n'est plus valide.

Soit *e* un évènement de l'état *s* $(e \epsilon E(s))$. Le délai de franchissement de l'évènement *e* est le temps requis pour que tous les jetons de la pré-condition définis par $\bullet e$ deviennent disponibles. Comme chaque jeton possède un intervalle de disponibilité qui lui est associé, il est plus judicieux de spécifier un délai de franchissement minimal FD_{min} et un autre maximal FD_{max} que de définir un temps de franchissement exact. Ces délais de franchissement sont définis en considérant l'intersection des domaines de validité des jetons de la pré-condition.

Définition 4.5 (délais de franchissement) [58]

Les délais de franchissement d'un évènement $e =< u, \sigma, \bullet e, e \bullet >$ *sont définis comme suit :*

$$FD_{min}(e) = max\ \left(0, max_{((p,v),h,[a,b])\epsilon \bullet e}(a–h)\right).$$

$$FD_{max}(e) = min_{((p,v),h,[a,b])\epsilon\ \bullet e}(b - h).$$

L'évènement *e* est dit *valide* si son délai de franchissement minimal est inférieur à son délai de franchissement maximal : $FD_{min}(e) \leq FD_{max}(e)$. On note par $E_v(s)$ l'ensemble des évènements valides à l'état s. Un évènement sensibilisé ne peut être franchi que s'il est valide.

Par convention, si $\bullet e$ est un multi ensemble vide alors $FD_{min}(e) = 0$ et $FD_{max}(e) = \infty$.

4.5.2. Evolution du modèle

A partir d'un état initial s_0, le modèle peut évoluer de deux manières : Par la progression du temps (pas continu) ou par le franchissement d'un évènement (pas discret) [58]. Dans le reste de ce paragraphe, nous présentons les règles qui régissent ces deux modes d'évolution dans le modèle Timed AMSPN.

a) Progression de temps (Pas continu)

Une première forme d'évolution du modèle est la progression du temps. En effet, à partir de n'importe quel état, le modèle peut subir une progression de temps. L'effet de cette progression de temps consiste à augmenter les valeurs de toutes les horloges associées aux éléments jetons temporisés de l'état courant par une valeur réelle positive.

Définition 4.6 (Progression de temps)

Une progression de temps de dh unités est réalisable à partir de l'état s, si et seulement si :

$$dh \leq min_{e \epsilon E_v(s)} FD_{max}(e).$$

On note cette progression par $s \xrightarrow{dh}$ *et on désigne le nouvel état atteint* $s' = (M_T', loc')$ *par* $\lfloor s \rfloor_{+dh}$*tel que :*

- $M_T{}' = \sum_{((p,v),h,[a,b]) \epsilon M_T} M_T\big((p,v),h,[a,b]\big).\ ((p,v),h+dh,[a,b]).$
- $loc' = loc\,.$

La progression de temps de *dh* unités n'est réalisable à un état *s* que si cette valeur est inférieure ou égale au délai de franchissement maximal de tous les évènements valides de l'état *s*. Après cette progression, l'horloge de chacun des jetons sera augmentée de *dh* unités. On écrira $s \xrightarrow{dh} s'$ si et seulement si l'état s' est accessible à partir de *s* par la progression de temps *dh*.

b) Franchissement d'évènements (Pas discret)

L'autre forme d'évolution du modèle est le franchissement d'évènements. Le franchissement a pour effet d'amener le système d'un état vers un autre par la consommation des éléments jetons temporisés de la pré-condition et la génération des éléments jetons temporisés de la post-condition de l'évènement franchi.

Définition 4.7 (Franchissement d'un évènement)

Soit $e_f = \langle u, \sigma, {}^\bullet e_f, e_f{}^\bullet \rangle$ *un évènement tel que* $e_f \, \epsilon E_v(s)$*. Cet évènement peut être franchi de deux manières :*

1. *L'évènement* e_f *est immédiatement franchissable (sans aucune progression) si et seulement si son délai de franchissement minimal est nul, c'est-à-dire* $FD_{min}\big(e_f\big) = 0.$

L'occurrence de cet évènement est instantanée et mène vers l'état $s' = (M_T', loc')$ *tel que :*

- $M_T' = M_T - {\bullet}e_f + e_f{\bullet}$;
- *Si* u *est un évènement autonome tel que* $u = t \epsilon T_i$ *et* $\lambda(t) = go$ *alors* $loc'(N_j) = loc(N_j)$ *pour tout* $j \neq i$ *et* $loc'(N_i) = \sigma(F(p,t))$ *où* $p \epsilon {\bullet}t_L$. *Sinon* $loc' = loc$.

On écrira : $s \xrightarrow{e_f} s'$.

2. *L'évènement* e_f *peut être franchi à l'état s après une progression de temps de dh unités si et seulement si :*

- *Son délai de franchissement minimal n'excède pas le délai maximal de chacun des autres évènements valides au niveau de l'état s. Autrement dit :* $FD_{min}(e_f) \leq min_{e \epsilon E_v(s)} FD_{max}(e)$.
- $dh \epsilon [FD_{min}(e_f), min_{e \epsilon E_v(s)} FD_{max}(e)]$.

L'occurrence de cet évènement est instantanée et mène vers l'état $s' = (M_T', loc')$ *tel que :*

- $M_T' = \lfloor M_T - {\bullet}e_f \rfloor_{+dh} + e_f{\bullet}$.
- loc'*est calculée de la même manière que dans le premier cas.*

On écrira : $s \xrightarrow{e_f}_{dh} s'$.

L'évolution d'un état *s* est une séquence d'évènements et de progressions de temps. En effet, les deux règles d'évolution présentées ci-dessus permettent de calculer les états et les relations d'accessibilité entre eux. Il est à noter que ces règles implémentent une sémantique de tir forte. Un évènement sensibilisé doit être franchi avant que les jetons de sa pré-condition ne deviennent invalides, à moins qu'il ne soit désensibilisé par le franchissement d'un autre évènement.

Exemple 4.2

Nous reprenons l'exemple 4.1 afin de dérouler les règles d'évolution du modèle. Pour des raisons de simplicité, nous représentons un évènement juste par la transition (autonome ou

synchronisée) qui le constitue et sa pré-condition. On prend pour état initial $s_0 = (IM_T, loc_0)$ tel que :

- $IM_T = (p_2, d) + (p_4, k_1) + (p_4, k_2) + (p_9, H) + (p_{10}, \bullet) + (p_{12}, H)$.
- $loc_0(N_{T1}) = H, loc_0(N_{T2}) = k_1 \; et \; loc_0(N_{T3}) = l$.

Au niveau de cet état, il existe un seul évènement franchissable $e_0 = \big(t_9, (p_{12}, H)\big)$. Cet évènement correspond au déplacement du client vers la localité de l'agent fournisseur de tickets AFT. Les délais associés à cet évènement sont : $FD_{min}(e_0) = 0$ et $FD_{max}(e_0) = \infty$.

Comme $FD_{min}(e_0) \leq FD_{max}(e_0)$ donc e_0 est le seul évènement sensibilisé et valide de l'état s_0 $(E_v(s_0) = \{e_0\})$. De plus, il est immédiatement franchissable car $FD_{min}(e_0) = 0$. Son franchissement conduit à l'état $s_1 = (M_T^1, loc_1)$ tel que :

- $M_T^1 = (p_2, d) + (p_4, k_1) + (p_4, k_2) + (p_9, H) + (p_{10}, \bullet) + (p_{13}, \bullet)$.
- $loc_1(N_{T1}) = H, loc_1(N_{T2}) = k_1 \; et \; loc_1(N_{T3}) = H$.

Au niveau de l'état s_1, il existe un seul évènement valide $e_1 = \big((t_2, t_{10}), (p_2, d) + (p_{13}, \bullet)\big)$. Cet évènement correspond à l'action synchronisée *ticket*. Il peut être immédiatement franchi et son franchissement conduit à l'état $s_2 = (M_T^2, loc_2)$ tel que :

- $M_T^2 =$
  $(p_1, \bullet) + (p_4, k_1) + (p_4, k_2) + (p_9, H) + (p_{10}, \bullet) + (p_3, \bullet) + \big((p_7, d), 0, [0,4]\big) +$
  $(p_{14}, \bullet) + \big((p_{17}, d), 0, [5,7]\big)$.
- $loc_2 = loc_1$.

Au niveau de l'état s_2, il existe trois évènements valides : e_2, e_2' qui correspondent à l'action synchronisée Ask et e_{new} qui correspond à la génération d'un nouveau ticket (identifiant). Ces évènements sont définis comme suit :

- $e_2 = \big((t_3, t_{11}), (p_3, \bullet) + (p_4, k_1) + (p_{14}, \bullet)\big)$.
- $e_2' = \big((t_3, t_{11}), (p_3, \bullet) + (p_4, k_2) + (p_{14}, \bullet)\big)$
- $e_{new} = \big(t_1, (p_1, \bullet)\big)$.

