Contribution à la modélisation et à la vérification des systèmes multi agents

Borhen Marzougui

ÉCOLE NATIONALE DES SCIENCES DE L'INFORMATIQUE	CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

École Doctorale Informatique, Télécommunications et Électronique(Paris)

Centre d'Etude et De Recherche en Informatique et Communications

École Doctorale des Sciences et Technologies de l'Informatique, de la Communication, du Design et de l'Environnement(Tunis)

THÈSE DE DOCTORAT

présentée par : **Borhen MARZOUGUI**

soutenue le : **12 Juin 2014**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

et le grade de : **Docteur de l'Ecole Nationale des Sciences de l'informatique**

Discipline / Spécialité: **Informatique / Systèmes distribués**

Contribution à la Modélisation et à la Vérification des Systèmes Multi Agents

THÈSE DIRIGÉE PAR

M. BARKAOUI Kamel	*Professeur des Universités, CNAM (Cedric)*
M. BEN HADJ ALOUANE Nejib	*Professeur, ENIT (OASIS)*

RAPPORTEURS

M. YEDDES Mohammed Moez	*Professeur, INSAT*
M. MONSUEZ Bruno	*Professeur, ENSTA ParisTech (LIE)*

EXAMINATEURS

Mme. Isabelle Wattiau,	*Professeur des Universités, CNAM (Cedric)*
Mme. SI-SAID CHERFI Samira	*Professeur, CNAM (Cedric)*

Dédicaces

A mes parents, pour leur perpétuel soutien et leur amour incessant.

A ma grande mère pour ses supplications de réussite.

A ma femme, pour sa patience et ses encouragements.

A mes petites filles Hadiil, Asiil, Baylacen et Liyan pour leurs petit cœurs aimant.

A mes frères et mes sœurs pour leurs encouragements.

A toute ma famille.

A mes amis.

Dédicace spéciale à l'âme du Martyr (Chahid), mon Maître Prof. Hatem Bettaher,

Nulle dédicace ne pourrait exprimer mes sentiments et mon profond attachement.

Borhen MARZOUGUI

Contribution à la Modélisation et à la Vérification des Systèmes Multi Agents

le cnam

Résumé :

Les Réseaux de Petri (RdP) sont actuellement les approches les plus prometteuses pour modéliser et vérifier les systèmes complexes tels que les Systèmes Multi Agents (SMA). De nombreuses solutions ont été proposées pour remédier aux problèmes de communication, de coordination et d'interaction entre les Agents. Cependant, il n'existe aucune en mesure de traiter, à la fois les aspects structurels et comportementaux, du moins à notre connaissance. La thèse s'intéresse à la problématique de modélisation formelle et de vérification automatique et semi-automatique de propriétés pour les Systèmes Multi Agents. Plus précisément, l'objectif consiste à proposer un nouveau modèle formel original basé sur les réseaux de Petri, les Réseaux de Petri à Agents (RdPA), qui permettent d'exprimer de manière consistante et plus précise les systèmes Multi Agents. Il s'intéresse de plus à l'extension de ce modèle aux fins de modéliser la migration des Agents dans le cadre des systèmes à Agents mobiles. Cette classe de modèle permet de s'intéresser à la vérification formelle de propriétés classiques comme notamment la vivacité ou l'absence d'interblocage dans le cadre des Systèmes Multi-Agent.

Mots clés :

Réseaux de Petri (RdP) ; Systèmes Multi Agents (SMA) ; Interaction ; Approche formelle ; Réseau de Petri à Agent (*RdPA*).

Abstract:

Petri nets (PN) are currently the most promising approaches to model and to verify complex systems such as Multi Agent Systems (MAS). Several solutions have been proposed to solve the problems of communication, coordination and interaction among Agents. However, to best of our knowledge, none of this solution has able to handle both aspects: structural and behavioral. The thesis focuses on the problem of formal modeling and automatic and semi-automatic verification of properties in Multi Agent Systems. More specifically, the objective is to propose a new original formal model based on Petri nets, Agents Petri nets (APN), which express consistently more accurate a Multi Agent Systems. There is growing interest in the extension of this model for modeling the migration of Agents within the mobile Agent systems. This class of model allows focusing on the formal verification of classical properties such as alertness or absence of deadlock in the context of Multi Agent Systems.

Keywords:

Petri Nets (PN); Multi Agents System (MAS); Interaction; Formal approach; Agent Petri Nets (APN).

Remerciements

Le présent travail a été effectué à l'Ecole Nationale des Ingénieurs de Tunis au Laboratoire OASIS en cotutelle avec le Laboratoire CEDRIC du Conservatoire National des Arts et Métiers de Paris.

Je voudrai exprimer ma profonde reconnaissance pour mon directeur de thèse Monsieur Kamel BARKAOUI, professeur au CNAM de Paris pour la confiance qu'il m'a accordé en me déléguant ce travail. Je souhaite saluer ici la qualité de son encadrement, sa grande compétence, la richesse de ses connaissances, sa disponibilité, sa patience et ses qualités humaines avec lesquelles il a supervisé ce travail. De même, je profite l'occasion de le remercier infiniment pour l'accueil au sein de l'équipe de recherche VESPA

Mes sincères remerciements vont, au CNAM de Paris. Qu'il trouve ici le témoignage de ma profonde gratitude pour avoir ouvert devant moi les portes de ses Laboratoires de Recherche.

Je tiens à remercier vivement Monsieur Nejib BEN HADJ-ALOUANE, professeur à l'ENIT, pour sa contribution dans le co-encadrement de ce travail. Ses conseils d'ordre scientifique, méthodologique, technique et rédactionnel ont considérablement contribué à la qualité des travaux développés dans le cadre de cette thèse. Je lui suis très reconnaissant d'avoir toujours été à l'écoute de mes interrogations.

Je dois reconnaître que tous les remerciements, aussi grands soient-ils que je puisse adresser au Monsieur Mohammed Moez YEDDES, professeur à l'INSAT de Tunis et au Monsieur Bruno Monsuez, professeur au ENSTA ParisTech (LIE) ne sauraient être à la hauteur de l'honneur qu'ils m'ont réservé pour examiner ma thèse.

A tous mes collègues stagiaires et doctorants : **Ridha**, **Yamen**, **Amel, Nasiba, Maryam, Ding, Yufeng, et Gaiyun** je veux dire que je garderai ancrées dans mon cœur leur aimable compagnie, les recréations et l'atmosphère de détente qui nous réunissaient après les longues heures de travail.

Mes remerciements vont également vers les membres de l'unité de recherche OASIS, ENIT Tunisie, dont je suis membre. Je souhaite spécialement remercier le Professeur **Atitel BEN HADJ-ALOUANE** de m'avoir accueilli dans son unité et pour les excellentes conditions de travail qui a su faire régner tout au long de cette thèse.

Je tiens à remercier vivement le Professeur **Hassane ALLA** pour l'accueil et pour l'intégration dans un projet CMCU avec son équipe de recherche.

Table des Matières

Liste des Tableaux

Liste des Figures

Liste des Acronymes

AUML : Agent Unifyied Model Language

BDI : Beliefs Desires Intentions

CG : Conceptual Graph

CGPN : Conceptual Graph Petri Nets

CSV : Comma Separated Values

CSM : Comité de la Sécurité Maritime

CTL : Computation Tree Logic

EVP : Équivalent Vingt Pieds

FIPA : Foundation for Intelligent Physical Agents

GL : Génie logiciel

GTM : Gestion de Transport Maritime

IA : Intelligence Artificielle

IAD : Intelligence Artificielle Distribuée

IHM: Interaction Homme Machine

KQML: Knowledge Query Manipulation Language

LTL: Linear Time Logic

MASIF: Mobile Agent System Interoperability Facility

OMG : Object Management Group

POA : Programmation Orientée Agents

RCA : Représentation des Comportements d'Agents

RCPN : Recursive Colored Petri Nets

RdP : Réseau de Petri

RdPA : Réseau de Petri à Agents

RdPC : Réseau de Petri Colorés

RdP PT : Réseau de Petri Place/Transition

SDL : States Description Language

SGTM : Système de Gestion du Transport Maritime

SMA : Systèmes Multi Agents

SR : Système de Raisonnement

UML: Unified Modeling Language

VDM: Vienna Development Method

Introduction Générale

Les systèmes informatiques n'ont pas arrêté de s'accroitre ces dernières décennies. Leurs utilisations, implémentations et déploiements deviennent de plus en plus vastes et complexes. Il est devenu essentiel de gérer de plus en plus l'interactivité et la mobilité dans leurs utilisations les plus répandues. Ainsi, l'apparition des Systèmes Multi Agents, que nous désignons par la suite SMA, apporte une nouvelle dimension au concept de modélisation utilisé pour représenter une application du monde réel avec un degré approprié de complexité et de dynamisme. Cette discipline est à la connexion de plusieurs domaines dont les systèmes distribués, le génie logiciel et en particulier l'Intelligence Artificielle Distribuée (IAD).

A la différence de l'intelligence artificielle classique (IA) qui modélise le comportement intelligent d'un seul Agent, l'IAD s'intéresse à des comportements intelligents qui sont le produit de l'activité coopérative de plusieurs Agents.

Les recherches menées au début des années 70 afin de faire évoluer les domaines d'application de l'IA pour couvrir des domaines complexes, telle que l'aide à la décision, ont montré les limites de l'approche classique qui s'appuie sur la centralisation de l'expertise. Ces limites ont contribué à la naissance de l'IAD. Cette discipline a pour but la distribution de l'intelligence sur un groupe d'Agents. Ces derniers sont capables de travailler et d'agir dans un Environnement commun pour coopérer et résoudre les conflits éventuels. En effet, nous identifions trois thèmes de recherche fondamentaux dans l'IAD. Le premier thème concerne la résolution distribuée des problèmes : il étudie la manière de diviser un problème sur un ensemble

d'entités distribuées et coopérantes. Le deuxième thème est celui de l'intelligence artificielle parallèle : il vise l'amélioration des performances des systèmes de l'intelligence artificielle sans, toutefois, s'intéresser à la nature du raisonnement ou au comportement des Agents. Le dernier thème décrit les Systèmes Multi Agents : il s'agit de faire coopérer un ensemble d'Agents dotés d'un comportement intelligent et coordonner leurs buts et leurs plans d'actions pour la résolution d'un problème. C'est ce dernier thème auquel nous nous intéressons dans cette contribution. Nous cherchons à résoudre plusieurs problématiques telles que la modélisation de la connaissance et le problème de sa répartition sur les différents Agents, la gestion des conflits, le maintien de la cohérence des décisions et des plans d'actions et le problème de la communication et l'interaction entre les Agents. Nous montrons que la modélisation et la vérification du comportement des Agents, leurs mobilités, la coordination, la coopération et l'interaction présentent un défi à relever par le biais d'un outil formel.

Nous envisageons que la modélisation de ces systèmes dynamiques nécessite des outils de modélisations avancés qui respectent les différents critères de tels systèmes. Notre réflexion fait parvenir les méthodes formelles comme étant les outils les plus adéquats vérifiant les exigences de modélisation. Il est utile de souligner que les méthodes formelles ont été utilisées pour assurer un niveau de précision, de cohérence et d'exactitude assez élevé. Elles sont basées sur des fondements mathématiques afin de diminuer les risques d'incertitude et d'ambiguïté. En phase de conception des logiciels, les méthodes formelles permettent à un langage particulier, d'exprimer très rigoureusement les propriétés issues du problème de base. Elles peuvent être vues comme des méthodes intéressantes, puissantes et complémentaires entre elles ou entre d'autres méthodes classiques comme le test et le débogage.

Les résultats très précis, générés en utilisant un formalisme mathématique, restent tout au long de la phase conception le seul moyen de vérification sans ambiguïté. En effet, quel que soit l'étape du cycle de développement, nous pouvons déduire que le système étudié est valide ou non puisque la méthode utilisée fournit des moyens de démonstration le valorisant. D'un point de vu implémentation, ces méthodes démontrent aussi que le système a été correctement implémenté sans nécessairement le faire tourner. Pour les systèmes complexes tels que les systèmes distribués et les

Systèmes Multi Agents, l'analyse et la vérification sont beaucoup plus difficiles puisqu'elles peuvent produire des comportements non prédictibles.

A l'aide des méthodes formelles, nous pouvons révéler des ambiguïtés, des cas incompréhensibles et de l'incertitude. En effet, des failles de conception peuvent être découvertes dans un temps très réduit par rapport aux autres méthodes classiques qualifiées qui utilisent des étapes coûteuses. Chaque méthode hérite d'une autre un certain nombre de propriétés qui seront toutefois réutilisées ou bien raffinées. Certes, une grande complémentarité se présente en passant d'une méthode à une autre en se basant sur un modèle appelé modèle de base. Cependant, le passage à la concrétisation n'est envisageable que si la spécification soit bien validée par rapport aux fonctionnalités attendues du système. D'où, la nécessité de la mise en place d'outils, et de formalismes pour accomplir la vérification et la validation du modèle abouti.

Un enjeu important dans le domaine des *SMA* est de combiner les techniques de vérification formelles déjà existantes et de proposer de nouvelles méthodes qui permettent d'étudier les propriétés du bon fonctionnement du système dans le but de s'assurer de la conformité et de la cohérence de ces modèles et la satisfaction de l'utilisateur.

Etant donné que n'importe quel système est exposé aux erreurs vulnérables de conception, et que la moindre défaillance d'un logiciel peut entraîner des problèmes de perte de résultats, voir même la destruction du système et la question concernant de l'amélioration de notre confiance vis-à-vis les modèles conçus. Dans le cadre de systèmes complexes et dynamiques, l'amélioration de la pertinence de ces modèles vis-à-vis des exigences des systèmes concrets, exige des réflexions scientifiques poussées. La réponse attendue est de proposer des techniques fiables assurant la vérification des systèmes pour se protéger contre les bugs énormes qui peuvent être causés par les erreurs de conception qui se détectent tardivement. Ainsi, cette défaillance peut avoir des conséquences dramatiques sur la qualité, et les délais de livraison.

Suite à notre étude bibliographique, nous montrons que les modèles à base des Réseaux de Petri (Petri, 1962) *(RdP)* sont les plus convenables pour répondre à nos exigences : modéliser et vérifier les aspects structurels et comportementaux d'un

système dynamique. Les réseaux de Petri utilisent des présentations graphiques en présence des notations mathématiques montrant clairement un pouvoir expressif très avancé.

En effet, depuis leur apparition, les concepteurs ne cessent de proposer de nouveaux modèles à base de *RdP* soit en proposant une nouvelle extension soit en fusionnant deux extensions ou même en cherchant un nouveau domaine d'application. Toutes les recherches faites dans ce sens ont eu une influence, directe ou indirecte sur ce qu'est devenue la théorie des Réseaux réseaux de Petri au fil du temps. Cela s'est fait par l'intermédiaire de nombreux travaux, chacun amenant sa brique, soit en ce qui concerne la modélisation, soit en ce qui concerne la vérification et l'analyse. Les réseaux de Petri, comme tout formalisme qui ambitionne de modéliser des systèmes complexes tels que les Systèmes Multi Agents, se sont très rapidement confrontés à la nécessité de se doter de méthodes de structuration.

Les travaux dans ce domaine ont tout naturellement cherché à apporter à ce formalisme des techniques de structuration des données absentes dans le modèle original (Place/Transition). Par conséquent, les Réseaux de Petri Colorés (*RdPC*) ont été proposés pour apporter des primitives comme les macros places et macros transitions en émulant les notions du monde de la conception via des classes de couleurs.

Ces formalismes résultent des types de systèmes étudiés. Ils ont permis de rendre la conception par un RdP plus naturelle, intuitive et familière. Pour modéliser et analyser les systèmes à évènements discrets, particulièrement les systèmes concurrents, parallèles et non déterministes nous devons choisir le type convenable des *RdP* à utiliser. Ce type doit être capable de modéliser d'une manière rigoureuse les systèmes de grande taille telle que les Systèmes Multi Agents. Ces systèmes permettent de coordonner le comportement des Agents intelligents interagissant et communiquant dans une société pour réaliser des tâches ou résoudre des problèmes (Ferber, 1997).

Il était donc assez intelligible que les *RdP* suivent l'évolution des approches de conception. L'étape suivante consiste, tout naturellement, de considérer des réseaux ayant tendance à la modélisation et à la vérification des systèmes complexes avec

l'apparition des concepts orientés Agents. C'est ainsi que le formalisme des Réseaux de Petri à Agents ($RdPA$), par notre contribution, a vu le jour.

La complémentarité entre Systèmes Multi Agents et réseaux de Petri, que nous cherchons à développer par le biais de ce travail, apparaît alors clairement : une approche par Agents nécessite un formalisme puissant et expressif qui lui permet de modéliser et vérifier le comportement d'un ensemble d'Agents qui interagissent. En effet, nous exploitons les jetons comme étant des Agents. Nous définissions des fonctions (au cours du franchissement d'une transition) décrivant les aspects de coordination, de coopération et d'interaction. Notre nouveau modèle permettrait de décrire clairement l'Environnement et les groupes d'Agents.

Les travaux développés dans le cadre de cette thèse ont abouti à l'élaboration d'une méthode formelle basée sur les RdP et les SMA. Cette méthode constitue une extension des modèles des RdP de haut niveau. Le cahier de charges est ainsi lourd et les verrous technologiques sont nombreux. Cela résulte d'une rencontre de deux thèmes de recherche : les réseaux de Petri, et les Systèmes Multi Agents. Une piste qui reste, jusqu'à présent, très peu explorée.

Organisation de la thèse

Afin d'aboutir à cette nouvelle approche, nous avons choisi d'organiser le rapport de notre thèse comme suit :

Dans le premier chapitre, nous présentons le concept d'Agent et ses différentes classes à travers une variété de propriétés. Nous présentons, ensuite, la notion de SMA. Nous étudions ainsi la communication, la coopération et l'interaction entre les Agents, leurs formes et la complexité qui en découle. Nous étudions aussi les Agents Mobiles comme étant une entité dynamique et qui nécessite une réflexion poussée.

Le deuxième chapitre décrit les méthodes formelles, particulièrement, celles dédiées à la modélisation et à la vérification des Systèmes Multi Agents.

Le troisième chapitre fournit les éléments nécessaires à la définition de notre modèle de réseau de Petri à Agents ($RdPA$) pour la modélisation des SMA, ainsi que sa sémantique.

Dans le quatrième chapitre nous procédons à l'application de notre approche sur des exemples variés. Nous étudions, en particulier le protocole d'interaction entre les Agents.

Dans le cinquième chapitre nous étudions l'extension de notre modèle de base pour modéliser un aspect important des Agents : à savoir la mobilité.

Ce document s'achève par un ensemble de conclusions générales et de perspectives des travaux en cours et à venir.

Publications

Cette thèse a fait l'objet de nombreuses publications. Elles sont présentées ci-dessous par catégorie.

Revue Internationale avec comité de lecture

- [MBB13a] , K. Barkaoui, N. Ben Haj Alouane - APN Model for Specification of the Communication Protocols in Multi-Agent System, Journal of Software Engineering and Applications, vol. 6 (Special Issue on Software Engineering for Safety-Critical Systems and Medical Devices), pp. 14-22, 2013.
- [MB13] B. Marzougui, K. Barkaoui - Interaction Protocols in Multi-Agent Systems based on Agent Petri Nets Model, International Journal of Advanced Computer Science and Applications (IJACSA) , vol. 4(7), pp. 166-173, 2013.
- [MHB13] B. Marzougui, K.Hassine, K. Barkaoui - Modeling Migration of Mobile Agents, Lecture Notes in Business Information Processing- Springer, vol. 132, pp. 530-540, 2013.
- [MB12] B. Marzougui, K. Barkaoui - تصوّر نموذج لمرسوم التّفاعل في الأنظمة المتعدّدة الوكلاء, Arabic Computer Sociaty, vol.5 (August), pp. 65-77, 2012.
- [MHB10] B. Marzougui, K. Hassine, K. Barkaoui - A New Formalism for Modeling a Multi Agent Systems: Agent Petri Nets, Journal of Software Engineering and Applications (JSEA), vol. 3(12), pp. 1118-1124, 2010.