Les trois évènements sont immédiatement franchissables. De plus, il est possible d'effectuer une progression de temps de deux unités, car $min_{e \epsilon E_v(s_2)} FD_{max}(e) = \infty$.

On choisit d'exécuter le pas continu de $dh = 2$, ce qui conduit à l'état $s_2' = (M_{T2}', loc_2')$ tel que :

- $M_T^{2\prime} =$
  $(p_1, \bullet) + (p_4, k_1) + (p_4, k_2) + (p_9, H) + (p_{10}, \bullet) + (p_3, \bullet) + \big((p_7, d), 2, [0{,}4]\big) +$
  $(p_{14}, \bullet) + \big((p_{17}, d), 2, [5{,}7]\big).$
- $loc_2' = loc_2.$

Les évènements valides à l'état s_2 restent ainsi pour l'état s_2'. Comme dans notre exemple nous modélisons le serveur de la localité k_1, nous allons choisir d'exécuter l'évènement e_2, ce qui conduit à l'état $s_3 = (M_T^3, loc_3)$ tel que :

- $M_T^3 =$
  $(p_1, \bullet) + (p_4, k_2) + (p_9, H) + (p_{10}, \bullet) + \big((p_7, d), 2, [0{,}4]\big) + \big((p_{17}, d), 2, [5{,}7]\big) +$
  $\big((p_5, k_1), 0, [0{,}1]\big) + \big((p_{15}, k_1), 0, [3{,}5]\big).$
- $loc_3 = loc_2'.$

Au niveau de l'état s_3, il existe trois évènements valides :e_{new}, e_3 qui correspond au déplacement de l'AFT vers la localité du serveur et e_4 qui correspond au déplacement du client vers la localité du serveur également. Les deux derniers évènements sont définis comme suit :

- $e_3 = \Big(t_4, \big((p_5, k_1), 0, [0{,}1]\big)\Big).$
- $e_4 = \Big(t_{12}, \big((p_{15}, k_1), 0, [3, 5]\big)\Big).$

Les délais associés à ces deux évènements sont donnés par :

- $FD_{min}(e_{new}) = 0, FD_{max}(e_{new}) = \infty.$
- $FD_{min}(e_3) = 0, FD_{max}(e_3) = 1.$
- $FD_{min}(e_4) = 3, FD_{max}(e_4) = 5.$

L'évènement e_3 peut être immédiatement franchi alors que l'évènement e_4 ne peut être franchi qu'après l'écoulement d'au moins trois unités de temps. Il est également possible d'effectuer un pas continu d'une unité de temps car $min_{e \epsilon E_v(s_2)} FD_{max}(e) = 1$. L'exécution du pas continu conduit à l'état $s_3' = (M_T^{3\prime}, loc_3')$ tel que :

- $M_T^{3'} =$
  $(p_1, \bullet) + (p_4, k_2) + (p_9, H) + (p_{10}, \bullet) + \big((p_7, d), 3, [0,4]\big) + \big((p_{17}, d), 3, [5,7]\big) +$
  $\big((p_5, k_1), 1, [0,1]\big) + \big((p_{15}, k_1), 1, [3,5]\big)$.
- $loc_3' = loc_3$.

Les évènements valides à l'état $s_3$restent valides à l'état s_3'. Les délais de franchissement associés aux évènements $e_3$et e_4 sont donnés par :

- $FD_{min}(e_{new}) = 0, FD_{max}(e_{new}) = \infty$.
- $FD_{min}(e_3) = 0, FD_{max}(e_3) = 0$.
- $FD_{min}(e_4) = 2, FD_{max}(e_4) = 4$.

L'évènement e_4 ne peut toujours pas être franchi car $FD_{min}(e_4) = 2$. On exécute l'évènement $e_3$qui est immédiatement franchissable, ce qui conduit à l'état $s_4 = (M_T^4, loc_4)$.

- $M_T^4 =$
  $(p_1, \bullet) + (p_4, k_2) + (p_9, H) + (p_{10}, \bullet) + \big((p_7, d), 3, [0,4]\big) + \big((p_{17}, d), 3, [5,7]\big) +$
  $\big((p_{15}, k_1), 1, [3,5]\big) + (p_6, \bullet)$.
- $loc_4(N_{T1}) = k_1, loc_4(N_{T2}) = k_1, loc_4(N_{T3}) = H$.

L'ensemble des évènements valides à l'état s_4 est constitué des évènements e_{new}, e_4 et de l'évènement nouvellement sensibilisé $e_5 = \big((t_5, t_7), \big((p_7, d), 3, [0,4]\big) + (p_{10}, \bullet)\big)$ qui correspond à l'action synchronisée étiquetée *next.* Les délais de franchissement associés aux évènements e_4 et e_5 sont donnés par :

- $FD_{min}(e_{new}) = 0, FD_{max}(e_{new}) = \infty$.
- $FD_{min}(e_4) = 2, FD_{max}(e_4) = 4$.
- $FD_{min}(e_5) = 0, FD_{max}(e_5) = 1$.

L'évènement e_5 peut être franchi après une progression de temps d'une unité de temps $(dh = 1)$ car d'une part, $FD_{min}(e_5) \leq min_{e \epsilon E_v(s_4)} FD_{max}(e)$ et d'autre part $dh \epsilon \big[FD_{min}(e_5), min_{e \epsilon E_v(s_4)} FD_{max}(e)\big]$. L'occurrence de cet évènement conduit à l'état $s_5 = (M_T^5, loc_5)$ tel que :

- $M_T^5 = (p_1, \bullet) + (p_4, k_2) + (p_9, H) + \big((p_{17}, d), 4, [5,7]\big) + \big((p_{15}, k_1), 2, [3,5]\big) +$
  $(p_8, \bullet) + \big((p_{11}, d), 0, [0,2]\big)$.

- $loc_5 = loc_4$.

Nous pouvons constater que l'évènement e_4 ne peut en aucun cas être franchi avant l'évènement e_5, ce qui signifie que le client ne peut se déplacer vers le serveur qu'après que celui-ci soit mis à jour par l'AFT.

Au niveau de l'état s_5, il existe trois évènements valides : e_{new}, e_4 et l'évènement nouvellement sensibilisé $e_6 = \big(t_6, (p_8, \bullet)\big)$ associé au déplacement de l'AFT vers sa localité initiale. Les délais associés à ces évènements sont donnés par :

- $FD_{min}(e_{new}) = 0, FD_{max}(e_{new}) = \infty$.
- $FD_{min}(e_4) = 1,\ FD_{max}(e_4) = 3$.
- $FD_{min}(e_6) = 0,\ FD_{max}(e_6) = \infty$.

On choisit d'exécuter l'évènement e_6 ce qui conduit à l'état $s_6 = (M_T^6, loc_6)$ tel que :

- $M_T^6 = (p_1, \bullet) + (p_4, k_2) + (p_9, H) + \big((p_{17}, d), 4, [5,7]\big) + \big((p_{15}, k_1), 2, [3,5]\big) + \big((p_{11}, d), 0, [0,2]\big)$.
- $loc_6 = loc_1$.

Au niveau de l'état s_6, seuls les deux évènements $e_{new}\ et\ e_4$ sont valides. On peut choisir d'exécuter un pas continu de *dh* unités de temps pour $dh \leq 3$. Nous constatons que pour une valeur de *dh* égale à 2, le franchissement de l'évènement e_4 est immédiat. Par contre le choix de faire progresser le temps de trois unités bien qu'il soit permis, va causer l'expiration du jeton localisé au niveau de la place p_{11}car son horloge aura la valeur 3 qui excédera la borne supérieure de son intervalle de disponibilité qui est égale à 2. Ainsi, l'action synchronisée SERV ne peut plus être exécutée.

Sur le plan modélisation, l'apport de l'introduction de la dimension temporelle au modèle AMSPN apparait clairement à travers cet exemple. En effet, dans le système ainsi défini, un client ne doit pouvoir se déplacer vers le site du serveur qu'après la mise à jour de ce dernier par l'AFT. En plus, il ne peut accéder au service que pendant une durée de temps bien définie (Time-out), une fois cette durée expire, le client perd son droit d'accès au service. Cela permettra une meilleure régularisation d'accès au service et permet d'éviter une situation d'engorgement au niveau du serveur.

4.6. Analyse

Les techniques existantes qui peuvent être employées pour l'analyse du comportement dynamique des réseaux de Petri colorés temporisés peuvent être classifiées en deux catégories : La simulation et l'analyse énumérative.

La simulation consiste à construire des traces du système modélisé. Il s'agit d'une bonne technique d'analyse car elle est généralement simple et permet de faire des mesures de performances. Toutefois, cette technique peut être dans certains cas très couteuse en termes de temps processeur. Un autre inconvénient de la simulation est qu'elle ne permet pas de prouver que le système modélisé possède l'ensemble des propriétés souhaitées. Cependant, les développements technologiques récents stimulent cette technique d'analyse. D'une part, la puissance de calcul permet la simulation de réseaux de grande taille et d'autre part, les écrans graphiques modernes sont caractérisés par leur rapidité et leur haute résolution, ce qui rend possible la visualisation de la simulation graphique (animations).

L'analyse énumérative est basée sur la construction de l'espace des états du système modélisé. L'idée fondamentale de l'espace des états consiste à organiser tous les états accessibles du système en un graphe, où chaque nœud représente un état et chaque arc représente un pas transformant un état en un autre. Le graphe des états renferme toutes les informations concernant le comportement dynamique d'un système. A cause de la densité du temps, les graphes des états associés aux réseaux de Petri temporisés sont généralement infinis. Pour cela, des méthodes de contraction de l'espace des états sont généralement proposés [59, 60].

Le modèle que nous proposons Timed AMSPN, est un type spécial des réseaux de Petri colorés (AMSPN) étendu par la dimension temporelle. Le graphe des états pour un AMSPN System est infini avant même d'introduire le facteur temps. Ceci est dû à la considération de l'ensemble potentiellement infini des identifiants *Id*. Par conséquent, le graphe des états d'un Timed AMSPN System peut être infini pour deux raisons : D'une part, le nombre infini de couleurs possibles et d'autre part, la nature dense du temps. Pour cela, la technique de simulation est considérée comme la plus appropriée pour l'analyse d'un Timed AMSPN système. L'élaboration d'un outil de simulation offrira la possibilité d'effectuer des mesures de performances des systèmes mobiles, ce qui permet entre autres, d'identifier les valeurs des paramètres temporels (Time-out par exemple) permettant d'offrir une meilleure rentabilité des systèmes. Il permettra également de visualiser le comportement des systèmes

mobiles (migration des composants entre les localités, communication entre composants, …). Notons finalement, que l'aspect analyse dépasse le cadre de ce mémoire et sera abordé de façon plus ample dans nos futurs travaux.

4.7. Conclusion

Dans le présent chapitre, nous avons présenté une nouvelle extension du modèle AMSPN obtenue par l'introduction de la dimension temporelle en associant des intervalles de temps aux jetons. Le modèle résultant est intitulé Timed AMSPN.

L'introduction du facteur temps a un double impact : Sur le plan modélisation, il permet de capturer le fonctionnement des systèmes mobiles soumis à des contraintes temporelles. Il permet, entre autres, de servir de support pour la modélisation des mécanismes d'authentification impliquant des contraintes temporelles (prouver la possession d'un élément d'identification tout en imposant sa fraicheur). Sur le plan analyse qui sort du cadre de ce travail, l'élaboration d'un outil de simulation permettra de visualiser graphiquement le fonctionnement des systèmes mobiles modélisés et de mesurer leurs performances.

Conclusion générale

Ce mémoire porte sur les formalismes de description dédiés à la modélisation des applications réparties conçues sur la base des paradigmes du code mobile, le schéma des agents mobiles en particulier. Dans ce travail, nous avons d'abord réalisé une étude des formalismes existants, et ce après avoir exhibé les caractéristiques essentielles qui doivent être supportées par ceux-ci. Sur la base de ces caractéristiques, nous avons pu déduire à l'issue de notre étude que le formalisme AMSPN constitue un bon support pour la modélisation des systèmes mobiles malgré sa simplicité.

Dans le modèle AMSPN, un système mobile est perçu comme une collection de composants actifs localisés. Les localités sont manipulées explicitement dans le modèle, ce qui permet d'exprimer les évolutions qui affectent la dimension spatiale de l'état du système et de raisonner sur les propriétés associées. L'adoption de la technique de gestion des noms permet d'une part de capturer la connectivité dynamique qui est un aspect fondamental dans les systèmes mobiles. Elle fournit, d'autre par, un support pour les mécanismes d'authentification et de contrôle d'accès.

Partant du constat que les systèmes mobiles peuvent être soumis à des contraintes temporelles, nous avons proposé d'étendre le modèle AMSPN par l'introduction de la dimension temporelle. Le modèle résultant intitulé Timed AMSPN, permet donc de capturer le fonctionnement des systèmes mobiles où le temps apparait comme un paramètre quantifiable. Ce modèle permet, entre autres, de supporter la modélisation des mécanismes d'authentification impliquant des contraintes temporelles (prouver la possession d'un élément d'identification en imposant sa fraicheur).

Le travail que nous avons présenté dans ce mémoire comporte plusieurs pistes de recherche à explorer. La première serait de raffiner la notion de localité dans le modèle AMSPN en lui donnant une sémantique bien claire qui dépendra éventuellement du domaine d'application et donc de la nature des localités considérées. Il serait également intéressant de mettre en œuvre et d'explorer de nouvelles méthodes d'analyse des réseaux de Petri de façon à gérer les localités de manière explicite. Cela permettra de vérifier les propriétés spatiales d'un système mobile, par exemple, un composant donné a la possibilité de se déplacer de

localité en localité mais il peut toujours revenir à sa localité d'accueil initiale. Une suite naturelle à ce moment, serait de développer un outil d'analyse implémentant ces nouvelles méthodes proposées.

Un autre axe intéressant à explorer serait d'étendre le modèle AMSPN par de nouvelles primitives et éventuellement de nouveaux types de couleurs pour représenter explicitement la double mobilité : Physique des dispositifs matériels et logique des entités calculatoires. Notons que l'étude de cadres formels pour le traitement unifié de cette double mobilité est au cœur d'un nombre de travaux de recherche dont l'un est présenté dans [61].

Par ailleurs, le modèle que nous avons proposé dans ce mémoire explore un autre axe de recherche et peut être étendu dans plusieurs directions. Il serait intéressant, par exemple d'introduire le facteur temps de manière plus sophistiquée et qui s'adapte mieux au contexte distribué et au domaine d'application en affectant, par exemple une horloge à chaque localité puis en proposant des techniques pour synchroniser ces horloges. Il serait intéressant également d'élaborer un outil de simulation pour le formalisme Timed AMSPN proposé. Un tel outil permettra de mesurer les performances des systèmes mobiles modélisés avant leur implémentation et de visualiser leur fonctionnement.

En conclusion, du travail important reste à faire pour fournir un cadre formel permettant une meilleure exploitation des possibilités offertes par la technologie du calcul mobile, les agents mobiles en premier lieu, dans le contexte de plus en plus fréquemment répandu des réseaux de communication dynamiques.

Annexe A

Les Réseaux de Petri

Les réseaux de Petri [47] sont un formalisme graphique largement répandu pour la modélisation du comportement des systèmes concurrents. Ils allient les avantages d'une description graphique et ceux d'une description formelle qui, d'une part, met à l'écart toute ambiguïté et qui d'autre part, permet l'analyse par les nombreux outils et techniques mis en place, ce qui les a rendus très appropriés à de vastes domaines d'application.

Un réseau de Petri (RdP) est un graphe bipartite, les deux classes de sommets qu'il réunit sont les *places* et les *transitions*. Ces éléments représentent respectivement les états possibles du système et les actions susceptibles de modifier ces états. Les seuls arcs valides sont ceux reliant une place et une transition. Les arcs sont orientés et étiquetés par une valuation entière. Une place est dite en entrée d'une transition s'il existe un arc orienté de la place vers la transition. Réciproquement, une place est dite en sorite d'une transition s'il existe un arc allant de la transition à la place. Un réseau de Petri est défini formellement comme suit :

Définition A.1 (Réseau de Petri)

Un réseau de Petri est un triplet $N = (P, T, W)$ *tel que :*

- *P est l'ensemble des places du réseau;*
- *T est l'ensemble des transitions du réseau, tel que* $P \cap T = \emptyset$*;*
- $W: (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ *est une fonction d'étiquetage d'arcs (appelée aussi relation de flot). Elle indique le nombre de jetons consommés ou générés par le tir d'une transition.*

Nous dirons qu'un réseau est fini si l'ensemble $P \cup T$ l'est.

Lorsque W est définie dans $\{0,1\}$, nous dirons que le réseau est à arcs simples. $W(p,t) = k$ signifie que la transition *t* utilise *k* jetons de la place p ; de façon duale, $W(t,p) = k$ signifie que la transition *t* crée *k* jetons dans la place *p*.

Définition A.2 (Marquage)

Un réseau de Petri marqué est défini par le couple $\langle N, M \rangle$ *dans lequel N est un réseau de Petri et* $M: P \rightarrow \mathbb{N}$ *une application appelée marquage.*

Le symbole $M(N)$, plus simplement M quand le réseau est connu, définit le marquage du réseau de Petri. $M(p)$ indique le marquage de la place *p*, c'est-à-dire le nombre (entier) de jetons contenus dans la place *p*. Le marquage initial noté M_0, donne le nombre initial de jetons présents dans chacune des places du réseau et précise donc l'état global initial du système considéré. Le marquage peut être défini également comme étant un multi-ensemble sur l'ensemble des places *P* du réseau.

Un RdP peut être représenté de deux manières : Sous forme graphique ou sous forme matricielle.