Conférence Internationale avec comité de lecture

- [MHB13] B. Marzougui, K. Barkaoui- نمذجة البحث عن نقطة التقاء الروبوتات المتحركة, *ICCA* 2013 International Conference on Computing in Arabic, January 2013, pp.65-77.
- [EMH13] A. Eltoumi, B. Marzougui, K. Hassine - Modelling of loading and unloading system of a ship by a mullti-agents system , Computer Applications Technology (ICCAT), 2013 International Conference on , January 2013, pp.1-6.

- [M B11] B. Marzougui, Z. Abderrahim, M. Bousada - Modeling of Dynamic Placement Problem of Task by a Multi Agent Systems, 2010 The 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010), November 2011, pp.166-171,
- [MHB11] B. Marzougui, K. Hassine, K. Barkaoui - Method for Verification of a Multi Agents System, 2011 Second International Conference on Intelligent Systems, Modeling and Simulation(ISMS'11), January 2011, pp.62-65.

1

Chapitre 1 : Les Systèmes Multi Agents

Durant ces dernières décennies, les Systèmes Multi Agents se sont imposés comme étant le paradigme le plus approprié pour résoudre les problèmes récurrents que ce soit dans le domaine de l'intelligence artificielle, dans le domaine des systèmes distribués, dans le domaine de la robotique, ou même dans ce nouveau champ disciplinaire qu'est la vie artificielle. Le thème des *SMA* est de nos jours un champ de recherche très répondu. Les SMA une discipline qui s'intéresse aux comportements collectifs produits par les interactions de plusieurs entités autonomes appelées Agents. Les Systèmes Multi Agents soulèvent de nombreuses questions. Quels sont les aspects dynamiques fondamentaux de ce paradigme? Comment on distingue ces aspects? Quels sont les caractéristiques de ces éléments de base? Quels sont les propriétés à soulever lorsqu'on souhaite modéliser et vérifier tels systèmes?

Telles sont les principales questions auxquelles le présent chapitre tentera d'apporter des éléments de réponse.

1.1. Notion d'Agent

Plusieurs travaux portent sur la notion d'Agent engendrant ainsi des définitions aussi riches que variées mais elles ne se diffèrent que selon le type d'application pour laquelle est conçu l'Agent.

1.1.1. Définition

L'un des premiers ouvrages francophones sur le sujet intitulé « *Les Systèmes Multi Agents* » de Jacque FERBER qui a défini un Agent comme étant « *une entité autonome, réelle ou abstraite, ayant la capacité de communiquer et d'agir sur elle-même et sur son Environnement constitué par d'autres Agents, et dont le comportement résulte de ses observations, de ses connaissances et des interactions avec les autres entités*».

L'auteur assume que la coordination d'actions des Agents est le fruit d'une interaction entre ses différentes entités relativement autonomes et indépendantes afin de réaliser les buts qu'ils motivent.

(Barreteau, 1998) a considéré un Agent comme étant « *une entité informatique autonome capable de communiquer avec d'autres Agents, de percevoir partiellement leur Environnement et les objets qui y sont situés et d'avoir des représentations justes ou erronées sur les comportements d'une partie ou de l'ensemble des autres Agents et de l'Environnement* ».

Un Agent est connu donc sous forme d'un système informatique, situé dans un Environnement et plongé dans une structure sociale, et qui agit d'une façon autonome et flexible pour achever les objectifs et les finalités pour lesquels il a été conçu (Nicolas, Sycara, & Wool, 1998).

L'Agent doit donc être capable de percevoir son Environnement (**Figure 1**), et d'y avoir une représentation que ce soit partielle ou totale. De ce fait, l'Agent peut alors déclencher ou recevoir une action dans son Environnement.

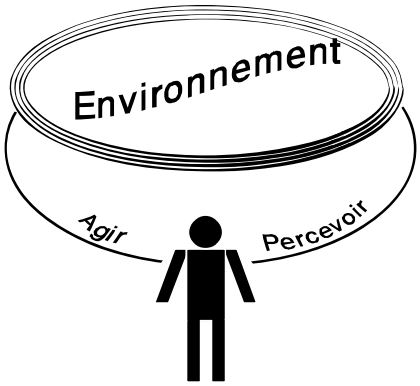


Figure 1. Agent et Environnement

Selon (Wooldridge, Jennings, & Kinny, 2000), un Agent est un système informatique, situé dans un Environnement (capable d'agir sur son Environnement à partir des entrées sensorielles qu'il reçoit de ce même Environnement), et qui agit d'une façon autonome (capable d'agir et contrôler ses actions et son état interne sans l'intervention d'un tiers) et flexible (capable de percevoir son Environnement, d'interagir avec d'autres Agents et d'intervenir dans le bon moment) pour atteindre les objectifs pour lesquels il a été conçu.

Plus tard, les travaux de (Augeraud, Collé, & Sarramia, 2006) ont schématisé un Agent comme étant une entité physique (quelque chose qui agit dans le monde réel, un robot, un avion, une voiture, etc.) ou virtuelle (module informatique) munie des ressources facilitant la communication et l'interaction avec l'Environnement. L'Agent est doté pour cela de capteurs pour percevoir, d'effecteurs pour agir ainsi que d'autres moyens de communication dont le but est d'achever une tâche bien déterminée.

(Fougères, 2000) a défini l'Agent de la façon suivante :

« *Agent:=<Communication/langage, Perception, Buts/intentions, Décision/plan, Contrôle, Identification/Interprétation, Connaissances/Mémoire, Actions/ Réactions>* »

Un Agent est formellement vu comme étant un triplet *<K, O, SR>* tel que les connaissances sont représentées par la lettre *K*, les objectifs sont représentés par la lettre *O* et le système de raisonnement est représenté par *SR*, et il est de type : *When<Event> IF <Cond>Then<Act>*

Event représente un événement qui peut être un objectif à atteindre ou bien un nouvel état d'une conversation à laquelle participe par un Agent. Dans le cas où il s'agit d'un objectif, *SR* choisit quel protocole permet d'achever l'objectif. Dans le cas où il s'agit d'un état, *SR* choisit l'action à exécuter. *Cond* est la ou les conditions qui déterminent l'action à exécuter. *Act* est l'action à exécuter qui peut se manifester sous l'une des formes suivantes:

- Initialisation d'une conversation,
- Modification des connaissances de l'Agent,
- Modification des variables de la participation,
- Sélection d'une action à exécuter de la participation.

Des définitions anglo-saxonnes sont arrivées plus tard. (Chapurlat, 2008) a affirmé, ensuite, qu'un Agent est un système informatique situé dans un Environnement et capable d'agir d'une façon autonome. Plus récemment, (Fujita, 2009) a défini l'Agent de point de vue entité interagissant avec le monde réel afin d'apprendre et développer ses propres intelligences. De ce fait, un Agent peut jouer plusieurs rôles selon l'application pour laquelle il a été conçu (**Figure 2**) dans un ou plusieurs groupes afin de réaliser des affinités bien déterminées.

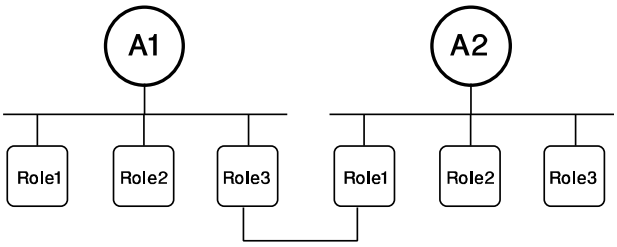


Figure 2. Agent possédant un ou plusieurs Rôles (Fujita, 2009)

Entre les différents chercheurs, il n'y a encore aucun consensus, quant à la définition du mot «Agent». Nous pouvons expliquer cette difficulté par le fait que les chercheurs, dans ce domaine, ne sont pas à l'origine de ce mot qui a été et continue d'être utilisé dans notre vie depuis longtemps et dans plusieurs domaines.

1.1.2. Propriétés d'un Agent

Partant des différentes définitions proposées de la notion d'Agent, plusieurs propriétés clés comme l'autonomie, la flexibilité, l'intelligence et la communication peuvent être distinguées et ceci dépendamment des applications. Certaines propriétés sont plus importantes que d'autres :

1.1.2.1. Autonomie

Les Agents fonctionnent sans l'intervention directe des êtres humains et des autres systèmes. Ils doivent ainsi pouvoir interagir au bon moment et prendre des décisions pour effectuer leurs tâches.

1.1.2.2. Sociabilité

Un Agent est dit sociable lorsqu'il est capable d'interagir avec les autres Agents de son Environnement pour collaborer afin de réaliser une opération commune (El jed, 2006). Il peut ainsi échanger des informations et partager des connaissances afin d'accomplir ses tâches ou aider d'autres Agents à accomplir les leurs.

1.1.2.3. Intelligence

D'après (Steels, 1994), un Agent peut paraître avoir un comportement intelligent pour l'utilisateur sans toutefois disposer des capacités de raisonnement ou de représentation symboliques sur son Environnement. En effet, l'intelligence d'un Agent se manifeste dans sa capacité d'interagir d'une manière autonome pour satisfaire ses buts.

1.1.2.4. Action et Communication

La communication entre les Agents, bien qu'elle soit aussi une forme d'action, a pour but de modifier l'état mental d'un autre Agent (Ferber J. , 1995) et de provoquer un comportement spécifique alors qu'une interaction avec l'Environnement contribue simplement à modifier son état. L'interaction, est l'une des principales propriétés d'un Agent dans un *SMA*, divisée en actions effectuées et actions utilisées pour

communiquer avec d'autres Agents de ce même Environnement (envois de messages, etc.).

1.1.2.5. Intentionnalité

Une intention est la déclaration explicite des buts et des moyens d'y parvenir (Labidi & Lejouad, 1993). Un Agent est dit intentionnel s'il est guidé par ses buts en exprimant sa volonté d'atteindre un but ou d'effectuer une action. D'autres propriétés peuvent être attribuées aux Agents selon divers points de vue et domaine d'application tels que l'engagement, le caractère situé etc.

1.1.3. Comparaison d'un Agent avec un Objet

Avec l'apparition du paradigme Agent. Ce dernier a connu un grand succès dans la spécification des systèmes complexes vu qu'il a permis de remédier aux défauts de la modélisation classique. Nous pouvons, donc, envisager une comparaison entre objet et Agent pour mieux comprendre le concept Agent. En fait, les deux concepts sont assez similaires sans nier qu'ils possèdent quelques différences. Nous présentons dans le tableau ci-dessous (**Tableau 1**) une comparaison entre les deux notions.

Agent	Object
– Comprendre des structures de comportement modulaires (méthodes/compétences) – Disposer d'un «état interne» – Poser des actions sur un état par le biais de leurs méthodes – Communiquer en s'envoyant des messages – Structure initiale	
Degré d'autonomie : recevoir une requête et décider de son propre gré s'il doit poser ou non une action. Agir pour modifier l'état	Pas d'autonomie: l'objet est invoqué par un appel de méthode qu'il ne peut refuser (pas de réactivité)
Caractère flexible (réactif, proactif et social)	Le modèle standard d'un objet n'est pas flexible
Notion d'Environnement : importante et complexe	Faible niveau social: pas d'évolution dans le temps
L'Agent est une source de contrôle	Une seule source de contrôle

Tableau 1. Comparaison entre l'Agent et l'Objet

Avant de présenter les différentes classes d'Agents, nous devons définir les propriétés de son Environnement.

1.1.4. Notion d'Environnement

Nous notons que le concept d'Agent ne se suffit pas en lui-même car ce dernier est indissociable de celui d'Environnement.

En effet, l'ensemble des perceptions et des actions qu'un Agent est susceptible de réaliser est entièrement défini par rapport à l'Environnement où celui-ci va opérer. (Michel, Gouaich, & Ferber, 2003)

Un Agent est situé dans un Environnement. Pour modéliser sa structure, il faut avoir un modèle de l'Environnement. L'Environnement peut être vu comme étant dans un état *e* parmi un ensemble d'états $E = \{e_1, e_2, \ldots, e_n\}$. L'évolution de l'Environnement se modélise différemment selon les caractéristiques que l'on prend en compte, et les simplifications que l'on s'autorise. L'Environnement, comme il a été défini par (Jean, 2002), *est « un ensemble d'éléments au sein duquel évoluent les Agents »* (**Figure 3**). En d'autres termes, c'est l'ensemble de ressources partagées par les Agents. Selon (Noél, 2009), il peut jouer un certain nombre de rôles variés tout en adoptant deux aspects : microscopique et macroscopique.

Au niveau micro, l'Environnement peut être vu comme étant un médium d'interactions, ou un coordinateur d'interactions en définissant comment une modification est mise en place lors de la perception d'un autre Agent. Ça sera ainsi le lieu de contraintes sur les dynamiques possibles des Agents.

Au niveau macro, l'Environnement en assemble les modifications faites par les différents Agents et en permet de les représenter aux Agents en utilisant une mémoire collective. L'Environnement peut être interprété soit par l'observateur extérieur, soit par les Agents.

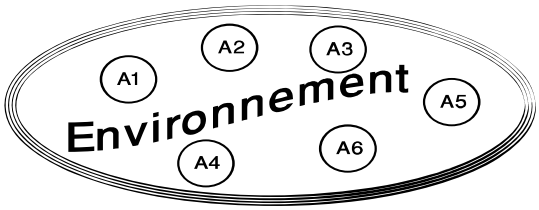


Figure 3. Environnement des Agents (Noél, 2009)

En se basant sur les travaux de (Ferber, 1995), nous pouvons, ainsi, présenter l'Environnement par les propriétés suivantes:

- *Accessible* : un Agent peut déterminer l'état de l'Environnement et agir sur lui.
- *Episodique :* si le prochain état de l'Environnement ne dépend pas des actions réalisées par les Agents.
- *Statique* : si l'état de l'Environnement ne change pas pendant que l'Agent délibère, sinon il est qualifié de Dynamique.
- *Discret* : si le nombre d'actions et d'états possibles de l'Environnement est fini, sinon il est qualifié d'Environnement continu.

1.1.5. Classes d'Agents

Selon (Ferber, 1995) deux classes d'Agents ont été distinguées: les Agents réactifs et les Agents cognitifs. D'autres classes, qualifiées d'hybrides, utilisant ces deux types de comportement, sont ensuite apparues comme des nouveaux compromis. D'autres se sont nommées Agent *BDI* (*Beliefs Desires Intentions*). En effet, la faculté de représentation et d'interprétation chez les Agents est celle qui fait la différence.

1.1.5.1. Agents Réactifs

(Brooks, 1991) a mentionné que le comportement d'un Agent réactif devrait émerger de l'interaction entre divers comportements simples. Il est alors vu comme un ensemble d'interactions accomplissant une tâche donnée, ne possédant pas de représentation symbolique ni de son Environnement, ni de lui-même. Il perçoit des stimuli venant de l'Environnement qui déclenchent une action particulière, mais, il n'est pas apte de se préparer aux futurs évènements. Nous pouvons dire qu'il est dirigé

par l'Environnement. Souvent, le comportement des Agents réactifs est construit en une hiérarchie de couches, telle que celle citée par (Brooks, 1991). Les couches du bas représentent les comportements primitifs du genre «*Eviter un obstacle*». Ce type d'Agent fait intervenir l'intelligence collective (**Figure 4**).

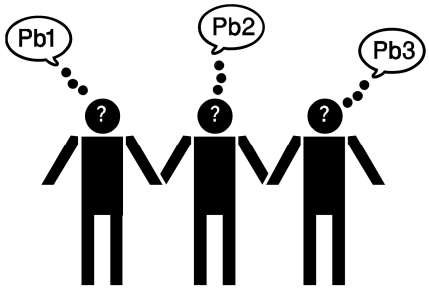


Figure 4. Intelligence Collective des Agents

Malgré En dépit de la simplicité apparente et les bons résultats obtenus pour certaines applications, des critiques ont été adressées à cette approche réactive. Ceci a été démontré par (Le Bars, 2003) et (Kolski & Mathieu, 2004). Parmi les critiques, nous pouvons mentionner les suivantes:

- Si les Agents ne disposent pas du modèle de leur Environnement, alors ils doivent acquérir suffisamment d'informations locales pour bien favoriser une action.
- Il est difficile de voir comment les Agents pourraient raisonner sur des informations non-locales vu qu'ils fondent leurs jugements sur des informations locales.
- Il est difficile de voir comment un Agent réactif peut apprendre de son expérience et progresser ainsi ses performances.
- La conception de tels Agents s'avère très délicate car ce type d'Agents représente des phénomènes difficiles et complexes.
- Les Agents contiennent des couches hiérarchiques de comportements difficiles à construire.

1.1.5.2. Agents Cognitifs

Ce type d'Agent dispose et utilise une représentation de son Environnement (les autres Agents et lui-même). Il est doté de capacités de raisonnement, en particulier, ce que l'on nomme un processus de prise de décision sur les actions à effectuer à un instant donné (Chapurlat, 2008). Il est apte de mémoriser des situations et de les analyser, et donc planifier son propre plan d'action en se basant sur un raisonnement logique (Ferber, 1995), tout en admettant les facultés fondamentales de perception, délibération et d'action, couplés à des nouveaux concepts de représentation et de régulation des processus comportementaux. La communication entre des Agents cognitifs est effectuée intentionnellement. Ce type d'Agent fait intervenir l'intelligence individuelle (**Figure 5**). Les limites de cette approche sont dues à la complexité des algorithmes de manipulation symboliques.
Selon (Le Bars, 2003), *un Raisonnement Symbolique = Formules logiques + Manipulation Syntaxique* (déduction ou preuves de théorèmes).

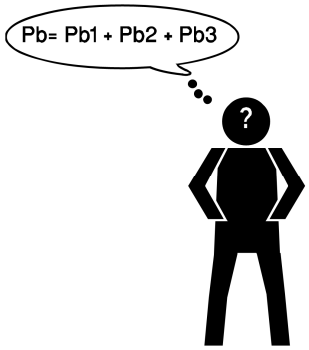


Figure 5. Intelligence Individuelle (Gouardères, 2009)

Plusieurs types d'Agents cognitifs sont distingués par (Kolski & Mathieu, 2004) (**Figure 6**). On retrouve :

- *Les Agents intelligents* : combinent les trois caractéristiques (autonomie, coopération, adaptation), dotés de la capacité d'apprentissage, de planifier leurs actions, de négocier avec d'autres Agents et d'acquérir ou modifier leurs connaissances,

- *Les Agents collaborants* : non apprenants, ils sont à la fois fortement autonomes et coopérants, mais, peu adaptatifs,
- *Les Agents d'interface (assistants)*: ils ont pour mission de capturer les actions de l'utilisateur (clavier, souris, à terme : voix, gestes, expression du visage),
- *Les Agents d'information*: dédiés à la recherche d'information. Ils possèdent une grande autonomie, capables d'agir seuls, soit en fonction d'un calendrier, soit en fonction d'un manque d'information, soit en fonction d'une nouvelle information.