A.1. Représentation graphique

Les places d'un RdP sont représentées graphiquement par des cercles, les transitions par des rectangles ou des traits. Un arc relie une place *p* à une transition *t* si et seulement si $W(p,t) \neq 0$. Un arc relie une transition *t* à une place *p* si et seulement si $W(t,p) \neq 0$.

A.2. Représentation matricielle

Il est possible de représenter la fonction d'étiquetage *W* par des matrices. On appelle matrice pré-condition *pré* (respectivement matrice post-condition *post*) la matrice de dimension *(m, n)* à coefficients dans $\mathbb{N}$ où *m* est le nombre de places et *n* est le nombre de transitions telle que $pré(p_i, t_j) = W(p_i, t_j)$ (respectivement $post(t_j, p_i) = W(t_j, p_i)$). L'élément $pré(i,j)$ indique le nombre de marques que doit contenir la place p_i pour que la transition t_j soit franchissable. De façon duale, l'élément $post(i,j)$ indique le nombre de marques déposées dans la place p_i suite au franchissement de la transition t_j. La matrice $C = post - pré$ est dite matrice *d'incidence*. Un marquage *M* sera représenté par un vecteur de dimension *m* à coefficients dans $\mathbb{N}$.

Exemple A.1

Soit le réseau de Petri marqué $N = (P, t, W, M_0)$ tel que $P = \{p_1, p_2, p_3\}$ et $T = \{t_1, t_2, t_3, t_4\}$. On définit W et M_0 comme suit :

$$W(p_1, t_2) = 1\ ; W(p_2, t_1) = 1; W(p_2, t_3) = 3; W(p_3, t_4) = 1;$$

$$W(t_1, p_1) = 1; W(t_2, p_2) = 1; W(t_3, p_3) = 1; W(t_4, p_2) = 3;$$

$$M_0(p_1) = 0;\ M_0(p_2) = 3;\ M_0(p_3) = 0.$$

Il est facile de voir le confort apporté par la représentation graphique du réseau illustré par la figure A.1.

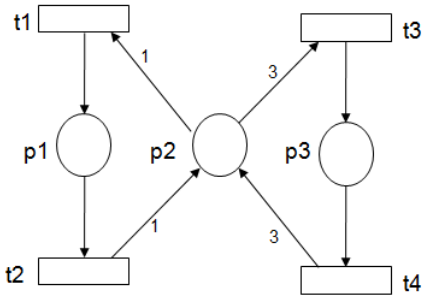

Fig.A.1- Réseau de Petri

La représentation matricielle associée est donnée comme suit :

$$pré = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}, \quad post = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array},$$

Le marquage initial est donné par $M_0 = \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix} \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}$. Il peut être donné également sous la forme d'un multi-ensemble $M_0 = 3p_2$.

A.3. Sémantique d'un réseau

L'un des intérêts majeurs des modèles formels est la possibilité de définir sans ambiguïté le comportement d'un système, de développer des algorithmes de vérification de propriétés puis de les intégrer dans un outil logiciel.

Définition A.3 (Fonctionnement)

Une transition $t \epsilon T$ est sensibilisée par un marquage M du réseau de Petri si et seulement si $\forall p \epsilon P \ M(p) \geq W(p,t)$. Son tir conduit alors à un nouveau marquage M' calculé par la règle suivante : $\forall p \epsilon P \ M'(p) = M(p) - W(p,t) + W(t,p)$ ou par $M' = M - pre(t) + post(t)$.

On note M $[t >$ *M' ou encore M* $\xrightarrow{t}$ *M' le franchissement de t à partir du marquage M. S'il existe une suite* $M_0[t_1 > M_1[t_2 > \cdots [t_n > M_n$ *alors la séquence* $\sigma = t_1 \ldots t_2$ *est dite séquence de franchissement.*

La règle de franchissement établie dans la définition précédente permet d'associer un *graphe d'accessibilité* (fini ou infini) au réseau. Ce graphe constitue une représentation formelle du comportement du réseau. Le graphe d'accessibilité est constitué de l'ensemble des marquages accessibles depuis le marquage initial.

Définition A.4 (Ensemble d'accessibilité)

Soit (N, M_0) un réseau de Petri marqué, l'ensemble d'accessibilité de ce réseau, noté $A(N, M_0)$, est l'ensemble des marquages qui peuvent être atteints par une séquence de franchissement :

$$A(N, M_0) = \{M / \exists \sigma \in T^* \ tel \ que \ M_0[\sigma > M\}$$

Une manière plus complète consiste à prendre en compte la relation d'accessibilité immédiate entre marquages accessibles à travers le *graphe d'accessibilité*.

Définition A.5 (Graphe d'accessibilité)

Soit (N, M_0) un réseau de Petri marqué, le graphe d'accessibilité d'un réseau noté $G(N, M_0)$ est défini par

- *L'ensemble des nœuds $A(N, M_0)$;*
- *L'ensemble des arcs : un arc étiqueté t joint M à M' si et seulement si $M[t > M'$.*

A.4. Propriétés

L'intérêt d'un modèle est la possibilité de définir des propriétés du système modélisé de manière formelle et de vérifier ces propriétés à l'aide d'algorithmes ou d'heuristiques. L'apport majeur des réseaux de Petri réside dans leur support solide pour l'analyse de plusieurs propriétés et problèmes associés aux systèmes concurrents. Deux classes de propriétés peuvent être étudiées avec les modèles à base de réseaux de Petri : Les propriétés qui dépendent du marquage initial et celles qui sont indépendantes du marquage initial. Les propriétés appartenant à la première classe sont dites *propriétés comportementales*, celles appartenant à la deuxième classe sont dites *propriétés structurelles*. Dans la section courante nous nous limitons à la présentation de quelques propriétés comportementales de base [47].

A.4.1. L'accessibilité

L'accessibilité constitue une base fondamentale pour l'étude des propriétés dynamiques d'un système. Un marquage M_n est dit accessible à partir d'un marquage M_0 s'il existe une séquence de franchissement qui transforme le marquage M_0 en M_n. Le problème d'accessibilité pour un marquage donné $M_n$est celui de trouver si $M_n \epsilon A(N, M_0)$.

A.4.2. La bornitude

Un réseau de Petri (N, M_0) est dit *k-borné* ou simplement *borné* si le nombre de jetons dans chacune des places du réseau n'excède pas le nombre fini k pour n'importe quel marquage accessible à partir de M_0. Autrement dit $\forall M \epsilon\, A(N, M_0), \forall p \epsilon P : M(p) \leq k$. Un réseau est dit *sain* s'il est *1-borné*. Le problème de bornitude consiste à décider pour un réseau (N, M_0) si $|A(N, M_0)|$ est fini ou pas.

A.4.3. La vivacité

Le concept de vivacité est lié à l'absence complète de blocage dans un système. Un réseau de Petri (N, M_0) est dit vivant si, indépendamment du marquage atteint à partir de M_0, il est possible de franchir n'importe quelle transition du réseau après le tir d'une séquence de franchissement donnée. Etant donné un réseau de Petri (N, M_0), le problème de vivacité revient à décider si pour chaque transition *t* , et pour chaque marquage $M \epsilon A(N, M_0)$, il existe

une séquence de transitions σ telle que $M \xrightarrow{\sigma.t}$, c'est-à-dire que *t* est sensibilisée après le franchissement de σ à partir de M.

A.4.4. La couverture

Etant donné un réseau de Petri (N, M_0) et un marquage M, le problème de couverture consiste à décider s'il existe un marquage $M' \epsilon A(N, M_0)$ tel que $M'(p) \leq M(p)$ pour chaque place *p* du réseau. La couverture possède de nombreuses applications, elle permet de vérifier les propriétés de sûreté usuelles. Par exemple, si on veut tester si deux places sont en exclusion mutuelle, il suffit de chercher à couvrir le marquage composé d'un jeton dans chaque place.

Un problème de décision est dit *décidable* s'il est récursif [48], c'est-à-dire qu'on arrive à exhiber un algorithme permettant de le résoudre automatiquement. Les problèmes décrits ci-dessus sont tous décidables dans le modèle des réseaux de Petri ordinaires décrit précédemment. L'étude des résultats de la décidabilité et de la complexité dans les réseaux de Petri sort du cadre de notre travail. Le lecteur intéressé trouvera dans [48, 49, 50] un panorama détaillé sur cet aspect.

Annexe B

Le paradigme nets within nets

La technologie de l'orienté objet a permis de définir des modèles qui se rapprochent des systèmes réels, constitués généralement d'objets discrets en interaction qui incorporent les structures de données et les comportements à la fois, et qui peuvent changer d'état de manière individuelle. Les modèles résultants sont essentiellement caractérisés par leur simplicité et leur facilité d'interprétation. En effet, le style de modélisation orienté objet permet entre autres, de construire une conception claire organisée au tour d'abstractions d'objets indépendamment du langage d'implémentation. Il permet par ailleurs, une meilleure compréhension et maintenance des systèmes.

Dans les modèles de Réseaux de Petri colorés, la notion d'objet se restreint à des structures de données passives (telles que les variables et les enregistrements) modélisées par différents types de jetons et privées de tout comportement individuel. Afin de prendre en charge ce type de comportement des objets, une approche naturelle consiste à décrire ces jetons par des réseaux dont le marquage peut évoluer individuellement. Cette idée a été formalisée par le paradigme *nets within nets* introduit par Valk [34, 35]. Dans ce paradigme, on modélise les objets d'un système par des réseaux qu'on appelle *réseaux objets.* A partir de ces réseaux sont instanciés les *jetons réseau* qui font partie du marquage du réseau qui modélise le système sous-jacent appelé *réseau système*. Ces modèles sont dits *systèmes à objets.* Plusieurs niveaux hiérarchiques peuvent être envisagés, mais lorsque l'on se limite à un seul niveau d'imbrication, on parle de *systèmes à objets élémentaires EOS.*

Valk a introduit dans les EOS deux sémantiques différentes [36] : Une sémantique par référence et une sémantique par valeur. Elles correspondent aux techniques d'accès aux objets par référence et par valeur connues dans les langages de programmation similaires à Java. Dans la sémantique par référence, les jetons réseaux d'un même réseau objet consistent en une référence à celui-ci, alors que dans la sémantique par valeur, les différentes instances d'un réseau objet sont considérées comme étant des copies indépendantes. Celles-ci sont alors

identiques dans leur structure mais leurs marquages peuvent être différents. Dans ce qui suit, nous présentons les EOS ainsi que les deux sémantiques par référence et par valeur.

B.1. Les systèmes à objets élémentaires EOS

Le modèle des systèmes à objets élémentaires est une forme simplifiée des nets within nets structurée en deux niveaux et composée de :

- Un *réseau système* qui reflète la structure générale du système modélisé, c'est un réseau de Petri P/T défini par $\widehat{N} = (\hat{P}, \hat{T}, W)$.
- Un ensemble fini de *réseaux objets* $\mathcal{N} = \{N_1, \dots, N_n\}$. Chaque réseau objet $N \in \mathcal{N}$ modélise la structure d'un objet différent du système et est donné par $N = (P_N, T_N, W_N)$.

On impose que tous les ensembles des nœuds (places et transitions) sont disjoints. Les places du réseau système sont typées par la fonction $d: \hat{P} \to \{\bullet\} \cup \mathcal{N}$. Une place $\hat{p}$ peut contenir des jetons noirs classiques seulement ($d(\hat{p}) = \bullet$) ou des jetons réseaux de type N seulement ($d(\hat{p}) = N$). L'ensemble des places de $\widehat{P}$ du type N sont données par $d^{-1}(N)$.

Le typage *d* est dit monotone si $\forall \hat{t} \in \hat{T}: \forall N \in \mathcal{N} : N \in d(\bullet \hat{t}) \Rightarrow N \in d(\hat{t}\bullet)$.

Exemple B.1

Le réseau présenté dans la figure B.1 illustre un EOS composé de :

- Un réseau système $\widehat{N} = (\hat{P}, \hat{T}, W)$ avec $\hat{P} = \{p_1, \dots, p_9\}$ et $\hat{T} = \{t\}$.
- Deux réseaux objets $\mathcal{N}=\{N_1, N_2\}$tels que
  - $N_1=(P_1, T_1, W_1)$ *avec* $P_1=\{a_1, b_1\}$ *et* $T_1=\{t_1\}$.
  - $N_2=(P_2, T_2, W_2)$ *avec* $P_2=\{a_2, b_2, c_2\}$ *et* $T_2=\{t_2\}$.

- Le typage des places est tel que :
  - $d(p_1)=d(p_2)=d(p_6)=N_1$,
  - $d(p_3)= d(p_7)= d(p_8)= N_2$,
  - $d(p_4)= d(p_5)= d(p_9)= \bullet$.

Les places de même type sont remplies par la même couleur.

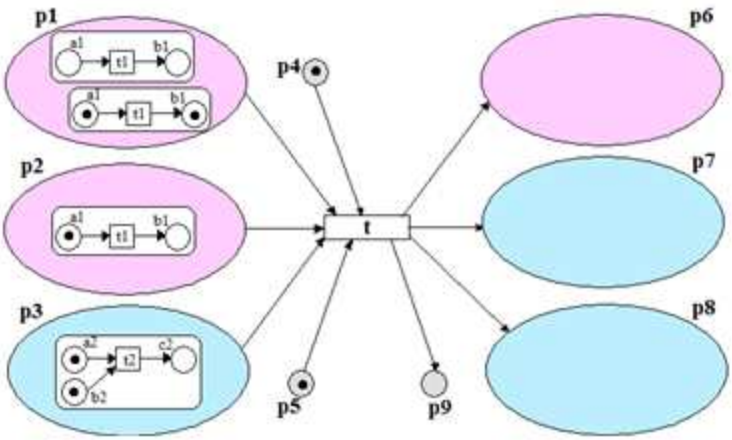

Fig.B.1- Réseau de Petri à objets

B.1.1. Les évènements dans les EOS

Dans les systèmes EOS, les jetons réseaux peuvent effectuer des activités internes (indépendantes du réseau système) conduisant au changement de leurs états. Ces activités correspondent à des évènements dits *autonomes objets*. Lorsque le changement d'état des jetons réseaux est provoqué par le réseau système, alors l'évènement déclenchant est dit *évènement synchronisé* ou *interaction*. Cet évènement est produit par la synchronisation d'une transition d'un réseau objet avec une transition du réseau système via des canaux de synchronisation. Un évènement système n'ayant aucun effet sur les réseaux objets est dit *évènement autonome système*.

Pour avoir une représentation uniformisée des trois types d'évènements qui peuvent survenir dans le système, les transitions suivantes ont été définies [38] :

- Une transition silencieuse ε_N pour chaque $N \in \mathcal{N}$ telle que $pre(\varepsilon_N) = post(\varepsilon_N) = 0$
- Un ensemble de transitions silencieuses $\{\varepsilon_{\hat{p}} \mid \hat{p} \in \hat{P}\}$ tel que $pre(\varepsilon_{\hat{p}}) = post(\varepsilon_{\hat{p}}) = \hat{p}$ pour le réseau système.

$\mathcal{C}$ est un ensemble de fonctions défini comme suit :

$$\mathcal{C} : \mathcal{N} \rightarrow \cup_{N \in \mathcal{N}} (T_N \cup \{\varepsilon_N\}).$$

Chaque $C \in \mathcal{C}$ est une fonction qui associe à chaque réseau objet une de ses propres transitions (la transition silencieuse comprise). On note alors $C(N) \in T_N \cup \varepsilon_N$. Particulièrement, la fonction C qui associe à un réseau objet N sa transition silencieuse est

notée ε_C telle que $\varepsilon_C(N) = \varepsilon_N$. Pour $\mathcal{N} = \{N_1, \dots, N_n\}$ on écrit $C \in \mathcal{C}$ sous la forme d'un tuple $(C(N_1), \dots, C(N_n))$.

Ainsi, un évènement du système, en tenant abstraction de sa nature, est donné par un couple $(\hat{\tau}, C)$ tel que $\hat{\tau}$ est une transition du réseau système (les transitions silencieuses comprises) et $C \in \mathcal{C}$. L'ensemble de tous les évènements possibles qu'on note $\boldsymbol{\tau}$ est défini comme suit :

$$\boldsymbol{\tau} = \{(\hat{\tau}, C) \mid \hat{\tau} \in \hat{T} \cup \{\varepsilon_{\hat{p}} | \hat{p} \epsilon \hat{P}\} \wedge\ C \in \mathcal{C}\} \setminus \{(\varepsilon_{\hat{p}}, \varepsilon_C) \mid \hat{p} \epsilon \hat{P}\}$$

L'évènement $(\hat{\tau}, C)$ signifie que la transition $\hat{\tau}$ est tirée simultanément avec toutes les transitions *C(N) ($N \in \mathcal{N}$).* Autrement dit, une transition du système réseau a exactement un partenaire de synchronisation (qui peut être une transition silencieuse) dans chaque réseau objet $N \in \mathcal{N}$.

Les évènements autonomes système sont de la forme$(\hat{t}, \varepsilon_C)$. Pour un évènement autonome objet se produisant au niveau du réseau objet N_i dans la place $\hat{p}$ du réseau système, nous avons d'une part $\hat{\tau} = \varepsilon_{\hat{p}}$ et d'autre part, pour tout j différent de i $C(N_j) = \varepsilon_c$.

On définit l'ensemble θ appelé *structure de synchronisation* tel que $\theta \subseteq \boldsymbol{\tau}$**.** Cet ensemble regroupe les évènements synchronisés possibles du système. La synchronisation est réalisée via des canaux dirigés. Les canaux sont représentés sous forme d'annotations de transitions de la forme $ON: nom()$ pour les transitions du réseau système et de la forme $: nom()$ pour les transitions des réseaux objets. L'asymétrie est due au fait que les réseaux objets (ON) sont supposés connus par le réseau système, le contraire n'est pas vrai. Les transitions annotées par le même canal sont obligatoirement tirées simultanément. Notons que les transitions silencieuses ne sont pas représentées graphiquement.