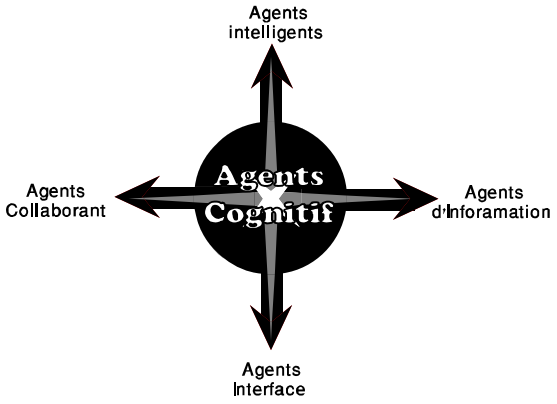


Figure 6. Classification des Principaux Agents Cognitifs (Kolski & Mathieu, 2004)

Nous présentons la différence entre les Agents réactifs et cognitifs. Le tableau suivant (**Tableau 2**) résume ces différences.

Agents réactifs	Agent cognitifs
Pas de représentation explicite de l'Environnement	Représentation explicite de l'Environnement
Pas de mémoire de son histoire	Peut tenir compte de son historique
Fonctionnement Stimulus/action	Agents complexes
Grand nombre d'Agents pour atteindre les objectifs de l'application	Nombre limité d'Agents à utiliser

Tableau 2. Comparaison entre l'Agent Réactif et l'Agent Cognitif

1.1.5.3. Agents Hybrides

Les Agents hybrides sont en quelque sorte un compromis entre des Agents purement réactifs et des Agents purement cognitifs. Ils cherchent à tirer profit des capacités intéressantes des deux types d'Agents. La structure d'un Agent hybride peut être divisée en trois couches. Au plus bas niveau, une couche réactive qui s'intéresse au traitement de données provenant des capteurs sensoriels. Une couche intermédiaire, raisonnant sur les connaissances de l'Agent à propos de son Environnement, fait abstraction des données brutes. Enfin, une couche supérieure prenant en considération les autres Agents du système, se charge des aspects sociaux de l'Environnement (Chapurlat, 2008).

1.1.5.4. Agents BDI

Cette catégorie d'Agents dite *BDI* a été développée afin de s'orienter vers la prise en compte des états mentaux (Bratman & Pollack, 1988). Elle s'intéresse aux Croyances, aux Désirs et aux Intentions, et est bâtie autour du raisonnement pratique. L'état mental se qualifie par les attitudes suivantes:

- Les croyances: ce que l'Agent connaît de son Environnement,
- Les désirs: les états possibles envers lesquels l'Agent peut vouloir s'engager,
- Les intentions: les états envers lesquels l'Agent s'est engagé, et a engagé des ressources.

Un Agent *BDI* (**Figure 7**) doit se baser sur les informations qui lui proviennent de son Environnement afin de mettre à jour ses croyances, de décider quelles options lui sont offertes, de filtrer ces options dont le but est de déterminer de nouvelles intentions et de dresser ses actions.

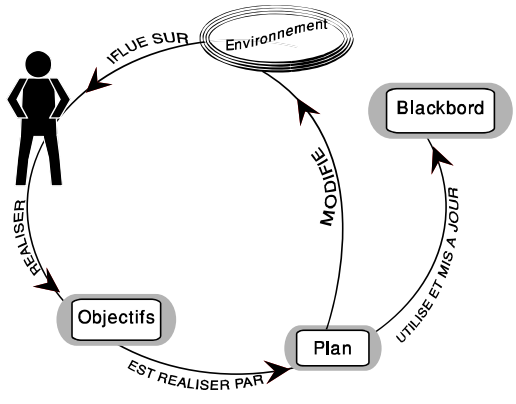


Figure 7. Représentation d'un Agent *BDI*

L'étude de notion d'Agent nous permet, dès lors, d'étudier facilement les *SMA*. Nous considérons que l'Agent constitue l'entité de base autour de laquelle sont définies les caractéristiques des *SMA*.

1.2. Les Systèmes Multi Agents

Selon (Ferber, 1995), la modélisation d'un *SMA* s'avère pertinente pour représenter les actions des Agents et leurs conséquences dans l'Environnement qui peut être complexe et d'une évolution autonome. Les Systèmes Multi Agents permettent de coordonner le comportement d'Agents intelligents interagissant et communicants dans une société pour réaliser des tâches ou résoudre des problèmes (Fougères, 2000).

Le principal intérêt des *SMA* est que ces derniers permettent de distribuer des entités communicantes, autonomes, réactives et dotées de compétences.

1.2.1. Définition

Un Systèmes Multi Agents est un système composé des éléments suivants :

- une distribution géographique des Agents ou encore un Environnement (espace disposant ou non d'une métrique),

- un ensemble d'objets situés (il est possible, à un moment donné, de leur associer une position dans l'Environnement). Ils sont passifs et peuvent être perçus, créés, détruits et modifiés par les Agents,
- un ensemble d'Agents capables de percevoir, produire, consommer, transformer et manipuler les objets de l'Environnement,
- un ensemble de relations (communications) unissant les Agents,
- un ensemble de règles de communications entre les Agents,
- un administrateur chargé de contrôler l'activation des Agents.

L'approche Multi Agents peut être considérée comme une évolution du paradigme orienté objet. Du point de vue conceptuel, un objet est simplement une structure de données à laquelle sont associées des fonctions (Barthélemy, 2005).

Un Système Multi Agents est un système distribué composé d'un ensemble d'Agents en interaction, le plus souvent, selon des modes de coopération, de concurrence ou de coexistence (Ferber, 1995) afin de résoudre une tâche donnée. Ils coopèrent entre eux afin de surmonter les conflits et anticiper la concurrence, survivre, persister et influencer le comportement des autres Agents (Fujita, 2009). Les entités en interaction avec le *SMA* forment son Environnement, nommé *Monde* comme il est indiqué par (Adam, 2008), (**Figure 8**).

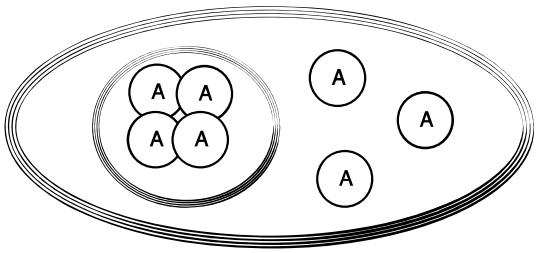


Figure 8. Le Monde d'un SMA (Adam, 2008)

Chaque Agent possède ainsi des connaissances et un savoir-faire limités, ce qui l'oblige à interagir avec d'autres pour mener à bien le projet commun (Jennings, 2000). Nous disons que la solution du problème émerge des interactions des Agents au sein de

l'Environnement (Hilaire, 2008). A titre d'exemple, nous pouvons prendre un Système Multi Agents constitué de deux Agents (**Figure 9**): un Agent *B* reçoit un problème, il l'analyse et demande à l'Agent *C* s'il peut la résoudre. Si *C* est disponible, *B* lui transmet le problème, sinon il va créer un nouvel Agent de type *C* et lui transmettre le problème. Une fois le problème résolu, *C* renvoie le résultat à *B* qui le transmettra à un autre Agent. Selon (Mathieu, Routier, & Yann , 2003), il s'agit de définir une répartition des compétences (Je sais faire : *C*) et des connaissances, à fournir un outil de recomposition (l'interaction) et à préciser les règles de la recomposition afin d'achever un objectif (But : *B*).

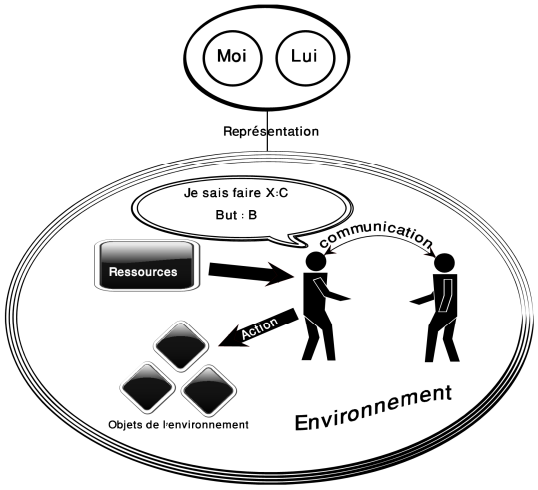


Figure 9. Simplification d'un SMA (Ferber, 1995)

1.2.2. Exemple

Pour mieux comprendre le lien entre ces différentes entités, nous pouvons nous baser sur le schéma simplifié d'un *SMA* (**Figure 10**) mis en place par (Tiberiu S, 2009). Le *SMA* est centré sur la notion d'Agents en interactions. Chaque Agent est autonome dans son activité, et peut communiquer avec n'importe quel autre Agent. Il dispose de compétences individuelles et fournit des services.

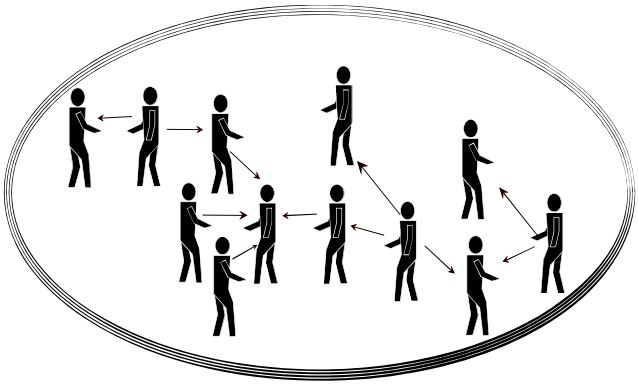


Figure 10. Interactions au sein d'un SMA (Tiberiu, 2009)

Pour mieux assimiler le concept *SMA* on peut se baser sur l'exemple concret mis en place par (Ferber, 1995) dont la tâche est d'envoyer des fleurs à un ami habitant à une autre ville (**Figure 11**):

- Bob envoie des fleurs à Alice,
- Bob ne peut pas les envoyer directement et fait appel à Fred, le fleuriste,
- Bob communique à Fred l'adresse d'Alice, le type de fleurs, et le montant,
- Fred prend contact avec le fleuriste du village d'Alice qui va préparer les fleurs, contacte ensuite un chauffeur qui va délivrer le bouquet à Alice,
- On peut continuer à réfléchir, et introduire d'autres acteurs (grossiste, jardinier, etc.).

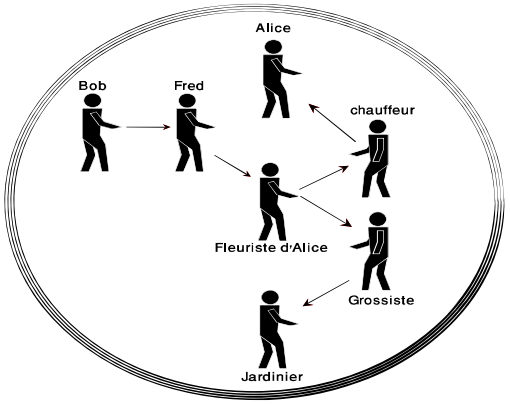


Figure 11. Exemple d'Interactions au sein d'un *SMA* (Envoie de fleurs)

En analysant la figure ci-dessous (**Figure 12**), nous pouvons mentionner que dans la communauté *SMA*, tout Agent a un rôle à jouer et un service à fournir. Un *SMA* peut être vu comme étant une distribution d'Agents capables de percevoir, produire, consommer, transformer et manipuler les objets de l'Environnement, en possédant un ensemble de relations joignant les Agents suivant un ensemble de règles de communication.
Selon (Le Bars, 2003), un *SMA* se montre sous forme d'un ensemble d'Agents qui interagissent dans un Environnement commun *E*. Il contient un ensemble *O* d'objets, un ensemble *A* d'Agents (*A* inclus dans *O*), un ensemble de relations R qui unissent des objets entre eux, un ensemble d'opérations *Op* permettant aux Agents *A* d'interagir (percevoir, produire, consommer, transformer et manipuler).

Dans un Système Multi Agents, il existe au moins un élément commun entre les différents Agents. Celui-ci peut être l'objectif visé, ou le langage de communication, ou un Environnement commun. Les interactions entre Agents au sein d'un Système Multi Agents peuvent être de type coopératif (les Agents coopèrent entre eux afin d'atteindre un objectif commun), conflictuel (les Agents poursuivent des buts différents qui peuvent être conflictuels) ou d'un type hybride entre les deux (Mathieu, Jean-C, & Secq, 2001), (Chaib-draa, Jarras, & Moulin, 2001).

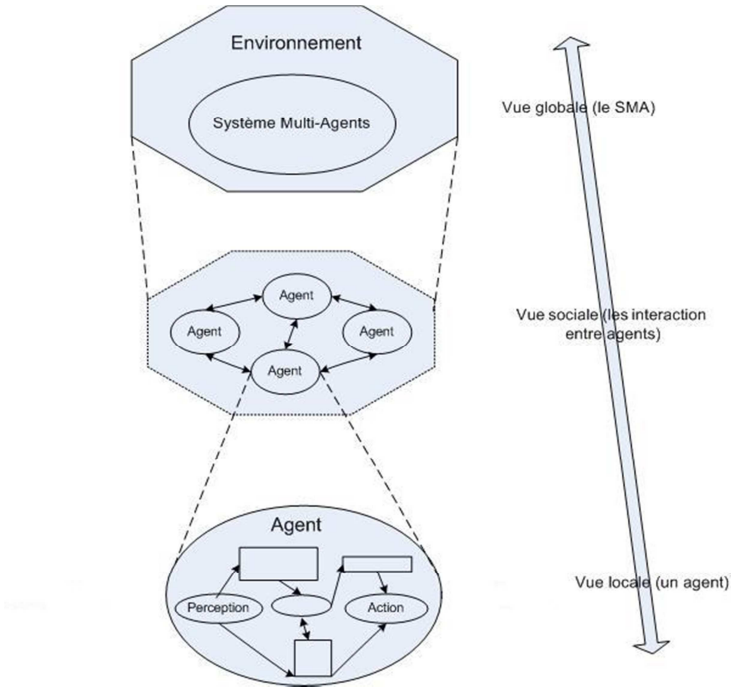


Figure 12. *SMA* Selon Différents Niveaux de Détail (Mathieu, Jean-C, & Secq, 2001)

1.2.3. Utilisations des SMA

Les différentes applications des Systèmes Multi Agents relèvent de quatre domaines :

- *La résolution de problèmes par émergence* : faire émerger une solution à un problème complexe à partir de comportements simples tel que la gestion des réseaux de télécommunication,
- *La simulation* : c'est là que les caractéristiques d'autonomie, de proactivité, des Agents interviennent. elle est utilisée pour pouvoir jouer sur des paramètres difficilement modifiables, voire non modifiables, dans les cas réels pour observer leurs influences,
- *Le contrôle de systèmes complexes* : les Systèmes Multi Agents sont aussi utilisés pour le contrôle ou le pilotage d'outils ou de composants immergés dans un Environnement dynamique, telles les chaînes de production ou encore les robots,
- *Les Environnements d'Interaction Homme Machine (IHM)* : ces Environnements sont des Environnements de développement ou des

applications d'assistanat (assistants bureautiques ou sur Internet). Le caractère fortement composite et modulaire de tels Environnements profite encore à la notion d'Agent.

1.2.4. Domaines d'Etude

Les *SMA* sont à l'intersection de plusieurs domaines scientifiques: informatique répartie, génie logiciel, intelligence artificielle et vie artificielle (**Figure 13**). Ils s'inspirent également d'études issues d'autres disciplines connexes notamment la sociologie, la psychologie sociale, les sciences cognitives et bien d'autres. Pour les Environnements où les données et le contrôle sont distribués, un *SMA* est une solution confiante, c'est aussi une solution fiable pour les applications ayant un Environnement ouvert, hautement dynamique, incertain et complexe. Disant que les Agents coopèrent ou bien qu'ils sont en compétition pour un but commun (Guyet, 2007). En outre, le concept *SMA* est devenu un concept clé dans plusieurs disciplines, tels que: les systèmes distribués, l'interface homme-machine, les bases de données et les bases de connaissances distribuées coopératives, les systèmes pour la compréhension du langage naturel, les Protocoles de communication et les Réseaux de télécommunications, Programmation Orientée Agents (POA), Génie logiciel (GL), Robotique cognitive et coopération entre robots, applications distribuées comme le web, l'Internet et le contrôle aérien.

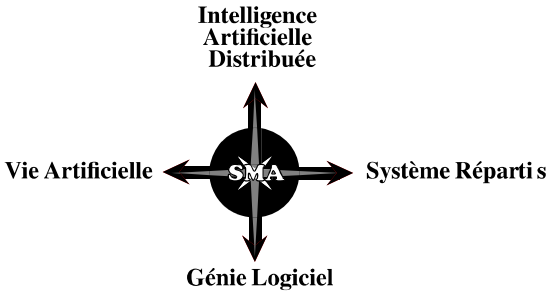


Figure 13. *SMA* et la Connexion de Multitudes de Domaines

1.2.5. Caractéristiques d'un SMA

En plus des caractéristiques individuelles des Agents, un *SMA* accumule d'autres caractéristiques décrivant en particulier les liens entre les différentes entités qui le composent.

1.2.5.1. Interactions et Coopération Entre les Agents

Un *SMA* se distingue d'une collection d'Agents indépendants par le fait que les Agents interagissent en vue de réaliser une tâche ou d'atteindre un but commun. Les Agents peuvent interagir en communiquant directement entre eux ou par l'intermédiaire d'un autre Agent.

Chaque Agent peut être caractérisé par trois dimensions: ses buts, ses capacités et les ressources dont il dispose. Il peut changer ses buts pour répondre aux besoins des autres Agents afin d'assurer une meilleure interaction entre eux (Ferber, 1995) remarque que les chercheurs ont développé différents points de vue sur la coopération qui peut être considérée comme une attitude adoptée par les Agents qui décident de travailler ensemble. C'est, par exemple, le cas d'un groupe de personnes qui acceptent de travailler dans une entreprise.

1.2.5.2. Coordination Entre les Agents

De nombreux exemples de coordination existent dans la vie quotidienne: deux déménageurs déplaçant un meuble lourd, deux jongleurs échangeant des balles avec lesquelles ils jonglent, etc.

Nous pouvons distinguer deux des composantes fondamentales de la coordination entre les Agents qui sont l'allocation de ressources rares et la communication de résultats intermédiaires.

Cette caractéristique s'avère très pratique dans des Environnements dynamiques où les Agents peuvent échouer dans leurs tâches ou découvrir de nouvelles opportunités. Dans de telles situations, un *SMA* doit pouvoir évaluer sa performance et être en mesure de se réorganiser en conséquence soit par une planification centralisée soit par

une planification distribuée, où chaque Agent se charge de produire des plans qui seront ensuite coordonnés par un coordonnateur. De point de vue du concepteur d'un *SMA,* diverses questions doivent être considérées pour assurer la coordination comme indiqué dans (Fougères, 2000):

- Avec quel Agent, un Agent doit-il coordonner ses actions?
- Quand et où ces actions de coordination doivent-elles être accomplies?
- Comment détecter et traiter les interactions entre actions (conflits et renforcement)?

1.2.5.3. Négociation Entre les Agents

La négociation joue un rôle fondamental dans les activités de coopération en permettant aux Agents de résoudre des conflits. (Durfee & Montogomery, 1990) ont défini la négociation comme étant « *le processus d'amélioration des accords (en réduisant les inconsistances et l'incertitude) sur des points de vue communs ou des plans d'action grâce à l'échange structuré d'informations pertinentes*». Les Agents qui ont les ressources appropriées, l'expertise ou l'information requise, envoient au gestionnaire des soumissions ("bids") qui indiquent leurs capacités à réaliser la tâche annoncée. Le gestionnaire évalue les soumissions et accorde les tâches aux Agents les mieux appropriés. Ces Agents sont appelés des contractants ("contractors"). Enfin, les gestionnaires et contractants échangent les informations nécessaires durant l'accomplissement des tâches.