Exemple B.2

Dans le réseau illustré dans la figure B.1**,** $\boldsymbol{\tau}$ n'est constitué que d'évènements autonomes : un événement autonome système et deux évènements autonomes objet :

$$\boldsymbol{\tau} = \{\left(\boldsymbol{t}, \left(\boldsymbol{\varepsilon_{N_1}}, \boldsymbol{\varepsilon_{N_2}}\right)\right)\} \cup \left\{\left(\boldsymbol{\varepsilon_{\hat{p}}}, \left(\boldsymbol{t_1}, \boldsymbol{\varepsilon_{N_2}}\right)\right), \left(\boldsymbol{\varepsilon_{\hat{p}}}, \left(\boldsymbol{\varepsilon_{N_1}}, \boldsymbol{t_2}\right)\right) \mid \boldsymbol{\hat{p}} \in \boldsymbol{\hat{P}}\right\}$$

Définition B.1 (EOS) [38]

Un système à objets élémentaires EOS est un tuple OS =($\widehat{N}$, $\mathcal{N}$, d, θ) tel que :

1. *$\widehat{N}$est un réseau de Petri Place / Transition appelé réseau système.*

2. *$\mathcal{N}$ est un ensemble fini de réseaux de Petri Place/Transition, appelés réseaux objets.*
3. *$d : \hat{P} \rightarrow \{\bullet\} \cup \mathcal{N}$ est un typage monotone des places du réseau système.*
4. *$\mathcal{t} \subseteq \boldsymbol{\tau}$ est une structure de synchronisation finie.*

On cite quelques propriétés d'un EOS :

- Un EOS est dit *minimal* si et seulement s'il possède un seul type de réseaux objets : $|\mathcal{N}|=1$.
- Un EOS est dit *pur* s'il ne possède aucune place ayant le jeton noir ordinaire comme type : $d^{-1}(\bullet) = \phi$.
- Un EOS est dit *unaire* si et seulement s'il est pur et minimal.

B.1.2. Les sémantique des EOS

Les modèles *nets within nets* peuvent être étudiés suivant deux sémantiques différentes : une sémantique par référence et une sémantique par valeur. Elles correspondent respectivement à l'appel par référence et par valeur dans les langages de programmation. Dans la sémantique par référence, des identifiants sont utilisés comme références pour le même jeton réseau. Une même référence peut être utilisée comme jeton pour plus d'une place. Dans la sémantique par valeur, chaque jeton réseau est modélisé comme étant un objet différent et se trouve dans une place spécifique. La différence principale entre les deux sémantiques réside dans leur gestion de la distribution.

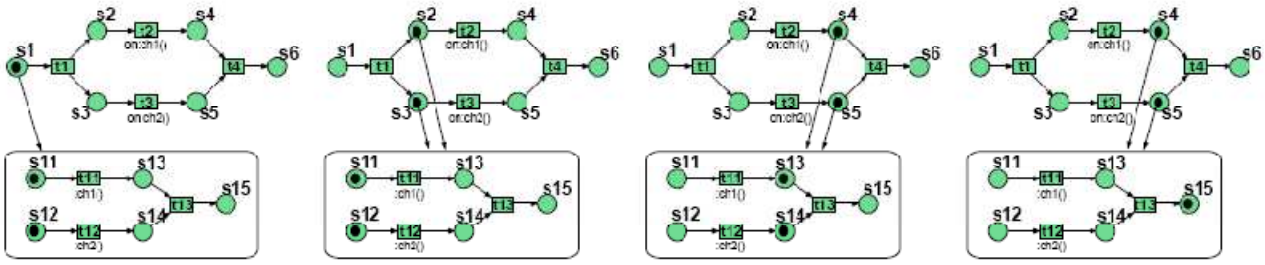

Fig.B.2- Séquence de tir par rapport à la sémantique par référence

Dans la figure B.2, une séquence de tir par rapport à la sémantique par référence est illustrée. D'abord, le tir de la transition t_1 (évènement autonome système) crée deux références sur les places s_2 et s_3. Le tir concurrent des deux évènements synchronisés (t_2, t_{11})

et (t_3, t_{12}) conduit au niveau du réseau système à la suppression des références existantes sur les places s_2 et s_3 et génère deux références s_4 et s_5, et au niveau du réseau objet à la consommation du marquage $s_{11} + s_{12}$ et à la génération du marquage $s_{13} + s_{14}$. La transition t_{13} , dont le tir correspond à un évènement autonome objet, devient alors sensibilisée. A son tir, elle consomme le marquage $s_{13} + s_{14}$ et produit le marquage s_{15}.

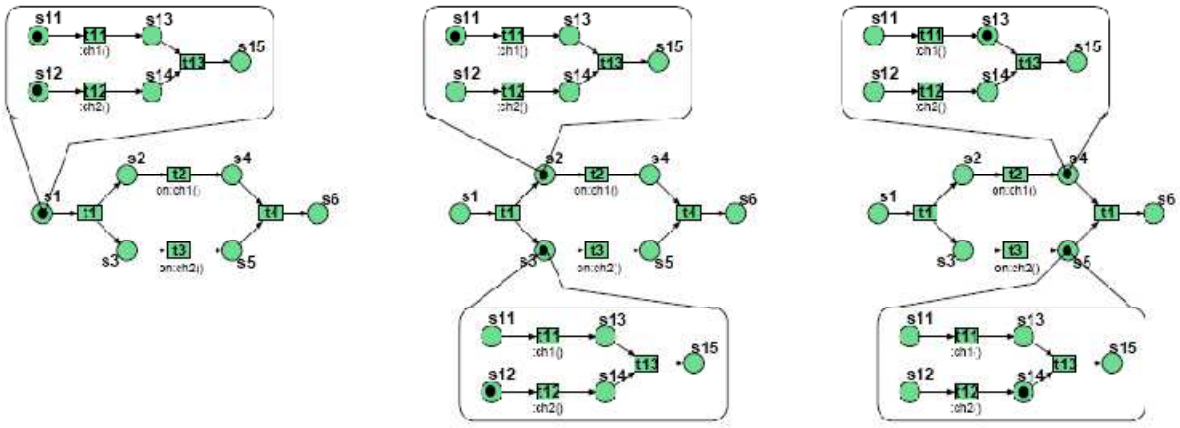

Fig.B.3- Séquence de tir par rapport à la sémantique par valeur

La figure B.3 illustre une séquence de tir suivant la sémantique par valeur. Le tir de la transition t_1 crée deux jetons réseaux qui sont des copies du jeton réseau original situé dans la place s_1. Le marquage du jeton réseau original consommé est distribué sur les deux copies nouvellement crées. La distribution choisie dans cette séquence rend les deux évènements (t_2, t_{11}) et (t_3, t_{12}) franchissables. Le tir de (t_2, t_{11}) modifie le marquage du jeton réseau de la place s_2 tandis que le tir de (t_3, t_{12}) modifie celui du jeton réseau de la place s_3. Du moment que les jetons des places s_{13} et s_{14} sont distribués dans deux copies différentes, la transition t_{13} ne peut être tirée qu'après la recombinaison des deux copies par le tir de la transition t_4.

En analysant les deux séquences de tir, il est facile de constater que la sémantique par valeur est plus intuitive lorsqu'il s'agit de la modélisation des objets mobiles dans les systèmes distribués du moment que chaque place p du réseau système constitue une localisation pour les jetons réseaux indépendante des autres localisations.

B.1.2.1. La sémantique par référence

Dans la sémantique par référence, les jetons réseaux sont interprétés comme des références, au même titre que les pointeurs dans les langages de programmation.

Définition B.2 (Marquage)

Un marquage par rapport à la sémantique par référence est un multi-ensemble $\boldsymbol{M} \epsilon M_r$ *tel que* $M_r = MS(\hat{P} \cup \cup_{N \epsilon \mathcal{N}} P_N)$*. Le marquage initial d'un EOS par rapport à la sémantique par référence est donné par* $\boldsymbol{M_0} = \widehat{M_0} + \sum_{\boldsymbol{N \epsilon \mathcal{N}}} M_{N_0}$ *tel que* $\widehat{M_0}$ *est le marquage initial du réseau système et* M_{N_0} *est le marquage initial du réseau objet N.*

Définition B.3 (Fonctionnement)

Soit OS =($\widehat{N}$*,* $\mathcal{N}$*, d,* θ*) un système à objets élémentaires et* $\boldsymbol{M} \epsilon M_r$ *le marquage de OS suivant la sémantique par référence tel que* $\boldsymbol{M} = \widehat{M} + \sum_{\boldsymbol{N \epsilon \mathcal{N}}} M_N$*. Un évènement* $(\hat{\tau}, C)$ *est sensibilisé par le marquage* $\boldsymbol{M}$ *si et seulement si :*

- $\widehat{M} \geq pre(\hat{\tau})$ *et*
- $\forall N \in \mathcal{N}: M_N \geq pre\big(C(N)\big)$ *;*

Le marquage successeur est donné par $\boldsymbol{M'} = \widehat{M}' + \sum_{\boldsymbol{N \epsilon \mathcal{N}}} M_N{}'$ *tel que :*

- $\widehat{M}' = \widehat{M} - pre(\hat{\tau}) + post(\hat{\tau})$ *et*
- $\forall N \in \mathcal{N}: M'_N = M_N - pre\big(C(N)\big) + post\big(C(N)\big)$ *.*

B.1.2.2. La sémantique par valeur

Dans la sémantique par valeur, chaque jeton réseau est une instance d'un réseau objet. Plusieurs jetons réseaux indépendants peuvent être dérivés d'un même réseau objet et peuvent avoir des marquages différents.

Définition B.4 (Marquage)

Un marquage par rapport à la sémantique par valeur est un multi-ensemble $\boldsymbol{M} \epsilon M_v$ *tel que :*

$$M_v = (\mathrm{d}^{-1}(\bullet) \times \{0\}) \cup \cup_{N \epsilon \mathcal{N}} (\ d^{-1}(N) \times MS(P_N))$$

Le marquage initial d'un EOS par rapport à la sémantique par valeur est donné par $\boldsymbol{M_0} = \sum_{k=1}^{|\boldsymbol{M_0}|} (\widehat{p_k}, M_{k_0})$ *tel que* $\widehat{p_k}$ *est une place du réseau système et* M_{k_0} *est le marquage initial du jeton réseau du type* $d(\widehat{p_k})$*. Il est égal au multi-ensemble vide dans le cas où* $d(\widehat{p_k}) = \{\bullet\}$*.*

Les marquages du système peuvent être comparés en utilisant des projections sur le premier ou le deuxième composant.

La projection $\Pi^1(M)$ sur le premier niveau (composant) fait abstraction de la sous structure des jetons réseaux. Elle est donnée par :

$$\Pi^1\left(\sum_{k=1}^{|\boldsymbol{M}|}(\widehat{p_k}, M_k)\right) = \sum_{k=1}^{|\boldsymbol{M}|}\widehat{p_k}$$

La projection $\Pi^2{}_N(M)$ sur le second composant est le marquage abstrait de tous les jetons réseau du type N$\in \mathcal{N}$ sans considérer leur distribution locale dans le réseau système. Elle est donnée par :

$$\Pi^2{}_N\left(\sum_{k=1}^{|\boldsymbol{M}|}(\widehat{p_k}, M_k)\right) = \sum_{k=1}^{|\boldsymbol{M}|}\pi_N(\widehat{p_k}).\ M_k$$

$\pi_N : \hat{P} \to \{0,1\}$ est une fonction indicatrice telle que $\pi_N(\hat{p}) = 1\ ssi\ d(\hat{p}) = N$.

Le tir autonome d'une transition *t* du réseau système consomme les jetons de la pré condition avec leurs marquages individuels internes. Comme les marquages sont des multi ensembles imbriqués, on doit considérer les termes $PRE\ et\ POST$ appartenant à M_v. Les projections des multi ensembles PRE et POST sur le premier composant correspondent respectivement à la pré condition et la post condition de *t* :

$$\Pi^1(PRE) = pre(t)\ et\ \Pi^1(POST) = post(t)$$

Le marquage successeur est donc $M' = M - PRE + POST$ par analogie avec le marquage $M' = M - pre(t) + post(t)$ dans un réseau de Petri P/T ordinaire. Le tir de la transition *t* doit par ailleurs obéir à la *condition de distribution du marquage objet* :

$$\Pi^2{}_N(PRE) = \Pi^2{}_N(POST)$$

Cette condition assure que la somme des marquages dans les copies d'un jeton réseau est préservée.

Définition B.5 (Fonctionnement)

Soit OS =($\widehat{N}$, $\mathcal{N}$, d, θ) un système à objets élémentaires et $\boldsymbol{M} \in M_v$ *le marquage de OS suivant la sémantique par valeur tel que* $\boldsymbol{M} = \sum_{k=1}^{|\boldsymbol{M}|}(\widehat{p_k}, M_k)$. *Pour un évènement* $\tau = (\hat{\tau}, C)$ *on définit :*

$$\boldsymbol{M} \xrightarrow{\tau} \boldsymbol{M'} \Leftrightarrow \exists\ PRE, POST \in M_v : \boldsymbol{M} \geq PRE \wedge$$
$$\varphi(\tau, PRE, POST) \wedge$$
$$M' = M - PRE + POST.$$

Le prédicat $\varphi(\tau, PRE, POST)$ est tel que :

$$\varphi(\tau, PRE, POST) \Rightarrow \prod^1(PRE) = pre(\hat{\tau}) \wedge \prod^1(POST) = post(\hat{\tau}) \wedge$$
$$\forall \mathrm{N} \in \mathcal{N} : \prod_N^2(PRE) \geq pre_N\big(C(N)\big) \wedge$$
$$\forall \mathrm{N} \in \mathcal{N} : \prod_N^2(POST) = \prod_N^2(PRE) - pre_N\big(C(N)\big) + post_N\big(C(N)\big).$$

Bibliographie

[01] C. CUBAT DITCROS. *Agent mobiles coopérant pour les environnements dynamiques*. Thèse de Doctorat à l'Institut National Polytechnique de Toulouse. Décembre 2005.

[02] M. LACOSTE. *Une machine virtuelle répartie pour la programmation de processus mobiles*. Thèse de Doctorat à l'université JOSEPH FOURIER. Octobre 2003.

[03] G.C. Roman, G.P. Picco, A. Murphy. *Software engineering for mobility: A road map*. Int. Conference on Software Engineering (ICSE)**,** 2000, p. 3-22.

[04] G. Bernard. *Applicabilité et performances des systèmes d'agents mobiles dans des systèmes répartis*. Première Conférence Française en Systèmes d'Exploitation (CFSE.1), Rennes, France, Juin 1999.

[05] N. Carriero, D. Gelernter. *Linda in context.* Communications of the Association for Computing Machinery (ACM), vol. 32, n° 4, April 1989, p. 444-458.

[06] G. Wells. *Coordination languages: Back to the future with Linda*. 2nd Int. Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'05), July 2005.

[07] A. Fuggetta, G.P. Picco, and G. Vigna. *Understanding Code Mobility*. IEEE Transactions on Software Engineering, May 1998, p. 342–361.

[08] G.P. Picco. *Understanding, Evaluating, Formalizing, and Exploiting Code Mobility*. PhD thesis, University of Trento, Italy, February 1998.

[09] L. Cardelli. *Abstractions for Mobile Computation*. Secure Internet Programming: Security Issues for Distributed and Mobile Objects, LNCS, vol. 1603, 1999, p. 51–94.

[10] G.C. Roman, G. Picco, L. Murphy. *Coordination and mobility*. Coordination of Internet Agents: Models, Technologies and Applications, 2001, p. 253-273.

[11] C.A.R Hoare. *Communicating Sequential Processes*. Prentice-Hall International, April 1985, 256 p.

[12] R. Milner. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science (LNCS). Springer Verlag, 1980.

[13] R. Milner, J. Parrow, and D. Walker. *A calculus of mobile processes*. Information and Computation, 1992, p. 1-40.

[14] R. Ameur-Boulifa. *Génération des modèles comportementaux des applications réparties*. Thèse pour obtenir le titre de docteur en sciences de l'université de Nice-Sophia Antipolis. Spécialité Informatique, Décembre 2004.

[15] A.D. Gordon. *Nominal Calculi for Security and Mobility*. DARPA Workshop on Foundations for Secure Mobile Code, California, USA, 1997.

[16] A.D. Gordon. *Notes on Nominal Calculi for Security and Mobility*. Foundations of Security Analysis and Design (FOSAD'00), vol. 2171, Springer Heidelberg, 2001, p. 262–330.

[17] R. Milner. *Polyadic π-Calculus : a Tutorial.* Laboratory for Foundation of Computer Science (LFCS) report published in F. L. Hamer, W. Brauer and H. Schwichtenberg, editors, *Logic and Algebra of Specification,* Springer-Verlag, 1993.

[18] D.Sangiorgi. *Expressing Mobility in Process Algebra: First order and high order paradigms*. LFCS report. May 1993.

[19] P.Sewell, Pawel T.Wojciechowski and Benjamin C. Pierce. *Location-independent communication for mobile agents : A two-level architecture*. LNCS, vol.1686, Springer, 1998, p. 1-31.

[20] R. Devillers, H.Klaudel and M. Koutny. *Petri Net Semantics of the finite π-calculus*. LNCS 3235, Springer, 2004, p. 309-325.

[21] G. Berry, G. Boudol. *The chemical Abstract Machine*. Theoritical Computer Science, 1992, p.217-248.