1.2.6. Interaction Inter-Agents

1.2.7. Définition

L'interaction est, avec l'organisation, l'un de concepts de base des Systèmes Multi Agents. Selon (Ferber, 1997) *"pour un Agent, interagir avec un autre constitue à la fois la source de sa puissance et l'origine de ses problèmes".* En effet, c'est la coopération des Agents qui leur apporte une sorte d'intelligence ou de capacité à résoudre des problèmes assez complexes, mais c'est aussi à cause de leur multitude que

les conflits apparaissent. La notion d'interaction constitue l'essence d'un Système Multi Agents puisque c'est grâce à elle que les Agents vont pouvoir produire des comportements collectifs complexes (Thomas, 2005) et dépendants les uns des autres. Selon (Ferbe, 1997), ce terme désigne les couplages entre Agents mais désigne aussi la notion de situation d'interaction: "*On appellera situation d'interaction un ensemble de comportements résultant du regroupement d'Agents qui doivent agir pour satisfaire leurs objectifs en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent et de leurs compétences individuelles.*"

(Ferber, 1997) fournit une classification des situations d'interaction selon plusieurs critères :

- La présence d'objectifs communs ou compatibles,
- L'accès à des ressources communes,
- La répartition des compétences au sein des Agents.

Le terme interaction peut aussi correspondre aux manières selon lesquelles ces couplages entre entités s'effectuent au sein du système (Thomas, 2005). En fonction de ces critères et de l'objectif du système, l'interaction peut être indirecte ou directe.

1.2.8. Interaction Indirecte

Une interaction est qualifiée d'indirecte si elle n'est pas adressée explicitement à un autre Agent mais se fait à travers l'Environnement (Koning & Hernández, 2003), qui conserve une trace de cette interaction. Les Agents participant à cette interaction sont les Agents qui vont percevoir ces changements de l'Environnement. Ainsi, un Agent en effectuant une interaction indirecte, n'est pas certain de savoir avec quel(s) autre(s) Agent(s) il est en train d'interagir puisqu'il ne sait pas quel Agent sera amené à modifier son comportement en observant ces changements dans l'Environnement comme le montre la **Figure 14**.

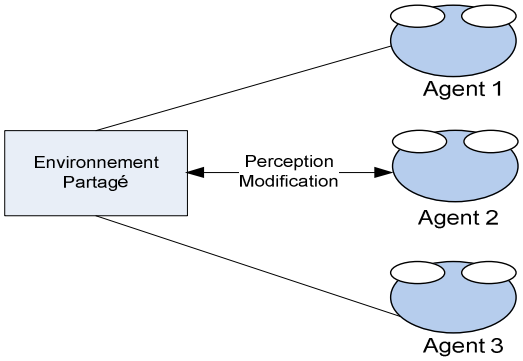


Figure 14. Interaction Indirecte entre les Agents

1.2.9. Interaction Directe

Une interaction est dite directe si elle est ponctuelle et dirigée explicitement vers un destinataire (un Agent ou un groupe d'Agents) dans le but de modifier son comportement (ou état interne) et ceci sans que l'état de l'Environnement ne soit affecté directement (Augeraud, Collé, & Sarramia, 2006). L'interaction directe est fondée sur des envois de messages entre les Agents et sur des échanges d'information (Koning & Hernández, 2003). Ses conséquences sont décidées par les lois comportementales des Agents comme illustré dans la **Figure 15**.

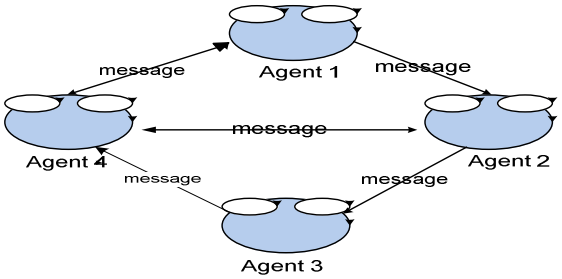


Figure 15. Interaction Par Envoi des Messages Entre les Agents

En fonction des types des Agents impliqués, l'interaction directe peut aussi prendre de nombreuses formes. Elle peut s'exprimer pour des Agents réactifs à l'aide d'échanges de signaux simples (comme dans le cas de l'éco-résolution (Ferber, 1997) et pour des Agents cognitifs, à l'aide de langages et de protocoles de communication élaborés. Elle s'inspire de l'interaction sociale (la communication entre humains) et supporte une vision de l'interaction et de la communication de haut niveau (Koning & Hernández,

2003). Ainsi, les recherches en *SMA* se sont orientées vers des modèles de communication plus complexes en considérant, à la manière de la philosophie du langage, que l'unité de base dans la communication est l'acte de langage.

1.3. Les Agents Mobiles

Un cas particulier des *SMA* est le cas où un Agent a la capacité de se déplacer de site en site tout en ayant la conscience de son déplacement. Ceci est connu sous le terme d'Agent mobile. Ce concept est issu principalement de deux domaines distincts : l'intelligence artificielle et les systèmes distribués avec la migration de processus. Ce caractère de mobilité ajouté aux caractéristiques intrinsèques des Agents (autonomie, social, proactif, etc.) offrent des atouts pour résoudre plusieurs problèmes.

Plusieurs travaux ont porté sur la notion d'Agent engendrant ainsi des définitions aussi riches que variées selon la classe et le type d'Agent. En effet, la programmation Agent se démarque parfaitement des précédents paradigmes de programmation. En particulier, la notion de localité ne concerne plus seulement le code et l'état des variables, mais également les invocations des méthodes. En d'autres termes, le moment et la façon dont un Agent réagit à un évènement externe (ou interne) ont désormais déterminés par ce dernier.

Dans la littérature, plusieurs définitions ont été proposées autour des Agents mobiles :

- Les Agents mobiles peuvent être vus comme des composantes logicielles qui selon leurs volontés, suite à une demande ou suite à une invitation, peuvent migrer d'un hôte à un autre afin de s'exécuter. La migration comporte non seulement le code de l'Agent mais aussi l'état d'avancement des processus qui lui sont liés (Fougères, 2000).

- Un Agent mobile n'est pas limité par les frontières du système où il a commencé son exécution. Il a la capacité de se transporter d'un système du réseau vers un autre. L'état de l'Agent (l'état de son exécution et de ses attributs) ainsi que son code sont aussi transportés au cours de cette opération de mobilité (Chaib-draa, Jarras, & Moulin, 2001).

Nous remarquons que le code de l'Agent est déplacé d'un système à un autre en insistant sur le fait que, d'une part, l'état de l'Agent et de ses attributs sont aussi déplacés. D'autre part, l'Agent peut effectuer l'opération de migration de façon autonome, c'est à dire que c'est lui qui décide quand et où il va migrer.

L'Agent mobile est créé avec des buts, des capacités, des compétences, des moyens de communication, un certain degré d'intelligence et d'autonomie. Jusqu'ici nous sommes toujours dans le cas d'un Agent au sens classique. Mais, cet Agent a la possibilité de quitter son lieu de naissance pour migrer d'un site à un autre à la recherche d'informations introuvables sur son site de création ou à la rencontre d'autres Agents afin de mieux exécuter ses tâches (Bernich & Creteil , 2007).

Un Agent qui reçoit l'ordre ou décide de migrer, se déplace dans le réseau de machines et accédera localement aux différentes ressources et services offerts par ces hôtes. Pour réaliser ces opérations nous pouvons distinguer trois phases :

- L'Agent mobile décide de migrer, il établit la description de sa mission,
- L'exécution de la mission par l'Agent. Ce dernier se déplace pour accéder aux différents services et aux ressources offertes par le site hôte,
- Le retour de l'Agent mobile au site de départ et la récupération éventuelle des résultats de sa migration.

Ces trois étapes nous permettent de décrire le cycle de vie d'un Agent mobile (**Figure 16**).

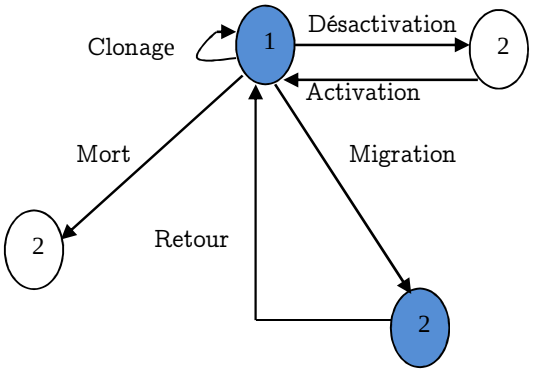


Figure 16. Cycle de Vie d'un Agent Mobile

Nous expliquons ce cycle dans le **Tableau 3** :

Etat / Transition	Explication
1	**É**tat initial : création de l'Agent. C'est au cours de cette étape que l'Agent aura un nom, des objectifs à atteindre et des capacités.
2	**É**tat de repos. : l'Agent peut, pour de raisons diverses, être désactivé (attente de ressource par exemple). Il sera donc stocké mais pourra reprendre une activité normale quand c'est possible.
3	Migration de l'Agent : C'est l'étape la plus importante. Pour satisfaire ses besoins, l'Agent décide de quitter le lieu dans lequel il se trouve pour aller vers un autre site.
4	Mort de l'Agent : l'Agent est supprimé soit à la fin d'exécution de ses tâches, soit c'est le site qui l'accueille qui décide ainsi, pour des raisons de sécurité entre autres.
Clonage	C'est une action héritée des concepts d'Agent classique. L'Agent aura une copie avec un nom et un identificateur différent (la copie sera à l'état initial même si son « père » ne l'est pas).
Migration	L'action de migration est un processus délicat et complexe. En fait, c'est le passage d'un Environnement à un autre.
Retour	C'est une migration dans le sens inverse. Une fois l'Agent a terminé l'exécution de sa tâche pour laquelle il a migré, il devra retourner à son site d'origine et continuer son activité localement.
Désactivation	Un événement s'est produit et l'Agent doit être stocké et mis au repos momentanément.
Activation	Un événement s'est produit, il faut relancer l'Agent à l'état où il s'est arrêté au moment de son stockage.

Tableau 3. Description de Cycle de Vie d'un Agent Mobile

La notion de clonage consiste à avoir un Agent identique au créateur, il aura ainsi les mêmes capacités mais aura un nom et des buts différents. Cette capacité à se faire

cloner permet d'assurer l'ubiquité de l'Agent et ainsi quand l'Agent est amené à réaliser des opérations en parallèle il lui suffit de se cloner et d'affecter à chacun de ses clones une des tâches à réaliser.

L'activation et la désactivation d'un Agent peuvent être considérées comme des opérations qui vont à l'encontre du principe d'autonomie, mais certaines mesures de sécurité ou de gestion du site receveur imposent ces actions. Notons que dans certains cas le site hôte peut même demander à un Agent de partir et lui fixer un délai avant de le détruire. Au cas où un Agent serait détruit par un des sites sur lesquels il naviguait ou s'il n'arrive pas à rentrer au site d'origine et se perd sur le réseau (l'Agent est bloqué sur une machine qui s'est déconnectée), il faudra prendre les précautions nécessaires pour générer un autre Agent identique afin de recommencer l'opération ou la modifier et éviter ainsi l'échec de la migration de nouveau.

Nous pouvons remarquer que l'un des avantages qu'offre le paradigme des Agents mobiles est la suppression de la contrainte de la connexion en continue. En effet, il suffit que la connexion soit établie lors de la migration pour que l'Agent puisse effectuer un nombre important de tâches en se déplaçant d'un site à un autre. Notons que pour les grands réseaux et les machines nomades la connexion de façon continue n'est pas toujours réalisable. L'opération de migration est à gros grains, puisque l'Agent migre une seule fois sans interruption et il n'y a pas d'échange de messages de longue durée pour cette opération. Mais pour cette migration nous pouvons distinguer deux cas :

- *Mobilité faible* : le système sauvegarde le code exécutable et s'occupe de l'opération de migration sans se soucier du contexte de l'Agent. Ce type de migration est dit non transparent. En effet l'Agent devra continuer son activité en des points différents de son point d'arrêt sur le site de départ, dans plusieurs cas l'Agent est même amené à recommencer son activité dès le début (Boussard, Bouzid, & Mouaddib, 2007).
- *Mobilité forte* : contrairement à la mobilité dite non transparente, dans le cas d'une mobilité forte, le système transporte non seulement le code exécutable mais aussi les données et l'état d'exécution de l'Agent qui une

fois sur le nouveau site pourra continuer son activité exactement là où il s'est arrêté avant sa migration. Donc la mobilité concerne le code et le contexte courant de l'Agent, ce qui garantit la transparence de l'opération de migration. En effet pour un programmeur une seule instruction suffit pour déplacer l'Agent. L'instruction d'après, ce dernier continuera son activité normalement sur le site destination (Boussard, Bouzid, & Mouaddib, 2007).

Cependant, il ne faut pas considérer la mobilité forte comme étant la meilleure alternative et ceci principalement pour deux raisons. D'une part, la sauvegarde du contexte de l'Agent est une opération très délicate puisqu'il faut tenir compte de tous les objets et de toutes les variables utilisées ou pointées par ce dernier. Il suffit d'examiner les problèmes rencontrés pour la capture d'états des threads d'un Agent écrit en java. D'autre part, certaines applications ne nécessitent qu'une mobilité faible. Donc le choix entre l'utilisation d'une mobilité faible et une mobilité forte dépend du type d'application et du coût d'une migration transparente.

Un troisième type de mobilité est mentionné dans la littérature, il s'agit de la mobilité nulle. C'est le cas où les messages échangés ne contiennent pas du code mais uniquement des données. Dans le cadre de notre travail, ce cas ne nous intéresse pas et ne sera pas étudié.

1.4. Problématiques dans les SMA

Nous pouvons relever plusieurs problématiques lors de la création de Systèmes Multi Agents:

- D'abord, la problématique de l'exécution des actions : comment un ensemble d'Agents peut agir de manière structurée dans un Environnement dynamique, et comment cet Environnement interagit en retour avec les Agents? L'Agent doit être capable de raisonner sur les actions dont il est dirigé par les buts disposant de pré-conditions et de post-conditions. Les questions sous-jacentes sont celles de la représentation de l'Environnement par les Agents, de l'interaction inter Agents et de la planification Multi Agents.

- Ensuite la problématique d'intégration de l'Agent et de sa relation au monde, qui est représentée par le modèle cognitif. L'Agent invoque un comportement organisationnel qui peut lui permettre d'attendre son but au sein de l'Environnement. L'Agent doit être capable de mettre en œuvre les actions qui répondent au mieux à ses objectifs. Cette capacité à la décision est liée à "son état mental" qui reflète les perceptions, les représentations, les désirs, les croyances et les tendances.
- Les Systèmes Multi Agents se basent, d'un point de vue communication, sur les processus d'Interaction, de collaboration et de coopération.
- Nous pouvons évoquer la problématique de l'adaptation et d'apprentissage (individuelle ou collective).
- D'un point de vue implémentation, nous identifions les problèmes de la réalisation effective et de l'implémentation des applications Multi Agents (les langages de communication entre Agents, la description des lois de l'Environnement et de représentation des connaissances).
- Finalement, la modélisation et la vérification de tous les problèmes précédents deviennent une nécessité et un défi majeur à résoudre.

1.5. Conclusion

Malgré les travaux de recherches avancées contournant les *SMA*, les chercheurs sont toujours en besoin de définir leurs systèmes par le biais des entités de base : Agent et Environnement. Les définitions fameuses de (Tiberiu, 2009), (Ferber, 1995), (Ferber, 1997), (Fougères, 2000), (Jennings, 2000), (Jean, 2002), (Wooldridge, Jennings, & Kinny , 2000) et (Chaib-draa, Jarras, & Moulin, 2001) sont, jusqu'à présent, le fondement de toute autre définition.

La définition que nous avons retenue pour déterminer les caractéristiques structurelles ou comportementales d'un Agent nous mène à définir, implicitement, le Système Multi Agents.

Le principal intérêt des *SMA* est qu'ils permettent de distribuer des entités communicantes, autonomes, réactives et dotées de compétences. Cet avantage permet d'augmenter le taux d'utilisation des *SMA* dans divers domaines. Généralement, on

ne trouve pas un Agent seul dans son Environnement. Il est recommandé qu'il y ait d'autres Agents, d'où l'opportunité d'interaction entre eux : coexister, coopérer et se compéter au sein d'un système où évoluent plusieurs entités. En effet, la construction de ce type de système comporte toutes les difficultés inhérentes aux systèmes répartis, auxquelles s'ajoute le caractère flexible et sophistiqué des interactions. Pour cela, leur modélisation nécessite un effort supplémentaire, puisque les systèmes étudiés deviennent eux aussi tellement plus complexes que les outils classiques qualifiés ne sont plus capables de réduire la tâche de modélisation. Nous envisageons, clairement, l'utilisation des méthodes formelles pour répondre aux besoins croissants de modélisation et de vérification des *SMA*.

2

Chapitre 2 : Méthodes de Modélisation et de Vérification Formelles

La qualité d'un logiciel s'évalue par rapport à la satisfaction des exigences à la fois recommandées par l'utilisateur et l'Environnement, et au même temps par son comportement qui doit rester correct et cohérent. En effet, le passage à la concrétisation n'est envisageable que si la spécification est bien validée par rapport aux fonctionnalités attendues du système; d'où la nécessité de la mise en place d'outils et de formalismes pour accomplir la vérification et la validation du modèle abouti. Un enjeu important dans le domaine des *SMA* est de combiner les techniques de vérification formelle déjà existantes et de proposer de nouvelles méthodes qui permettent d'étudier les propriétés du bon fonctionnement du système dans le but de s'assurer de la conformité et de la cohérence de ces modèles et de la satisfaction de l'utilisateur.

2.1. Les Méthodes Formelles

2.1.1. Nécessité et Importance

La question concernant l'amélioration de notre confiance vis-à-vis des modèles conçus s'impose étant donné que n'importe quel système est exposé aux erreurs vulnérables de conception, et que la moindre défaillance d'un logiciel peut entraîner des problèmes de pertes de résultat, voir même la destruction du système. Dans le cadre des systèmes de plus en plus caractérisés par la complexité et le dynamisme, l'amélioration de la pertinence de ces modèles vis-à-vis des exigences des systèmes concrets, réclame de plus en plus de réflexions scientifiques. La solution attendue est de proposer des techniques fiables assurant la modélisation et la vérification des systèmes pour se protéger contre les bugs énormes qui peuvent être causés par les erreurs de conception détectés tardivement et qui peuvent avoir des conséquences dramatiques sur la qualité, et les délais de livraison.

Dans notre contexte des *SMA*, la vérification et la validation du modèle sont considérées parmi les questions les plus fréquentes dans la modélisation surtout pour les systèmes dont la sûreté de fonctionnement constitue une préoccupation majeure.

2.1.2. Types d'Approches Utilisées

La modélisation, la vérification et la validation des *SMA* se basent sur des techniques mises en place par d'autres disciplines, que sur des approches développées dans le domaine des SMA. Ces approches peuvent être classées en 4 catégories : les approches conventionnelles, les approches semi formelles, les approches formelles, et les approches hybrides (Gómez, Pavón, & Pavón, 2004):

- *Approches conventionnelles*: elles s'intéressent aux méthodologies qui fondent leurs spécifications sur le langage naturel complété avec des diagrammes ou des images. On doit souligner que ces approches ne sont pas faciles à étudier parce qu'elles se basent sur les symboles de communication journalière.
- *Approches semi-formelles*: elles se basent sur les diagrammes pour décrire les informations, et possèdent une syntaxe bien définie avec une utilisation des langages naturels et ceci d'une façon limitée. Par conséquent elles manquent de

bases théoriques solides permettant de prouver la validité des systèmes vu que leur sémantique n'est pas précise.

- *Approches formelles*: elles se basent sur des principes mathématiques vérifiant des propriétés des systèmes de façon à démontrer si celles-ci sont satisfaites ou pas dans le cas des problèmes concrets. Ces approches offrent la possibilité d'examiner et de prouver des propriétés sur les modèles extraits à partir de la spécification en suivant les modes de raisonnement symboliques, que ce soit le procédé de démonstration de théorèmes « theorem proving » ou en procédant à des recherches exhaustives connues sous le nom d'évaluation des modèles « model checking ».
- *Approches hybrides*: elles combinent les approches précédentes en vue d'augmenter l'utilisabilité et la puissance d'analyse. Elles font recourt au approches semi- formelles comme outils de communication, et au approches formelles en cas où on a une spécification sophistiquée.

Notre choix a été porté sur l'approche formelle vu qu'elle présente des avantages indéniables dans un domaine où le zéro-erreur est nécessaire. Son intérêt majeur est de pouvoir prouver et vérifier des propriétés en raisonnant sur des modèles à base de principes mathématiques et en mettant en place une syntaxe et une notation définies rigoureusement par une sémantique précise. L'utilisation de telles méthodes est de plus en plus justifiée pour les logiciels qui nécessitent la certification de la spécification, puisqu'elles permettent de s'assurer de leur bon fonctionnement et leurs complétudes et d'éviter ainsi les risques d'erreur, d'incertitude et d'ambiguïté en apportant plus de précision dans la phase de modélisation avec un haut niveau d'abstraction réduisant le délai et le coût de développement.

2.1.3. Définition des Méthodes Formelles

Les méthodes formelles ont été utilisées pour assurer un niveau de précision, de cohérence et d'exactitude assez élevé. Elles sont basées sur des fondements mathématiques afin de diminuer les risques d'incertitude et d'ambiguïté. En phase de conception des logiciels, les méthodes formelles permettent à un langage particulier,

d'exprimer très rigoureusement les propriétés issues du cahier des charges (Lamy & Blaise, 2005).

Cependant, ces méthodes utilisent des notations et des concepts spécifiques qui génèrent souvent une faible lisibilité et une difficulté d'intégration dans les processus de développement et de certification (Idani, 2006). Les spécifications formelles sont exprimées dans des langages utilisant des syntaxes et une sémantique bien précise et stricte. Les validations automatiques résultent d'une base théorique solide, ainsi l'intégration de plusieurs méthodes formelles est en effet difficile (Gervais, 2004).

Les méthodes de spécifications formelles sont utilisées en génie logiciel pour raisonner sur des modèles mathématiques. L'intérêt est de pouvoir montrer ou vérifier des propriétés sur des systèmes. Malgré les coûts supplémentaires liés au travail d'analyse, de conception et de modélisation en spécification formelle, l'utilisation de telles méthodes est de plus en plus justifiée pour des logiciels qui impliquent des données ou des conditions de sécurité critiques, car elles permettent d'assurer le minimum de fiabilité et d'éviter des risques d'erreur.

En outre, une spécification formelle peut apporter de nombreux avantages. Les méthodes sont plus riches et permettent de mieux représenter les aspects statiques et dynamiques des systèmes. Les méthodes formelles ont été utilisées avec succès dans des domaines variés, dont la conception des interfaces hommes machine, les systèmes embarqués, les systèmes temps réel, etc.

Pour les systèmes complexes tels que les systèmes distribués et les Systèmes Multi Agents, l'analyse et la vérification sont beaucoup plus difficiles car ils peuvent produire des comportements non désirés.

A l'aide des méthodes formelles, on peut révéler que des ambiguïtés existent, que des cas sont incompréhensibles ou de même révéler certaines incertitudes. En effet, nous pouvons découvrir les failles de conception dans un temps très réduit par rapport aux autres méthodes classiques qui utilisent des étapes coûteuses. Ainsi, les méthodes formelles peuvent être vues comme des méthodes intéressantes, puissantes et complémentaires entre elles ou entre d'autres méthodes classiques comme le test et le

débogage. Chaque méthode hérite d'une autre un certain nombre de propriétés qui seront toutefois réutilisées ou bien raffinées. Ainsi, une grande complémentarité se présente en passant d'une méthode à une autre en se basant sur un modèle dit modèle de base.

Il est alors nécessaire d'approcher les Systèmes Multi Agents à l'aide de méthodes discrètes et finies, sur lesquelles on peut appliquer des techniques précises de vérification.

Pour les approches utilisées au niveau de la spécification des systèmes, on peut s'orienter vers deux approches : l'une classique et l'autre formelle. Pour l'approche classique, elle commence par une phase de spécification fonctionnelle, puis par la conception générale suivie par une conception détaillée et se termine par l'obtention du code. Pour l'approche formelle, on commence par un modèle général qui sera vérifié mathématiquement pour obtenir un premier modèle affiné qui à son tour subi d'autres vérifications. Ce traitement se répète jusqu'à l'obtention d'un modèle affiné détaillé qui conduit au codage.

En outre, un langage de spécification est dit formel lorsque sa syntaxe et ses notations sont rigoureusement définies par une sémantique précise, c'est-à-dire avec des modèles mathématiques (Hernandez, 2004). Une méthode de spécification est dite formelle lorsqu'elle utilise un ou plusieurs langages de spécification formels.

Le grand intérêt des méthodes de spécification formelle est de réduire la nécessité de toute preuve manuellement, et de remédier aux problèmes classiques de manque de compréhension des spécifications (par le biais d'un langage non ambigu pour faire le modèle).

Rappelons toutefois, que dans notre étude, nous nous intéressons à la modélisation des interactions entre Agents à travers un protocole. La nature complexe d'une telle tâche dicte le besoin d'utiliser un formalisme ayant un grand pouvoir d'expression pour modéliser, analyser, vérifier et valider les interactions. Bien que divers langages formels aient été utilisés à ce propos, nous avons opté pour les Réseaux de Petri qui ont l'avantage de représenter les notions de concurrence, de synchronisation, de modularité et de réutilisabilité et d'une facilité de conception.

2.1.4. Classification des Méthodes Formelles

Actuellement les formalismes de spécification sont nombreux et variés. Ils ont des inconvénients et des avantages et personne ne peut prétendre à une spécification exhaustive. Il existe plusieurs approches de classification des méthodes formelles de modélisation et de spécification. Une de ces approches les distingue entre deux catégories (**Tableau 4**) :

	Techniques basées sur les états	Techniques basées sur les évènements
Principes de base	Elles comportent un aspect statique pour décrire les entités du système et un aspect dynamique pour décrire les changements d'état que le système peut subir via des actions ou des opérations. Elles attribuent des propriétés d'invariance pour assurer la cohérence du système.	Elles représentent le système sous forme d'entités indépendantes en interaction, qui communiquent entre elles ou avec l'Environnement. Elles utilisent des séquences ou des arbres d'évènements pour modéliser le comportement du système.
Exemples	*Statechart , Esterel , ASM , Action Systems, VDM, Z , Object-Z ,B*	Les Réseaux de Petri, *LOTOS, CCS, CSP, EB3.*

Tableau 4. Classification des Techniques de Vérification Formelles

- **Les méthodes basées sur des états**

Les méthodes basées sur les états représentent le système à travers deux modèles complémentaires: la partie statique qui permet de décrire les entités constituant le système et leurs états, tandis que la partie dynamique qui modélise les changements d'états que le système peut effectuer par l'intermédiaire d'opérations ou d'actions. Des propriétés d'invariance sont souvent définies sur le système pour assurer la cohérence du système.

Les langages s'appuyant sur les états sont par exemple : *Statchart, Esterel, Z, Object-Z* et *B*. etc.

- **Les approches basées sur les événements**

Les approches basées sur les événements représentent le système à travers des processus ou des Agents, qui sont des entités indépendantes qui communiquent entre elles ou avec l'extérieur du système. Ces méthodes permettent de modéliser le comportement du système à l'aide de séquences ou d'arbres d'événements. Parmi les exemples de formalismes basés sur les événements, nous pouvons citer : les Réseaux de Petri, *LOTOS*, *CCS* et *CSP*.

Grossièrement, ces deux méthodes de vérification des systèmes, considérées formelles, sont connues respectivement sous le nom de : La preuve de théorème, et le *Model Cheking*:

- *La preuve de théorème* : le système et les propriétés à prouver sont tous les deux représentés sous formes de formules logiques qui peuvent être combinées entre elles en respectant des règles dont le but de déduire des nouvelles formules pour montrer la correction du système. Il est pourtant à noter qu'en cas d'erreur, si une propriété du système n'est pas de celui-ci, il est difficile de trouver le comportement favorable du système.

- *Le Model checking* : ces méthodes procèdent différemment, elles modélisent le système sous forme d'un espace d'états qui sont reliés entre eux via des liens, ce qui facilite la vérification algorithmique de ces propriétés. Ces méthodes sont en mesure d'exhiber des contre-exemples au cours de la vérification (retour sur erreur), ce qui représente le point fort du model checking vis-à-vis de la preuve de théorèmes.

Il existe également d'autres approches dites hybrides permettant de compléter une axiomatisation par un modèle de données ou bien de compléter un modèle de données par une axiomatisation. Il y a par exemple les approches algébriques comme *CASL* ou *LARCH*, ou bien des méthodes combinant plusieurs aspects comme *RAISE* ou *LOTOS*.

Dans la section suivante, nous décrivons les méthodes de vérification les plus utilisées, à savoir, *VDM*, *Logique temporelle*, *Equation différentielle*, *Langage Z*, *Méthode B* et *RdP*.

2.1.4.1. VDM

Vienna Development Method développée par (Jones, 1978), est une méthode de développement basée sur un ensemble de techniques de modélisation permettant l'analyse en descendant vers une conception et une implémentation de plus en plus détaillée, mélangeant des notions concrètes comme le type de données à des notions plus abstraites comme des prédicats "avant-après" sur les opérations.

VDM permet de vérifier quel état initial établi l'invariant (expression des restrictions sur les variables), que chaque opération préserve l'invariant, et que les réponses indiquées peuvent effectivement être produites. Pour cela, cette méthode définit les propositions comme étant une expression booléenne ayant une valeur de vérité Vrai ou Faux, les prédicats comme étant une proposition avec des variables, les fonctions, les opérations réalisées, les ensembles, les objets composites, les invariants, les correspondances et les séquences entre les états du système.

2.1.4.2. La logique Temporelle

En plus des opérateurs logiques classiques ($\wedge, \neg, \vee \cdots$), la logique temporelle ajoute des opérateurs temporels pour désigner des propriétés concernant les successeurs d'un état dans le temps.

CTL (*Computation Tree Logic*) est une logique à temps arborescent où chaque état est prévu avoir plusieurs successeurs possibles.

LTL (*Linear Time Logic*), en revanche, est une logique à temps linéaire où l'on considère l'ensemble des séquences d'exécution possibles du système.

- *CTL*: Pour vérifier un système, il faut exprimer les propriétés à valider sous forme de formules notées Φ qui seront vérifiées via un modèle mathématique qui s'applique sur un graphe d'états représentant le système en tenant compte d'un aspect non déterministe des choix des branches en chaque état. Nous

pouvons vérifier qu'une propriété est vraie pour au moins un choix de branche, ou bien qu'elle soit vraie pour tous les choix (Richard, Comet, & Bernot, 2006).

- *LTL*: on vérifie une propriété sur toutes les exécutions possibles de processus, ce qui permet d'exprimer directement certaines propriétés importantes comme l'équité. La vérification d'une formule *LTL* étant une opération de complexité exponentielle par rapport à la taille de la formule peut contenir un ensemble de propositions atomiques (p_1, p_2, ..., p_n), des connecteurs booléens et des opérateurs temporels, $X(next)$ et $U(until)$.

2.1.4.3. Equation différentielle

C'est une démarche de modélisation mathématique d'un système évolutif, connu comme étant une relation de la forme $F\ (x,\ y(x),\ y'(x)) = 0$ entre la variable x, la fonction $y(x)$ et sa dérivée $y'(x)$. La fonction f est dite solution (ou intégrale) de l'équation différentielle sur un intervalle I si pour tout $x \in I$, $F\ (\ x,\ f(x),\ f'(x)) = 0$

Exemple : $x.\ y'(x) - 2y(x) = 0$ admet pour solution $y\ (x) = x^2$ sur R.

Nous représentons le système sous forme de variables qui seront représentées dans l'équation afin de calculer les solutions qui seront ensuite facilement vérifiées à l'aide de la représentation mathématique qui peut avoir plusieurs formes.

2.1.4.4. Le langage Z

Le langage *Z* est un formalisme qui a fait l'objet de nombreuses études. Ce dernier a servi à la spécification d'applications industrielles. Il est inspiré de la théorie des ensembles et de la logique des propositions. La spécification des *SMA* à l'aide du langage *Z* a été proposée dans (Luck & Inverno, 1995) qui a défini le langage *Z* comme étant un outil offrant plus de modularité et un haut niveau d'abstraction qui est suffisamment expressive permettant une description consistante et unifiée, structurée du système cible. D'un point de vue statique, le langage *Z* permet de définir les états et les relations d'invariant qui sont préservés lors des transitions d'états. D'un point de vue dynamique, il s'intéresse aux opérations, aux relations entre les entrées et aux sorties, et les changements d'états (Gervais, 2004). En effet, la spécification s'effectue

par raffinement d'une hiérarchie de schémas *Z* composée d'entités abstraites loin de toute indication de plateformes.

L'Environnement constitue le sommet de la hiérarchie et encapsule les attributs pouvant être perçus par les Agents. Le regroupement d'un sous-ensemble de ces attributs et des actions associées forme un objet. Les actions représentent les capacités de l'objet. La troisième entité est l'Agent qui possède un état mental auquel sont associés des buts à atteindre ou des tâches à réaliser. Lorsque ces buts résultent des motivations de l'Agent, on le considère comme étant un Agent autonome représenté sous forme d'un objet avec des buts à réaliser (Hilaire, 2008). Ainsi, on parle souvent de Z comme un langage de description d'états SDL (States Description Language) en faisant recours à des notations mathématiques utilisant les propositions, la logique prédicative, les ensembles et les relations, assemblés à une notation schématique, représentant des manipulations algorithmiques de données ou des objets.

Une spécification en *Z* (**Figure 17**) contient trois parties :

- Déclarations : servent à introduire les variables,
- Expressions : servent à décrire les valeurs dont les variables peuvent assumer,
- Prédicats : servent à placer les contraintes sur les valeurs des variables que celles-ci peuvent assumer.

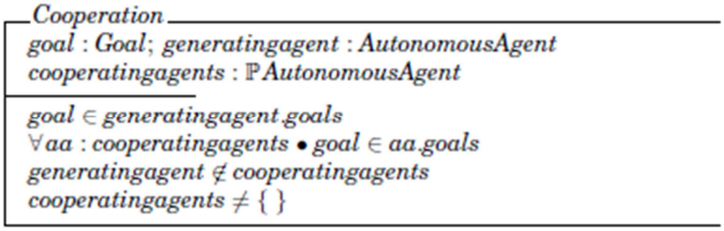


Figure 17. Schéma *Z* de Coopération

En revanche, l'inconvénient de ce langage dans notre contexte réside essentiellement dans l'incapacité à prendre en compte l'aspect réactif et la dynamique des *SMA*. Par conséquent, l'expression en *Z* des propriétés liées à la dynamique du système telles que la sureté et la vivacité est assez pénible.

2.1.4.5. Méthode B

Cette méthode a été inventée par (Abrial, 1996). Elle constitue une évolution du langage *Z*. Cette méthode est fondée sur les concepts mathématiques de la théorie des ensembles. Elle présente l'avantage d'être outillée pour l'affirmation des preuves des propriétés de vivacité. Ainsi, elle permet l'expression de propriétés sous la forme de prédicats du premier ordre. Les opérateurs auxquels on peut faire appel sont ceux de l'arithmétique et de la théorie des ensembles (Gervais, 2004). Les données sont encapsulées au sein de machines abstraites (état spécifié par une partie statique), et ne peuvent être modifiées que par des opérations définies dans la même machine. Les preuves que l'on cherche à faire en *B* sont des preuves d'invariance. Aussi, chaque machine est caractérisée par un invariant qui définit l'ensemble des valeurs que ses variables peuvent prendre. Toute machine engendre une obligation de preuve établissant la conservation de l'invariant par les opérations de la machine. Une caractéristique particulière de la méthode *B* est de permettre, par le moyen de raffinements successifs prouvés, de passer progressivement d'une spécification abstraite (indéterministe) à une spécification déterministe permettant de lever toute ambiguïté dès l'interprétation du besoin et de construire une spécification cohérente et conforme au besoin. Un autre aspect de la méthode *B* est qu'elle a été conçue pour être aisément automatisée assurant la génération des preuves mathématiques concernant l'exactitude et la légitimité des modèles (Manson, 2003). La méthode *B* est constituée des éléments suivants :

- Un langage modulaire qui couvre l'intégralité du cycle de développement,
- Un langage d'expression des propriétés et exigences de sûreté de fonctionnement en termes de fiabilité, disponibilité, maintenabilité, etc,
- La démonstration mathématique que le logiciel respecte ces propriétés,
- Le langage des composants (machines abstraites, raffinements, implantations),
- Le langage des prédicats : qui définit les propriétés que les variables doivent vérifier.

Selon (Manson, 2003), pour caractériser le fait qu'à chaque instant, un Agent est dans un état donné tant que plusieurs Agents peuvent être dans le même état, nous pouvons définir la variable état suivante :
$\text{état} \in \text{Agents} \rightarrow \text{AGENTS}$. Soit Actions la fonction définissant les actions d'un Agent (*ACTIONS*). Cette fonction désigne toutes les actions possibles. Chaque action est rattachée à un et un seul Agent si elle est instanciée, mais un Agent peut disposer de plusieurs actions. Soit *ac* une action de l'Agent *c,* on note donc : $\text{ac} \in \text{actions}\ \{\text{c}\}$.

2.1.4.6. Les Réseaux de Petri

Les fondements de la théorie des *RdP* ont vus le jour avec la thèse de *Carl Adam Petri* diffusée publiquement en *1962*. Cette théorie offre une symbolique assez simple pour décrire et étudier les systèmes qui sont à la fois complexes et communicants. Ceci permet de disposer d'un outil robuste d'analyse permettant de vérifier les propriétés que le système doit satisfaire, tout en bénéficiant d'un aspect graphique et mathématique. Le formalisme facilite la modélisation de l'interaction et de la communication entre les entités du système. Elle permet une spécification des modèles compacts et lisibles à base d'une sémantique précise offrant ainsi la possibilité de s'assurer de la justesse et de la qualité des applications qui seront ultérieurement développées (Huget, 2001). Un *RdP* est un outil montrant les relations existantes entre des conditions et des évènements. Il est évoqué sous forme d'un graphe biparti appréhendant deux sortes de nœuds: les places et les transitions qui sont reliées par des arcs. Les places sont marquées par des jetons en franchissant les transitions suivant des règles imposées par le déroulement du système.

2.1.5. Synthèse

Le langage *Z* et la méthode *B* sont des formalismes à base de principes mathématiques permettant de mettre en place une spécification formelle d'un système. Toutefois, ils ne sont pas en mesure de représenter la dynamique de comportement du système, tandis que les *RdP* qui combinent le formalisme graphique et le support visuel sont

capables de modéliser des propriétés reflétant l'aspect dynamique des systèmes telles que la synchronisation, le parallélisme, les conflits, l'exclusion mutuelle et le partage de ressources.

Grâce aux avantages que présentent les *RdP* : formalisme visuel, support mathématique, modularité, spécification hiérarchique et description facile des systèmes, on peut considérer ces outils comme les plus adaptés pour la modélisation de la dynamique des systèmes discrets. Nous allons, par la suite, nous concentrer sur les *RdP* comme étant un outil graphique puissant qui permet de comprendre facilement le système et qui permet aussi la simulation de ses activités dynamiques et concurrentes. Ce formalisme est bien adapté à la vérification formelle des *SMA* qui contient à la fois des états et des événements représentés par l'interaction entre les différents Agents. D'où, notre choix s'est centralisé sur le *RdP* qui nous garantira une modélisation exhaustive des aspects structurels et comportementaux de systèmes qui sera assimilé à des actions et des contraintes. Nous pouvons récupérer l'état d'exécution de système, la vérification des propriétés sera ainsi plus aisée. Le *RdP* est un outil convivial par son interprétation graphique, il possède une grande qualité d'analyse indéniable par sa démarche mathématique offrant un espace d'analyse structurelle.