[22] C. Fournet, G. Gonthier. *The reflexive CHAM and the join-calculus*. In Proc. of the 23rd ACM Symposium on Principles of Programming Languages, 1996, p. 372-385.

[23] C.Fournet, G. Gonthier, J.J. Lévy, L. Maranget, and D. Rémy. *A calculus of mobile agents*. In Proc. of *CONCUR*, 1996, p. 406-421.

[24] L. Cardelli A.D. Gordon. *Mobile ambients*. In Foundations of Software Science And Computation Structures: First Int. Conference, FOSSACS '98, Springer-Verlag, 1998, p. 140-155.

[25] L. Bettini, V.Bono, R. De Nicola, G.L. Ferrari, D.Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, B. Venneri. *The KLAIM Project: Theory and Practice.* In Proc. of Global Computing: Programming Environments, Languages, Security and Analysis of Systems, LNCS 2874, Springer,2003, p. 88-150.

[26] D. Gorla, R. Pugliese. *Ressource Access and Mobility Control with Dynamic Privileges Acquisition*. In Proc. Of Int. Colloquium on Automata, Languages and Programming ICALP'03, LNCS vol 2719, Springer, 2003, p. 119-132.

[27] R. Devillers, D. Latella and M. Massink. *Formal modeling and quantitative analysis of Klaim based mobile systems.* In Proc. of Applied Computing, Association for Computing Machinery (ACM), 2005, p. 428- 435.

[28] R. Devillers, H.Klaudel and M. Koutny: *"A Petri Net Semantics of a Simple Process Algebra for Mobility*". Electronic Notes in Theoretical Computer Sciences (ENTCS), vol. 154, 2006, p. 71-94.

[29] A. Asperti, N. Busi. *Mobile Petri Nets*. Technical Report UBLCS-96-10, Department of Computer Science University of Bologna, May 1996.

[30] N. Busi, L. Padovani. *A Distributed Implementation of Mobile Nets as Mobile Agents.* Formal Methods for Open Object based Distributed Systems (FMOODS), LNCS, vol 3535, Springer, 2005, p. 259–274.

[31] J.C.M. Baeten. *A Brief History of Process Algebra.* Theoretical Computer Science, vol. 335, Issues 2-3, May 2005, p. 131-146.

[32] G.C. Roman, P.J. McCann. *An Introduction to Mobile UNITY*. IPPS/SPDP Workshops, vol. 1388, 1998, p. 871-880.

[33] Gian Pietro Picco, Gruia-Catalin Roman, and Peter J. McCann . *Expressing code mobility in mobile unity*. In Proc. of the 6th European Software Engineering Conference held jointly with the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'97), vol. 1301, 1997, p. 500-518.

[34] R. Valk. *Petri nets as token objects: An introduction to elementary object nets*. Int. Conference on Application and Theory of Petri Nets (ICATPN), LNCS, vol.1420, Springer, Heidelberg, 1998, p. 1-25.

[35] R. Valk. *On processes of object Petri net*". Technical report, FBI-HH-B-185/96, Fachbereich Informatik, Universität Hamburg, 1996.

[36] R. Valk. *Relating Different Semantics for Object Petri nets.* Formal proofs and examples. Technical Report FBI-HH-B-226, University of Hamburg.

[37] S. Christensen and N.D. Hasen. *Coloured Petri nets Extended with channels for synchronous Communication*. 15 th Int. Conference on the Application And Theory of Petri Nets, Zaragoza (Spain), LNCS, vol. 815, Springer- Verlag, June 1994.

[38] M.Khöler, B.Farwer. *Object Nets for Mobility*. ICATPN, LNCS, vol. 4546, Springer Verlag, 2007, p. 244- 262.

[39] M. Bednarczyk, L. Bernardinello, W. Pawłowski, L. Pomello. *Modelling Mobility with Petri Hypernets*. In Proc. of Workshop on Algebraic Development Technics WADT, vol. 3423, 2004, p.28-44.

[40] D. Frutos Escrig, O. Marroqu´ın Alonso and F. Rosa Velardo. *Ubiquitous Systems and Petri Nets*. Ubiquitous Web Systems and Intelligence, co-located with ICCSA 2005, LNCS, vol.3841, Springer- Verlag, 2005, p. 1156-1166.

[41] F. Rosa-Velardo, D. Frutos-Escrig, O. Marroqu´ın-Alonso. *Mobile Synchronizing Petri Nets: a choreographic approach for coordination in Ubiquitous Systems*. ENTCS, vol. 150(1), Elsevier, Amsterdam, 2006, p.103-126.

[42] F. Rosa-Velardo, D. Frutos-Escrig, O. Marroqu´ın-Alonso. *On the expressiveness of Mobile Synchronizing Petri Nets*. In: 3rd Int. Workshop on Security Issues in Concurrency, SecCo'05. ENTCS, vol. 180, Issue 1, 12 June 2007, p. 77-94.

[43] F. Rosa-Velardo, D. Frutos-Escrig, O. Marroqu´ın-Alonso. *Replicated Ubiquitous Nets*. Int. Conference onComputational Science and its Applications (ICCSA 2006), LNCS, vol. 3983, Springer, Heidelberg, 2006, p. 158-168.

[44] F. Rosa-Velardo, D. Frutos-Escrig. *Name Creation vs. Replication in Petri Net Systems*. ICATPN 2007, LNCS 4546, Springer-Verlag, 2007, p. 402–422.

[45] F. Rosa-Velardo. *Petri nets for the verification of Ubiquitous Systems with Transient Secure Association*. Ubiquitous Intellegence and Computing (UIC), 2007, p. 1148-1158.

[46] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[47] T. Murata . *Petri nets: Properties, analysis and Applications.* In: Proc. of the IEEE, vol. 77, n°. 4, April 1989, p. 541-580.

[48] S. Haddad. *Décidabilité et complexité de problèmes de réseaux de Petri*. In Les *réseaux de Petri* - Modèles fondamentaux, chapitre 4, p. 119-158.

[49] J. Esparza, M. Nielsen. *Decidability issues for petri nets-a survey*. Bulletin of the European Association for Theoretical Computer Sciences (EATCS), 1994, p. 245-262.

[50] J. Esparza. *Decidability and complexity of Petri net problems-an introduction*. In Lectures on Petri Nets I: Basic Models, Springer-Verlag, 1998, p. 374-428.

[51] K.Jensen. *An Introduction to the Theoritical Aspects of Coloured Petri Nets*. A Decade of Concurrency, LNCS, vol. 803, Springer-Verlag 1994, pp. 230-272.

[52] C. Mascolo, G.P. Picco, G.C. Roman. *CODEWEAVE: Exploring Fine-Grained Mobility of Code*. Automated Software Engineering, vol. 11, Number 3, juin 2004. p. 207-243.

[53] E. Best, W. Fraczak, R. P. Hopkins, H. Klaudel, E. Pelz. *M-nets: an algebra of high level Petri nets, with an application to the semantics of concurrent programming languages.* Acta Informatica, 35. Springer, 1998.

[54] W.M.P. van der Aalst. *Interval Timed Coloured Petri Nets and their Analysis*. Application and Theory of Petri Nets, LNCS, vol. 691, Springer-Verlag, 1993, p. 453-472.

[55] X. Nicollin, J. Sifakis. *An overview and synthesis on timed process algebras*. Proc. of the Real-Time: Theory in Practice, REX Workshop, LNCS, vol. 600, 1991, p. 526-548.

[56] F. Stajano, R.J. Anderson. *The Resurrecting Duckling.* Security Issues for Ad-hoc Wireless Networks. 7th Int. Workshop on Security Protocols, LNCS, vol. 1796, Springer, 1999, p. 172-194.

[57] F. Stajano. *The Resurrecting Duckling – what next?*. Security Protocols, 8th Int. Workshop Proceedings, LNCS, vol. 2133, Springer, 2000, p. 204 – 214.

[58] H.Boucheneb. *Checking untimed and timed linear properties of the Interval Timed Colored Petri Nets model*. Revista Computacion y Sistemas, 10(2), 2006, p. 107-134.

[59] G. Berthelot, H. Boucheneb. *Occurrence graphs for interval timed colored Petri Nets*. 15th Int. Conference on Application and Theory of Petri Nets, Zaragoza (Spain), LNCS, vol. 815, Springer-Verlag, June 1994, p. 79-98.

[60] G. Berthelot, H. Boucheneb. *Contraction of the ITCPN state space.* ENTCS vol.6, Issue 5, June 2002.

[61] X. He, L. Chang, S. Shatz, J. Li**.** *Applying a Nested Petri Net Modeling Paradigm to Coordination of Sensor Networks with Mobile Agents.* Int. Workshop on Petri Nets and Distributed Systems PNDS 2008, Xi'an, China, June 2008.

[62] R. Milner. *Theories for the global ubiquitous computing*. Foundations of Software Science and Computation Structures, LNCS, vol. 2987, 2004, Springer Berlin.