2.2. Critères de Vérification et de Cohérence d'un SMA

La vérification formelle d'un système est l'utilisation de règles précises pour démontrer mathématiquement à l'aide d'un assistant la preuve que ce système satisfait une spécification formelle.

La vérification d'un système correspond à mesurer la justesse de construction du modèle et le respect des exigences. Elle doit aboutir à bien affirmer que le modèle est bien correct, c'est-à-dire qu'il vérifie et respecte sa spécification quelles que soient les actions de l'Environnement. Pour définir dans quel sens un système est correct, il faut définir formellement les comportements attendus et les comportements interdits de notre système. En effet, il faut garantir qu'un système aura un comportement en accord avec ce que l'on peut attendre de lui dans les conditions réelles d'exécution. Il faut s'assurer donc que le modèle respecte les lois du système et obéit aux exigences de l'Environnement. (Chapurlat, 2008) souligne qu'il faut décrire la propriété à vérifier

sous la forme d'un théorème énoncé au moyen d'une logique et de montrer qu'il peut être directement déduit de la spécification du système et des axiomes en utilisant les règles de déduction propres à la logique utilisée.

(Yeddes, Feng, & Ben Hadj-Alouane, 2009) ont proposé une technique formelle pour la vérification formelle de la sécurité des systèmes et des protocoles informatiques. En effet, deux stratégies sont proposées : la première consiste à élargir le comportement du système (ce qui lui donne plus de possibilités), et la seconde consiste à limiter le comportement du système. L'approche est destinée à être utilisé dans le cadre d'outils semi-automatiques, qui aide à la conception de systèmes, à travers des processus itératifs.

Pour n'importe quel système, le fait de vérifier la cohérence comportementale du modèle ainsi que la compatibilité entre ses entités reste un axe de recherche attirant de plus en plus l'attention des chercheurs qui essayent de bien identifier les normes avancées de la vérification.
Nous pouvons examiner et évaluer un *SMA* de deux façons : percevoir la société d'Agents en vue globale, ou en vue individuelle en se basant sur les caractéristiques d'un seul Agent dont ses propriétés influencent les propriétés du système. Selon (Giardini & Amblard, 2013), vérifier un système revient à assurer les facteurs suivants :

- *L'utilité du modèle* : avoir des résultats fonctionnels lors de la concrétisation,
- *L'exactitude*: la structure du modèle et les résultats attendus doivent être similaires,
- *Consistance*: le modèle doit être consistent en lui-même,
- *Universalité*: le modèle doit être applicable pour tous les cas,
- *Simplicité*: en cas de choix entre deux modèles, on doit avoir le modèle le moins compliqué qui est en mesure de refléter la complexité de la réalité,
- *Nouveauté*: un modèle doit créer des nouvelles informations,
- *Fiabilité, disponibilité, maintenabilité*,
- L'absence de *bug*s dans la communication entre les entités,
- *La non-contradiction* : le modèle doit respecter les relations observées,

- *Vivacité*: garantir que les meilleures propriétés restent vraies, « Some thing good will eventually happen »,
- *Sureté*: le fait de ne pas sortir de certains états dits surs, et se privilégier des autres états énoncés états d'erreur « No thing bad will be happenned »,
- *La stabilité* : le fait que le modèle soit insensible aux facteurs secondaires,
- *La fécondité*: prendre en compte les conséquences non prévues que le modèle entraîne,
- *La convergence* : la validité d'un modèle croît avec son usage,
- Un système invariant par décalage temporel est un système dont la sortie ne dépend pas explicitement du temps.

Nous identifions les propriétés ou les critères à vérifier par une technique de vérification formelle pour maintenir correctement les *SMA*:

- Encapsulation dans les Environnements,
- Autonomie des Agents,
- Interaction et Protocoles de communication des Agents,
- Mobilité des Agents.

2.3. Travaux Antérieurs de Modélisation et de Vérification des SMA

Peu de travaux de recherche sont orientés vers la création d'outils finis de modélisation et de vérification de toutes les propriétés des *SMA*. Ceci est justifié par la complexité de ce type des systèmes et la diversité des propriétés et d'entités qui les composent. Dans la littérature, les méthodes développées sont spécifiques à un critère particulier.

Dans cette section, nous présentons les techniques issues de formalisation des *SMA*. Nous pouvons identifier deux principaux axes de recherche dans ce sens :

- Méthodes pour l'étude de l'interaction des Agents,
- Méthodes pour l'étude des Agents mobiles.

2.3.1. Méthodes pour l'étude de l'interaction des Agents.

Ces méthodes concernent l'étude de communication, de coordination et de coopération. L'étude de ces trois propriétés se résume dans la formalisation de l'Interaction inter Agents. Nombreux travaux de recherche exploitent des méthodes formelles variées pour modéliser ou pour vérifier l'Interaction des Agents. Dans la suite nous présentons une liste variée et non exhaustive de ces méthodes.

(Devi & Dutta, 2012) ont proposé un outil formel appelé *CGPN* (*Conceptual Graph Petri Nets*) qui est une combinaison de *CG* (*Conceptual Graph*) et de *CPN* (*Color Petri Nets*) pour modéliser le comportement de collaboration des Agents dans un SMA afin d'atteindre certains objectifs. D'un côté, le *CG* est utilisé pour représenter les connaissances des Agents et de l'autre côté le *CPN* est utilisé pour modéliser les aspects concurrents et dynamiques d'un tel système.

Un Réseau de Petri à Graphe Conceptuel est un tuple (CG, P, T, A, N, G, E, I) :

- *CG est un graphe de conception,*
- *P est un nombre fini de Places,*
- *T est un nombre fini de Transitions,*
- *A est l'ensemble d'arcs tel que* $P \cap T = P \cap A = T \cap A = \emptyset$,
- *N est un nœud défini de A dans* $PXT \cup TXP$,
- *G est une fonction de protection définie à partir de T tel que* $\forall t \in T$ $[Type(G(t)) = B \wedge Type(Var(G(t))) = CG$,
- *E est une fonction définie sur un arc de A tel que* $\forall a \in A$ $Type(E(a)) = CG$ *et* $Type(Var(E(a))) = CG$,
- *I est une fonction d'initialisation qui mappe chaque endroit pour CG :* $I: P \rightarrow CG$,

Le modèle *CGPN* est une extension du modèle *GPN* (Vally & Courdier, 2002) qui est un modèle hybride pour modéliser la proactivité dans les Systèmes Multi Agents. Cependant, les deux modèles décrivent le comportement d'Agent par le biais d'un graphe connexe sans contrôler l'évolution dynamique de comportement

des entités du système. En fait, le nombre de nœuds croit exponentiellement si le nombre d'états (changement de comportement d'un Agent) augmente. Le processus d'interaction inter Agents est défini via une fonction E sur les arcs sous la forme CGP_i (**Figure 18**) et chaque place définit un état de processus (able, belog, perceive, req, etc.).

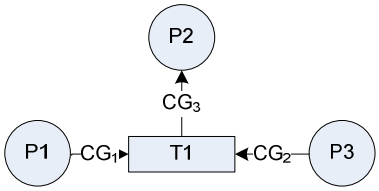


Figure 18. Réseau de Petri à Graphe Conceptuel (Vally & Courdier, 2002)

D'autres travaux de recherche modélisent l'interaction des Agents par les Réseau de Petri coloré. En effet, (Quani, Minjie, & Khin, 2004) introduisent un modèle en *CPN* pour représenter les interactions d'Agents flexibles. (Celaya, Desrochers, & Robert, 2009) utilisent les Réseaux de Petri pour modéliser l'architecture abstraite pour les Agents intelligents et l'analyse structurelle du modèle fournit une évaluation des propriétés d'interactions des Systèmes Multi Agents. La vérification de « *Deadlock* » été contrôlée dans ce modèle par l'évaluation de propriétés« *vivacité* » et « borné ». Cependant, l'analyse comportementale des Agents lors de leurs communications se fait par l'intermédiaire d'un autre sous réseau non cohérent avec le modèle principal. Ceci rend le modèle non cohérent et obéit à beaucoup de règles de transformation. Le processus d'interaction ne suit pas les standards ou les protocoles connus.

(Esterline & Rorie, 2007) ont utilisé π *–calcul* introduite par (Milner, Parrow, & Walker, 1992.) pour spécifier le Système Multi Agents *LOGOS* et (Kawabe, Mano, & Kogure, 2001) ont développé un π - *Système* à base de calcul dite *Nepi2* pour spécifier les Agents et le processus logiciel de communication.

En effet, π *–calcul* est une algèbre de processus basée sur *CCS* qui diffère de certains autres algèbres de processus dans le fait qu'elle favorise la mobilité des processus. Cela permet des liaisons de données à transmettre à d'autres processus et des liens peuvent alors être représentés par des noms de variables et par rapport à l'égalité ou l'inégalité.

Les Réseaux de Petri Coloré ont été utilisés dans (Quan, Minjie, & Haijun, 2005) pour réaliser la planification des Agents dans des Environnements dynamiques ouverts. En effet, la coordination des Agents dynamiques est un défi à résoudre dans les *SMA*. Le modèle basé sur *CPN* sert à planifier et à allouer de nouvelles tâches à l'Agent approprié ou à une combinaison d'Agents. Dans cette stratégie, grâce au *CPN* nous pouvons représenter les états dynamiques d'Agents et les coordinateurs de l'Agent. Ce modèle est en mesure de vérifier les statuts des Agents concurrents et de prendre des décisions optimales. (Weyns & Holvoet , 2002) ont présenté un modèle en *CPN* pour une application Multi Agents : Packet-World. L'approche consiste à l'intégration de nouvelles compétences sociales par l'ajout de nouveaux modules qui offrent une vue conceptuelle claire sur l'évolution des Agents et de l'Environnement. Le modèle développé est un modèle conceptuel générique pour les Agents sociaux situés dans un *SMA*. Ce modèle est constitué d'Agents qui ne peuvent interagir qu'à travers des objets passifs situés dans l'Environnement. Puisque l'interaction est le nerf central de Systèmes Multi Agents, les chercheurs ont intégré des protocoles de base pour la coordination de l'Agent dans le modèle de base. Par la suite, ils ont étendu le modèle, ce qui a permis aux Agents de communiquer des informations entre eux. En effet, la communication est la base de l'organisation sociale. Cependant, ce modèle ne peut pas exprimer clairement l'évolution comportementale des Agents après chaque action en liaison avec d'autre Agents de groupe sociale.

Dans (Ezzedine & Kolski, 2008), le *RdP* a été utilisé pour la modélisation des systèmes interactifs. L'originalité et le pouvoir des *RdP* résident dans leur capacité à visualiser le comportement de chaque Agent (actions externes ou internes), ainsi que leur capacité d'abstraction. Ceci permet de garder les mêmes informations sans aucune complexité visuelle. De plus leur aspect formel permet également d'être potentiellement efficace au moment de la phase d'évaluation et de validation. Cependant, cette technique n'offre aucun moyen de contrôler la génération automatique des sous classes du système interactif. La communication inter-Agents en cas d'un système ouvert reste difficile à vérifier.

(Mokhati, Mourad, & Badri, 2006) ont présenté un cadre formel aidant à la traduction des interactions entre Agents. Les interactions sont décrites à l'aide du formalisme

RCA (Représentation des Comportements d'Agents). Ce dernier permet de spécifier les rôles et les comportements des différents Agents (**Figure 19**). En effet, par rapport à d'autres formalismes, *RCA* permet de reconnaître la synchronisation entre les deux points de protocoles. Cependant, *RCA* n'étant pas encore doté d'une sémantique formelle, il ne permet qu'une formalisation partielle de *SMA*.

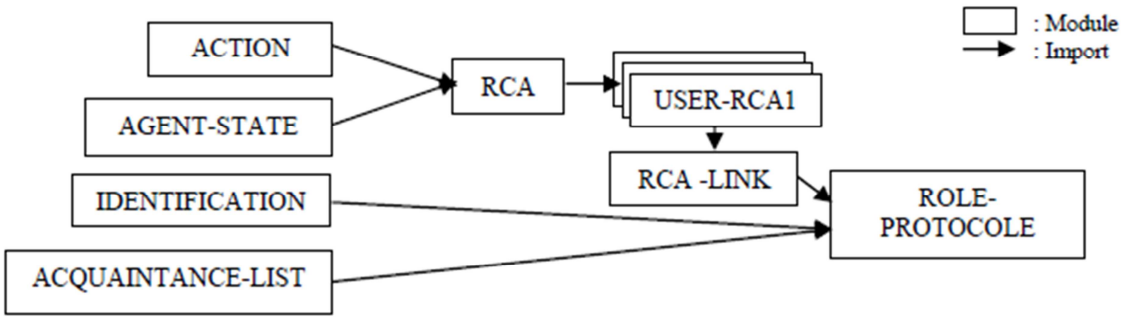


Figure 19. Architecture de Framework *RCA* (Mokhati, Mourad, & Badri, 2006)

Les conversations sont un moyen utile de structurer les interactions de communication entre les Agents. La valeur d'une approche basée sur la conversation est largement déterminée par le modèle qu'il utilise. L'interaction des Agents nécessite une notion de concurrence. (Scott, Chen, Finin, & Labrou, 1999) ont proposé l'utilisation de Réseaux de Petri colorés (**Figure 20**) comme un modèle sous-jacent pour la spécification du langue de conversation. Cette approche est relativement simple et expressive pour présenter l'aspect de concurrence. Cependant, l'intégration du Langage *KQML* (*Knowledge Query Manipulation Language*) n'est pas évidente et la vérification du processus d'Interaction nécessite à chaque étape l'exploitation de ce standard.

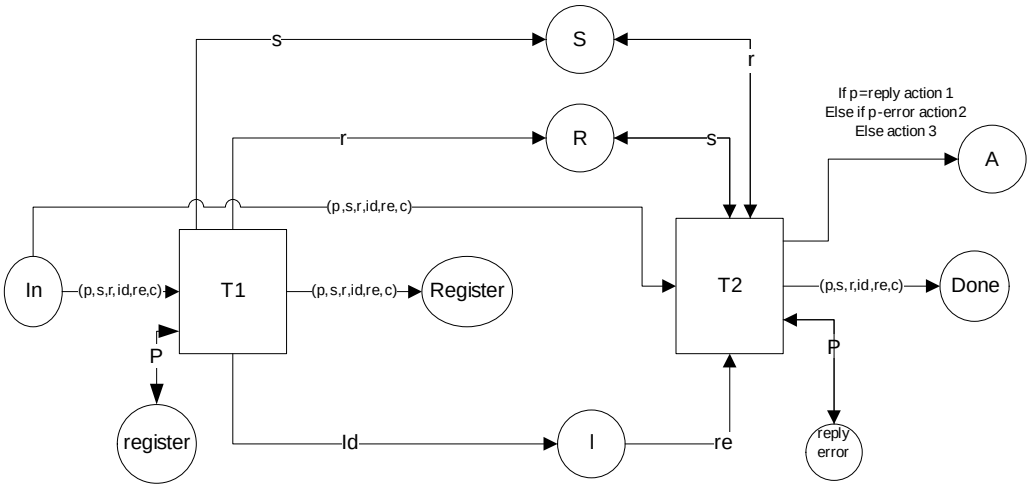


Figure 20. Modèle *CPN* de Conversation *KQML* (Scott, Chen, Finin, & Labrou, 1999)

Les Agents sont considérés comme une sorte d'extension des objets. (Odell, Parunak, & Bauer, 2000) ont développé une approche pour modéliser un protocole d'interaction (**Figure 21**) par l'intermédiaire d'*Agent Unified Model Language* (*AUML*). Cette modélisation passe par différents niveaux:

- Un niveau générique, où le protocole est représenté en utilisant des concepts du paquet et du modèle,
- Le deuxième niveau spécifie la séquence des interactions à l'aide de diagramme *AUML*,
- Le troisième décrit le traitement réalisé dans les divers Agents participant dans le protocole d'Interaction.

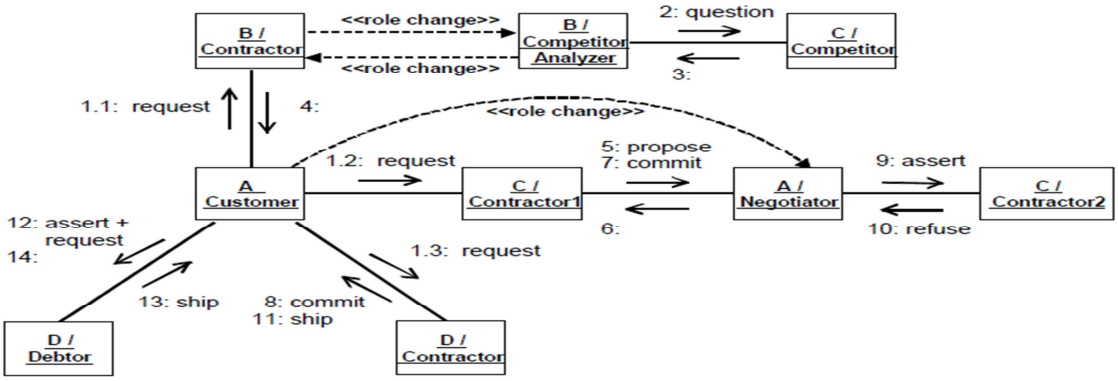


Figure 21. Diagramme de Collaboration *AUML* d'Interaction (Odell, Parunak, & Bauer, 2001)

(Kahloul, Barkaoui, & Sahnoun , 2005) ont proposé une extension des travaux de (Odell, Parunak, & Bauer, 2000) par la conception d'un nouveau modèle (**Figure 22**) à partir de laquelle nous pouvons tirer directement les principales caractéristiques des interactions dynamiques impliquées dans les *SMA*. Cette approche suit deux étapes. La première consiste à décrire les interactions de l'Agent dans un SMA en utilisant *AUML*. La deuxième étape consiste à traduire cette description *AUML* dans un modèle en Réseau de Petri Coloré Récursive (*RCPN*). Le choix du formalisme *RCPN* est motivé par sa capacité à modéliser la planification de l'Agent dynamique assurant l'entrelacement entre la planification et l'exécution des actions complexes et la disponibilité de l'analyse formelle et d'outils de vérification.

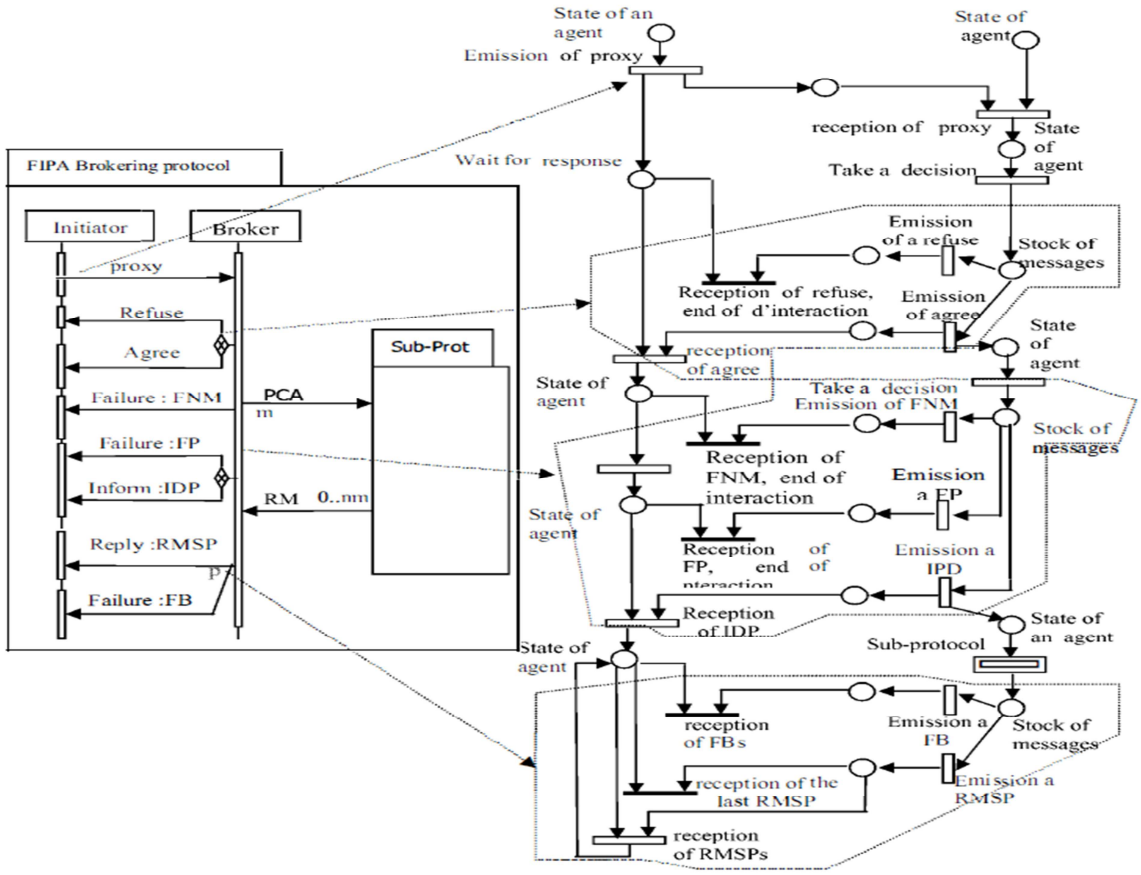
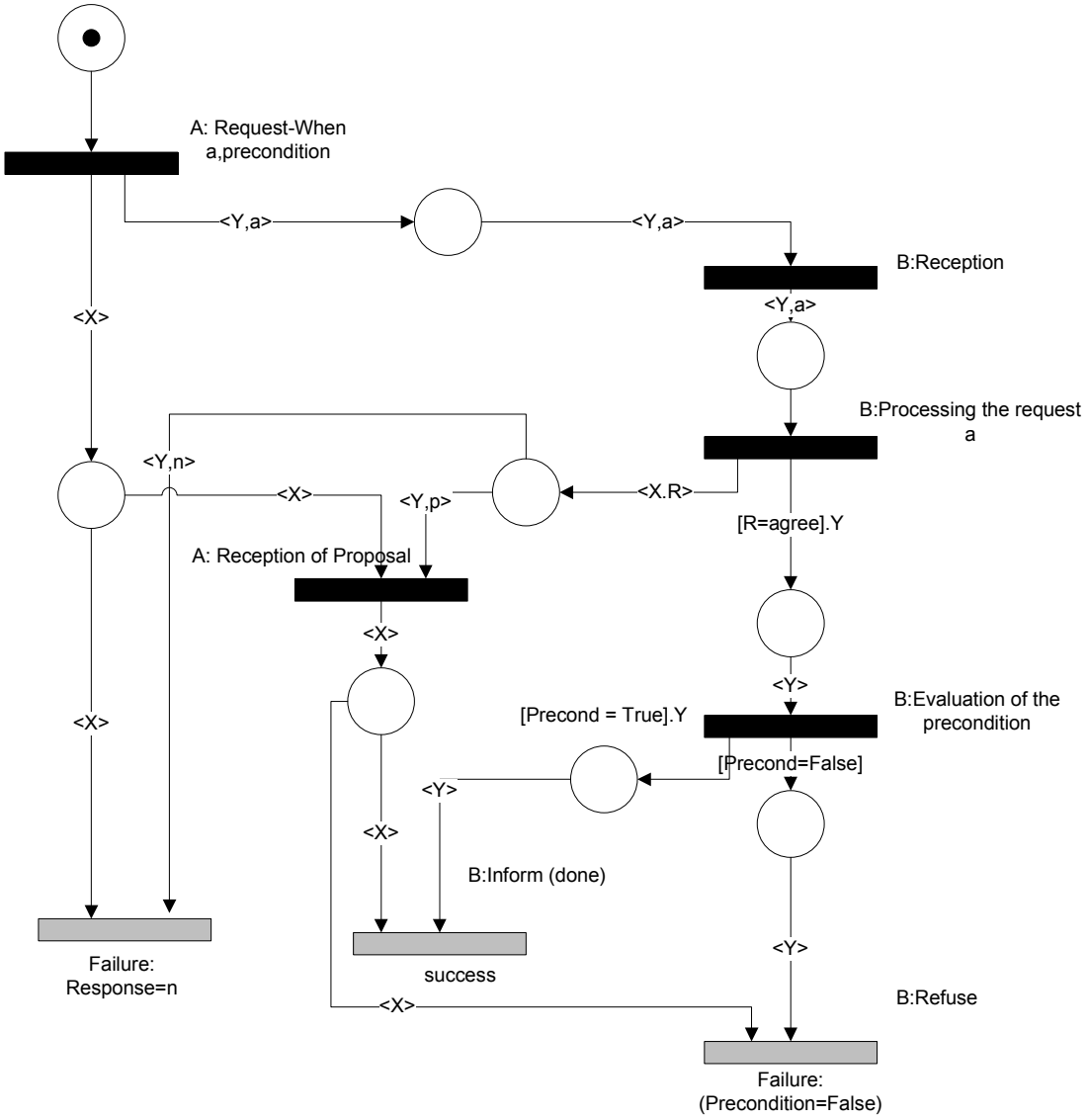


Figure 22. Protocole *FIPA* Brokering (Kahloul, Barkaoui, & Sahnoun, 2005)

Dans ce modèle, les règles de transformation et de traduction des principaux modes d'interaction entre les Agents sont présentées. Pour valider cette approche, les chercheurs ont appliqué ces règles dans la construction du modèle *RCPN* associé au *FIPA*[1] Brokering à partir de sa description *AUML*. Pour modéliser l'Interaction dans un SMA, (EI Fallahi, Haddad, & Mazouzi, 2001) ont proposé un modèle pour le Protocole *FIPA-Request* (**Figure 23**) et un autre pour le Protocole *FIPA- Conract-Net* par les Réseau de Petri Coloré.



[1]Foundation for Intelligent Physical Agents :FIPA

http://www.fipa.org

Figure 23. Modèle *CPN* pour *FIPA-Request* (El Fallahi, Haddad, & Mazouzi, 2001)

Nous affirmons, donc, qu'un modèle SMA est cohérent si la communication et l'interaction entre les Agents résultent en un système non bloquant n'aboutissant pas aux situations suivantes:

- L'exécution d'une infinité d'instructions dans un intervalle de temps borné. Le temps global du système ne peut donc plus évoluer et le système se retrouve bloqué,
- L'inter blocage: lorsque deux Agents *A1* et *A2* se retrouvent simultanément dans un état où *A1* attend une ressource ou un message émis par *A2* et inversement,
- La propriété d'invariant d'état : c'est-à-dire un Agent reste dans un état falsifié.

Synthèse

Notre étude bibliographique s'intéresse aux techniques de modélisation et de vérification formelle des propriétés fondamentales des *SMA* à savoir la communication, la coordination et la coopération. Ces trios sont la base de toute interaction inter Agents. Plusieurs travaux ont été enrichis dans ce sens. Nous constatons qu'une grande partie été consacrée à un cadre de modélisation formelle décrivant le processus de l'interaction et non au contrôle et à la vérification de ce processus. La majorité de ces travaux est basée sur les Réseaux de Petri Coloré. En effet, un modèle en *CPN* permet de bien distinguer, par la classe de couleurs, les différentes entités du système (Agents). L'échange de message entre les Agents se fait par un appel de fonctions prédéfinies (accepte, refuse, envoie, recevoir, etc.). Ces fonctions doivent obéir à des règles ou standards tel que le standards *FIPA*. Néanmoins, le nombre d'états générés pour aboutir à une interaction indirecte entre plusieurs Agents n'est pas facilement prouvé. Tous les travaux existants montrent qu'il faut établir un modèle d'Interaction entre deux Agents seulement, ensuite utiliser des règles du genre : appel récursif, règles de transformation du modèle, appel d'une autre méthode, etc.

Maintenant la question qui se pose est comment garantir la cohérence du modèle si on passe à l'échelle, si le nombre d'Agents augmente, si on ajoute dynamiquement d'autres nouveaux Agents et s'il s'agit d'une interaction indirecte ?

2.3.2. Méthodes pour l'Étude des Agents Mobiles

Dans le cadre de la modélisation et de la vérification des Systèmes Multi Agents, la mobilité des Agents est considérée comme étant un facteur de base. Elle est fondamentale pour exprimer la dynamique du déroulement des actions. Ceci est principalement dans le cas des systèmes ouverts. L'utilisation d'Agents mobiles est actuellement fortement recommandée. Il est à noter que leurs utilisations engendrent des problèmes de sécurité, de confidentialité, de gestion et de suivi. Nombreux travaux de développement des plateformes d'Agents mobiles sont commercialisés, notamment *Aglet,* Voyager, *Odyssey*, *Netlogo*, *Swarm, Mason* et *RePast.* Ces études se focalisent sur le développement d'un Environnement de développement des Agents Mobiles.

En effet, elles concernent la création des Agents, la transmission et la communication locale. Mais la mobilité se traduit par l'implémentation directe sur cet Environnement en utilisant des tâches (threads) multiples. La plupart de ces plateformes ne garantit pas la transmission du vecteur d'état des Agents avant et après la migration puisque la reprise d'architecture doit être réalisée par le concepteur (Treuil, Drogoul, & Zu, 2008). D'autres plateformes des Agents mobiles décrivent la migration d'une manière directe par un code, généralement orienté objet tels que *MADKIT, JADE*, *ZEUS* et *JNA*. Malgré que les Agents partagent certaines caractéristiques au niveau comportemental, il est difficile de transformer un modèle d'une plateforme à une autre. Ceci est dû à l'absence d'un modèle générique indépendant de l'implémentation du code.

Les travaux déjà réalisés dans le domaine des *SMA* n'ont pas considéré séparément la vérification de migration des Agents dans le cycle de développement. Dans la modélisation et la simulation de (Moldt & Weinberg, 1997), le modèle de Migration est pris comme étant un modèle prédéfini et standard pour la plateforme accueillant les Agents. Certaines plateformes d'Agents mobiles tels qu'*OMG*, *RePast* et *Masson*

fournissent le mécanisme de migration mais n'offrent pas de politique de migration. Cependant, (Bouzid, Chevrier, & Vialle, 2003) ont proposé un modèle de simulation parallèle basé sur la répartition des conflits survenant entre les Agents, et sur un équilibrage dynamique de charge entre les processeurs par le biais d'une politique dite « auto-partition dynamique ». Ceci engendre un contrôle insuffisant du comportement de l'Agent et des changements qui lui sont apportés au cours de la Migration. Récemment, des travaux font appel à des techniques de vérification pour les applications distribuées. En effet, (Bouali, 2009) a distingué deux niveaux de vérification ; le premier est structurel examinant les architectures des Agents Mobiles. Le deuxième vérifie les Environnements d'accueil des Agents. Cependant, (Chen, David, Linz, & Cheng, 2008) ont cherché à vérifier la sécurité des Agents visitant des sites différents. Le transfert de code dit « code Agent » peut subir des changements indésirables lors de la migration. Pour résoudre ce problème, les auteurs ont proposé une technique basée sur la multi-vérification côté site envoyant le code et côté client accueillant le code. (Cao, Feng, Lu, & Sajal, 2002) ont montré que la vérification de la migration est intégrée dans le processus de communication. Ceci engendre une ambiguïté pour distinguer les comportements générés localement (au sein du même Environnement) et ceux à l'extérieur (inter-Environnement). (Cauiya, 2007) a exigé un processus d'exécution continu des Agents mobiles lors des migrations vers un nouveau Environnement. Ceci limite la « liberté » pour qu'un Agent choisi son prochain Environnement et le modèle ne se construit pas ainsi en fonction des perceptions et des actions actuelles.

2.3.3. Synthèse sur les Méthodes de Modélisation et de Vérification

Les méthodes de modélisation et de vérification des Agents mobiles sont limitées. Ceci est justifié par la complexité des processus de migration de l'Agent d'un Environnement à un autre. Dans la plupart des cas, il faut contrôler l'Agent, l'Environnement de départ et l'Environnement d'arrivée. Chacune de ces entités définit un sous-système complexe. Les travaux de recherche antérieures se sont intéressés à modéliser le processus de migration en tant que scénario ou une séquence d'états. Néanmoins, l'aspect dynamique est absent dans ces méthodes. En effet, tous les processus développés sont toujours souffert de difficultés à contrôler le changement

dynamique des sites des Agents. Pendant sa migration, l'Agent obéit à des changements fréquents de son comportement liés aux nouvelles règles exigées par l'Environnement d'arrivé. Cependant, plusieurs travaux ont réussi à développer des Plateformes Multi Agents. A travers ces plateformes, on peut gérer les Agents mobiles. Ceci a été validé par de nombreux outils de simulation. En effet, à cause de l'évolution des systèmes ouverts, sa validation est toujours remise en cause. Avant de créer un Agent dans un *SMA* ou après avoir modifié certains comportements d'Agents, le concepteur veut une validation rigoureuse pour que l'Agent puisse coopérer avec d'autres sans erreurs. Donc, le moyen le plus souvent utilisé pour valider certaines propriétés des Agents mobiles est de le simuler directement dans une plateforme cible par l'exécution du modèle opératoire.

Nous confirmons qu'il est difficile de prouver que le comportement des Agents est bien contrôlé, par manque de démonstrations formelles.

2.4. Conclusion

Nous avons précisé précédemment la nécessité de diminuer la complexité des systèmes à étudier et nous avons introduit les méthodes formelles comme des outils puissants pour résoudre ce problème et satisfaire les exigences croissantes des développeurs.

La question était quelle méthode devons-nous utiliser pour modéliser le système dont on dispose. Dans notre cas, nous nous intéressons aux Systèmes Multi Agents. Grâce à l'étude bibliographique que nous avons menée, il est clair que notre orientation est plutôt penchée vers la modélisation et la vérification des *SMA* a l'aide des *RdP*. Cependant, ces outils formels sont de divers types et de diverses classes. Un autre enjeu de notre recherche est de démontrer le pouvoir expressif des *RdP* face à l'exigence de modélisation et de vérification des *SMA*. En effet, un *SMA* présente un comportement spécifique lors de sa construction et un autre comportement lors de l'évolution. Pour cela, nous nous proposons un formalisme qui peut remédier au problème de modélisation et de vérification des *SMA*.

3

Chapitre 3 : Réseau de Petri à Agents

Notre objectif est de fournir un formalisme capable de garantir une compréhension rapide, une modélisation et une vérification sûre de chaque entité ou de la totalité du Système Multi Agents. Le défi est de construire un cadre formel en se basant sur des outils graphiques et mathématiques. Ce formalisme doit être, en premier lieu, simple pour qu'il soit utilisable par des non-spécialistes. En second lieu, il doit être assez formel pour répondre aux exigences des développeurs lors de modélisations des systèmes complexes comme les Systèmes Multi Agents. En outre, il doit être flexible pour passer à un niveau de réutilisabilité souple. Par ailleurs, le développement d'une nouvelle extension des *RdP* était toujours difficile. Pour satisfaire un besoin de modélisation spécifique, certains travaux ont pris en considération l'extension de quelques types classiques des Réseaux de Petri. Pour décrire le comportement et les interactions des entités du système ou les contraintes sur ces variables caractéristiques, nous avons fait une modélisation dynamique. Notre contribution consiste à proposer un nouveau formalisme satisfaisant les exigences de modélisation et de vérification des *SMA*.

3.1. Définitions

Nous proposons une nouvelle extension des Réseaux de Petri basé sur les Agents. Ce nouveau type nous aide à résoudre les problèmes de modélisation et de vérification des *SMA*. Nous appelons ce modèle : Réseau de Petri à Agents : *RdPA*. En effet, nous présentons l'Agent par un jeton et ses actions par une transition associée par une ou par plusieurs fonctions prédéfinies.

3.1.1. Définition 1 : Réseau de Petri à Agents

D'une manière formelle, nous définissons Réseau de Petri à Agents (*RdPA*) par le 9-uplet: $Q =< P, T, A, Pré, Post, Prj, F, Ft, Envj >$ où :

> *P* : un ensemble fini non vide de Places,
>
> *T* : un ensemble fini non vide de Transitions,
>
> *A* : un ensemble fini non vide d'Agents,
>
> *Pré* : $PxT \rightarrow 2^{I \cup I_{at}}$ une application d'incidence avant,
>
> *Post* : $PxT \rightarrow 2^{I \cup I_{at}}$ ne application d'incidence arrière correspond aux arcs,
>
> *Prj* : pré condition de franchissement,
>
> *F (Ai, Aj)* : fonction relation Agent qui présente une condition de franchissement,
>
> *Ft* : fonction Agent décrivant la communication entre les Agents, *Ft (tk)= <Per, Valeur, Inter>*,
>
> *Envj* : Environnement du travail qui décrit un SMA ou un sous-SMA.

Selon notre modèle, le jeton représente un Agent ce qui n'est pas le cas dans tous les autres formalismes des *RdP*.

- I est un ensemble des Agents.
- I_{at} est construite sur I et regroupe tous les attributs de tous les Agents.
- Si *Pré* (P, T)=Ø alors ceci implique qu'il n'a pas d'arc reliant *P* à *T*.

D'un point de vue présentation graphique, un arc est étiqueté par le nom des jetons (Agents) en question. Alors que la transition est conditionnée par les trois fonctions : *Prj*, *F (Ai, Aj)* et *Ft*.

3.1.2. Définition 2 : Contrainte d'un Réseau de Petri à Agents

La définition de cette propriété décrit une condition sur un ou plusieurs jetons. La règle n'est applicable que si toutes les contraintes des jetons en question sont satisfaites par l'ensemble des jetons d'une place dans un réseau de Petri. Formellement, nous définissions une contrainte par une fonction renvoyant une valeur booléenne dont elle est définie sur l'ensemble des jetons d'une place *P*.

3.1.2.1. Définition 3 : Cardinalité

La cardinalité d'un élément Ai (Agent) dans un groupe d'éléments *Env* (ensemble des Environnements) décrit l'appartenance de cet élément, ou moins, à un sous-groupe. Nous devons vérifier que :

$$\forall\, i \in 1..|A|, \quad \sum_{j=1}^{j=|Env|} Env(Ai, Envj) = 1$$

3.1.2.2. Hypothèse d'Unicité

Lors d'exécution d'une tâche, un Agent Ai ne doit être engagé que dans un seul Environnement *Env*. Pour assurer cette unicité, nous devons toujours vérifier que *Card (Env(Ai))=1*.

3.1.3. Définition 4 : Fonction Contrainte d'un Réseau de Petri à Agents

Nous définissons une contrainte sur un Agent par la fonction booléenne:

Cont (Ai, K, j).

Cont(Ai, k, j) est définie comme étant une pré condition de franchissement d'une transition T issue d'une place *P*.

D'une manière formelle, nous définissons la contrainte sur un Agent issue d'une place P par :

$$\forall Ai \in A, \forall k \subset I, j \in J, \exists Cont(Ai, K, j) = \begin{cases} 0 & si \quad Ai \notin K \\ 1 & sin \ on \end{cases}$$

où :

I : est un ensemble de jetons d'une Place P,

J : est un ensemble de places P d'un Réseau de Petri à Agents donné,

K : est un sous ensemble de I,

j : une place appartenant au $RdPA$,

Ai : c'est un Agent d'indice i,

b : valeur booléenne (0 ou 1).

D'après la théorie de l'ensemble des parties : $\mathbf{Ai} \in \mathbf{K}$ **donc** $\{\mathbf{Ai}\} \subseteq \mathbf{K} \Rightarrow \{\mathbf{Ai}\} \in \mathbf{P(K)} \backslash \emptyset$

Soit K et I deux ensembles. On peut vérifier que K est un sous ensemble d e I par peut une fonction caractéristique $X_K : I \to \{0, 1\}$

$$X_K(I) = \begin{cases} 1 \ si \ K \subseteq I \\ 0 \ sinon \end{cases}$$

$$\forall Ai \in I \left[X_{\{Ai\}}(K) = 1 \Leftrightarrow \{Ai\} \subseteq K \Leftrightarrow Ai \in K \right]$$

$$\forall Ai \in I \left[X_{\{Ai\}}(K) = 0 \Leftrightarrow \{Ai\} \nsubseteq K \Leftrightarrow Ai \notin K \right]$$

3.1.4. Définition 5 : Fonction Pré condition

Soit $Cont\ (Ai,\ k,\ j)=b$.

Soit nk : le nombre d'éléments de sous-ensemble K.

$nk=card(K)$ *or* $nk = |k|$

Pour un nombre *nk* d'Agents qui s'engagent dans un Environnement on obtient :

$$Cont\ (A1,\ k,\ j) \wedge\ Cont\ (A2,\ k,\ j)\ \wedge \ldots \wedge Cont\ (Ank,\ k,\ j) = b$$

Ce qui donne :

$$\prod_{i=1}^{i=|k|} Cont(Ai\ ,K,j) = b$$

Nous définissons, ainsi, la fonction pré condition *Prj* issue d'une place *P* d'indice *j* par :

$$Pr\ j = \prod_{i=1}^{i=|k|} Cont(Ai\ ,K,j)$$

3.1.4.1. Interprétation des valeurs possible de Prj

La valeur booléenne renvoyée par *Prj* donne le point de départ d'une action (transition). L'engagement d'un Agent *Ai* dans un Environnement bien déterminé *Env* est précédé par le contrôle fait par cette fonction. Sous l'hypothèse d'unicité décrite ci-dessus, le sous-ensemble *nk* des Agents a une Cardinalité d'Environnement équitable qui est égale à *1* ou *0*.

- Si *Prj = 0,* alors la condition de franchissement n'est pas valide et dans ce cas il existe au moins un Agent qui n'a pas respecté le principe d'unicité, bien entendue qu'il est déjà engagé dans un autre Environnement.
- Si *Prj= 1,* alors la condition de franchissement est valide et dans ce cas on garantit que tous les Agents en question ont respecté le principe d'unicité.

3.1.4.2. Illustration

Nous supposons que :

- *Atelier1* contient les machines *M1* et *M2,*

- *Atelier2* contient les machines *M3*, *M4* et *M5*.

Nous envisageons 2 cas possibles (**Figure 24**) :

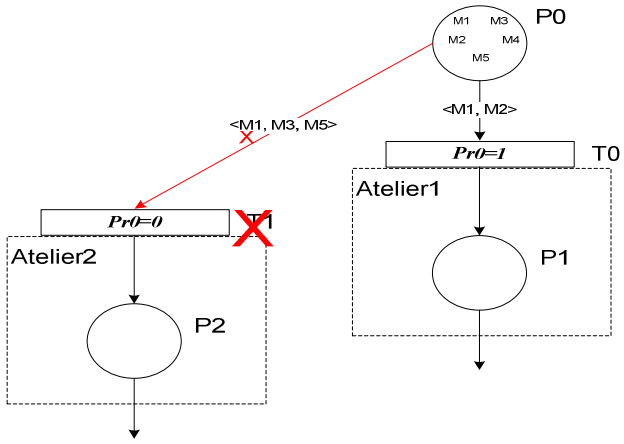


Figure 24. Illustration de la Fonction « Pré condition »

- Cas 1 : les deux machines *M1* et *M2* appartiennent au même atelier de travail (Environnement : *Atelier1*). Dans ce cas leur utilisation est permise : pré condition *Pr0=1*.

- Cas 2 : les machines *M1*, *M3*, *M4* et *M5* ne peuvent pas appartenir au même atelier de travail (Environnement : *Atelier2*) car la Machine *M1* est déjà engagée dans un autre Environnement. Dans ce cas nous ne pouvons pas franchir la transition *T1*.

3.1.5. Définition 6 : Fonction d'Appartenance (relative à un Agent)

D'une manière formelle, nous définissons la fonction d'appartenance d'un Agent Ai, à un Environnement $Envj$ noté $Apai$ par :

$$\forall Ai \in A \ \ et \ \ \forall \ \ Envj \subset Env, Apai = Apa(Ai, Envj, b, di); \text{ avec } di \in Z^{+}$$

Où :

> A : un ensemble fini non vide d'Agents,
>
> $Envj$: Environnement du travail qui décrit un ou un sous Système Multi Agents,
>
> b : contrainte / $Prj{=}b$, définit l'engagement de Ai dans $Envj$,
>
> d : degré d'appartenance, il donne le nombre de fois que l'Agent Ai a été engagé dans $Envj$.

di se calcule comme suit : $di = \sum_{J=1}^{j=|Env|} card(Envj(Ai))$

Cette fonction explique la relation entre un Agent et son Environnement. L'engagement d'un Agent Ai dans un Environnement $Envj$ décrit en premier lieu une relation d'appartenance et le nombre de fois que cette Ai a été engagé dans $Envj$. Ceci offre plus de clarté et minimise les difficultés avec les tâches qui nécessitent une connaissance du monde $Envj$. Nous pouvons ainsi, obtenir des informations supplémentaires lors de l'absence de mécanisme de mémorisation ou de perception. Nous montrons qu'un Agent cognitif a la capacité de raisonner sur des représentations du monde, de mémoriser des situations, de les analyser, de prévoir des réactions possibles à toute action, d'en tirer des conduites pour les évènements futurs et donc de planifier son propre comportement.

3.1.5.1. Interprétation de la fonction d'appartenance

Cette fonction donne une description de la relation entre l'Agent et un Environnement bien déterminé. Ceci garantie la mise à jour de sa base des connaissances. La réaction d'un Agent dépend de son Environnement. L'évolution de Réseau de Petri à Agents dépend du système à étudier. Ce qui implique, implicitement, que chaque Agent cherche à enrichir ses compétences. Pour cela il doit interpréter, partialement, la valeur de d.

3.1.5.2. Illustration

Reprenons l'exemple de la section 3.1.4.2 avec la possibilité de franchir $T2$. Nous obtenons la **Figure 25**.

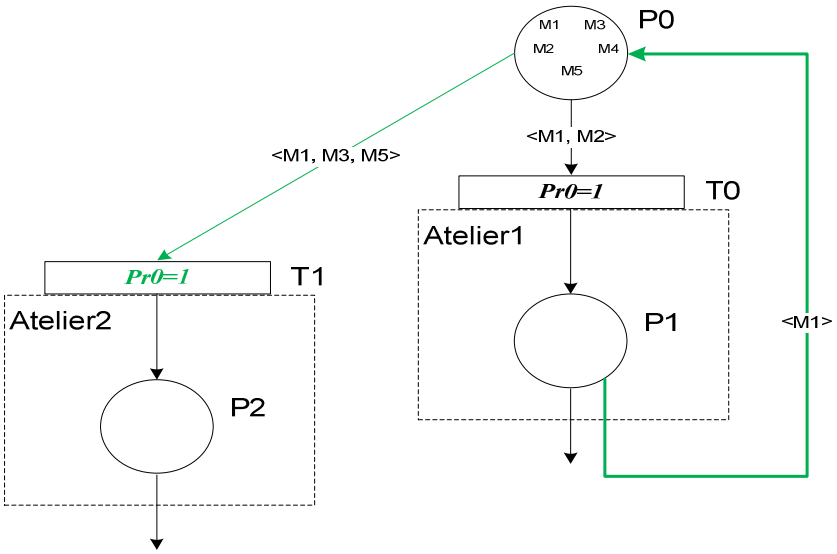


Figure 25. Illustration de la Fonction « Appartenance »

- Avant le franchissement de *T0* et *T1*, l'Agent *M1* admet comme degré d'appartenance *d=0* respectivement pour l'Environnement *Atelier1* et *Atelier2* :
  - *ApM1=Apa (M1, Atelier1, 1, 0)*
  - *ApM1=Apa (M1, Atelier2, 1, 0)*
- Après le franchissement de *T0*, l'Agent *M1* aura comme degré d'appartenance *d=1* pour l'Environnement *Atelier1* et d=*0* pour l'Environnement *Atelier2* :
  - *ApM1=Apa (M1, Atelier1, 1, 1)*
  - *ApM1=Apa (M1, Atelier2, 1, 0)*
- Après le franchissement de *T1*, l'Agent *M1* aura comme degré d'appartenance *d=1* respectivement pour l'Environnement *Atelier1* et *Atelier2* :
  - *ApM1=Apa (M1, Atelier1, 1, 1)*
  - *ApM1=Apa (M1, Atelier2, 1, 1)*

3.1.6. Définition 7 : Fonction d'Appartenance (relative à un Environnement)

La création de la fonction d'appartenance *Apai* d'un Agent *Ai* à un Environnement *Envi* nous permet de proposer, inversement, une fonction d'appartenance *Apej*. Cette nouvelle fonction décrit l'ensemble des Agents qui appartiennent au même Environnement *Envj* avec chacun son degré d'appartenance *di*.

Nous définissons la fonction d'appartenance relative à un Environnement *Envj* par :

$$Apej = Ape(Envj\ ,\ \bigcup_{i=1}^{i=nk}(Ai\ ,di\ ))$$

où :

nk : nombre d'Agents engagés dans un Environnement *Envj,*

Ai : Agent d'indice *i*,

di : degré d'appartenance de l'Agent *Ai*.

3.1.6.1. Interprétation de la fonction d'appartenance

L'application *Apej* → *di* est bijective. Chaque Agent *Ai* admet un et un seul degré d'appartenance *di* pour l'Environnement *Envj*.

3.1.6.2. Illustration

A partir d'illustration de la section 3.1.5.2 nous pouvons déduire le degré d'appartenance de chaque Machine *Mi* dans son Environnement *Atelierj* :

$$Ape(Atelier1) = Ape\ (Atelier1, \bigcup_{i=1}^{i=5}(d_i)\ )$$

$$Ape(Atelier2) = Ape\ (Atelier2, \bigcup_{i=1}^{i=5}(d_i)\ )$$

Nous déduisons, ainsi, la matrice d'appartenance suivante :

$$\begin{array}{c} \\ Atelier\ 1 \\ Atelier\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} M1 & M2 & M3 & M4 & M5 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Cette matrice représente le degré d'appartenance de chaque Machine *Mi* dans l'atelier de travail concerné.

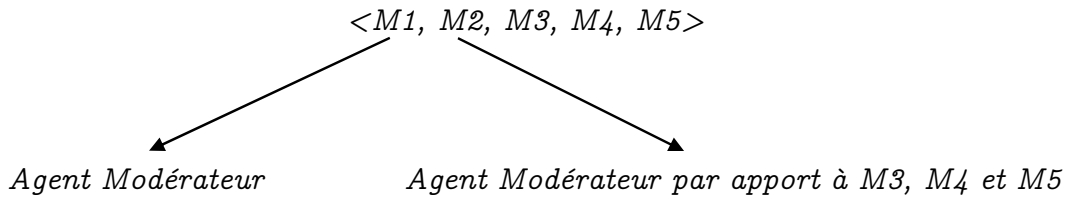
3.1.7. Définition 8 : Agent Modérateur

Un Agent *Ai* est dit « *Modérateur* » s'il est prioritaire par rapport à un autre. Le terme prioritaire indique que l'Agent modérateur domine lors d'une communication, ou encore il possède un degré hiérarchique (*dh*) moins élevé (*dh=2* domine *dh=3*).

Lors du lancement d'un processus de communication ou d'interaction. Un Agent modérateur est celui qui commence la conversation, l'envoi de requête ou de message.

Un Agent est dit *Modérateur total* s'il est prioritaire par apport à tous les Agents de son Environnement. Il possède un degré hiérarchique *dh* qui est égale *1*.

Illustration



3.1.8. Définition 9 : Fonction de Relation Agent d'ordre 2

Nous définissions une relation d'ordre *2* comme étant une fonction admettant deux entrées *Ai* et *Aj,* ainsi qu'une seule valeur booléenne *b* en sortie. L'entré *Ai* désigne, impérativement, un Agent modérateur. Cette fonction présente une pré-condition de franchissement d'une transition.

Donc, nous définissons cette relation par la fonction *F (Ai, Aj)=b*.

Soit deux Agents Ai et Aj de même Environnement Env.

$$\forall Ai \in A \text{ et } \forall Aj \in A, \exists F(Ai, Aj) = b$$

$$F : A \times A \rightarrow \{0,1\}$$

Où :

A : l'ensemble des Agents de l'Environnement,

Ai : un Agent modérateur,

Aj : un Agent non modérateur,

b : valeur booléenne renvoyant 1 ou 0.

Ainsi, nous pouvons généraliser cette fonction pour obtenir une fonction de relation Agent d'ordre n :

$$\forall Ai \in A, \exists F(Ai, Aj, \ldots, An) = b$$

$$F : A^2 \rightarrow \{0,1\}$$

3.1.8.1. Interprétation des valeurs possibles de b

- Si $b=0$ alors aucune communication ne peut être établie entre les deux Agents. Dans ce cas, l'Agent non modérateur Aj ne peut pas entrer en communication avec l'Agent modérateur Ai soit volontairement, soit forcé par un ordre de l'Agent modérateur d'ordre total, ou encore parce qu'il est déjà occupé par une autre conversation.
- Si $b=1$ alors une communication peut être établie entre les deux Agents en question. Dans ce cas, l'Agent modérateur demande le lancement d'une conversation avec un autre Agent qu'on a appelé non modérateur. Ce dernier accepte cette demande.

3.1.8.2. Illustration

La **Figure 26** présente un exemple d'utilisation de la fonction Relation Agent d'ordre 2. Ici, M1 est l'Agent Modérateur.

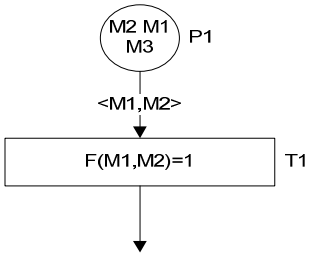


Figure 26. Illustration de la Fonction « Relation Agent d'ordre 2 »

3.1.9. Définition 10 : Fonction Agent

Nous définissons une fonction *Agent* comme étant une validation de relation entre deux Agents qui se communiquent. Elle décrit la phase de perception, la référence au contenu de message et la réussite de transfert de ce message. Ceci est présenté, respectivement, par les trois champs : *Perception*, *Valeur* et *Interprétation*. Cette fonction est relative à une transition *tk*. Elle décrit une action ou encore l'objectif de la conversation. Cette action met en jeu deux Agents qui échangent des messages (données ou variable). On note la fonction Agent par *Ft (tk)*.

D'une manière formelle, nous définissons la fonction *Agent* comme suit :

$$\forall tk \in T, \exists\ Ft(tk) =< Per,\ Valeur,\ Inter >$$

où :

tk : une transition d'indice k,

T : un ensemble fini de transitions d'un *RdPA*,

Per : Perception renvoyant un booléen,

Valeur : Donnés échangés ou tâches réalisés par les deux Agents en questions,

Inter : Interprétation de message, c'est une valeur booléenne.

3.1.9.1. Interprétation des valeurs possibles de Per

Per est une valeur booléenne qui met en relation deux Agents Ai et Aj à travers une transition T.

- Si *Per =0* alors l'Agent Ai n'a pas envoyé des données à l'Agent Aj,
- Si *Per =1* alors l'Agent Ai a envoyé des données à l'Agent Aj.

En fait, l'envoie de données consiste en un échange de n'importe quel type d'information par les deux Agents en question.

3.1.9.2. Interprétation des valeurs possibles d'Inter

Inter est une valeur booléenne qui valide la transmission ou l'échange de données entre deux Agents Ai et Aj à travers une transition T.

- Si *Inter =0* alors l'Agent Aj n'a pas encore reçu des données de l'Agent Aj ou il a refusé la réalisation des tâches demandées,
- Si *Inter =1* alors l'Agent Ai a bien reçu les données envoyés par l'Agent Aj. En effet, la tâche a été réalisée. D'où c'est une valeur de validation.

3.1.9.3. Interprétation des valeurs possibles de Valeur

Valeur est définie comme étant une action réalisée par les deux Agents Ai et Aj. Cette action représente un transfert de données de l'Agent modérateur Ai vers l'Agent non modérateur Aj. Il s'agit d'une structure de données allant du plus simple (entier, caractère, booléen, etc.) au plus complexe (tableau, matrice, objet, classe d'objet, composant, Agent, etc.). *Valeur* modifie le comportement d'un Agent et même sa structure.

3.1.9.4. Interprétation des valeurs possibles de Fonction Agent Ft

La fonction Agent *Ft* décrit la relation entre deux Agents communicants. L'échange des données et le comportement de chacun d'eux. Elle modifie directement les valeurs issues de deux Agents en question. Ceci défini la capacité de percevoir et de réagir aux modifications réalisées dans leurs Environnements.

- Initialement, $Ft\ (tk) = <0, \Phi, 0>$ ceci implique qu'il n'y a aucune interaction entre les Agents. Si la valeur de $Per = 0$ alors directement on aura $Inter = 0$. Nous ne pouvons jamais avoir $Per = 0$ et $Inter = 1$. $Valeur = \Phi$. Dans ce cas aucune action n'est déclenchée et nous gardons la situation précédente des Agents.

- Au cours du franchissement de la transition *tk*, il y'aura échange de données entre les Agents. Dans ce cas, *Per* prend la valeur *1*, *Inter* prend la valeur *0* et *Valeur* défini l'action ou la tâche à réaliser. La relation d'ordre déjà définie dans la section 3.1.8 donne le sens de transfert des données (de l'Agent Modérateur vers les autres Agents non modérateurs). D'où :

$$Ft\ (tk) = <1,\ Valeur,\ 0>$$

- Après le franchissement de la transition *tk*, *Inter* prend la valeur *1*, ceci indique que l'action a été réalisée avec succès. D'où :

$$Ft\ (tk) = <1,\ Valeur,\ 1>$$

3.1.9.5. Illustration

La **Figure 27** présente un exemple qui décrit la communication et l'échange de données entre les deux machines *M1* et *M2*.

- Etat initial de traitement : les deux machines *M1* et *M2* sont en attente : $Per=0,\ Valeur=\Phi$ et $Inter=0$.

- La machine *M2* veut faire passer la pièce *P* pour être traitée par *M1* :

Per =1, Valeur =M2. Passe [P] et Inter=0.

- La machine *M1* accepte et valide la demande de *M2* :

  Per =1, Valeur =M2. Passe [P] et Inter=1.

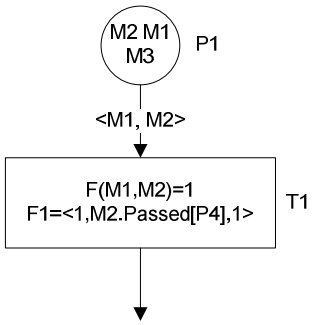


Figure 27. Illustration de la Fonction « Agent »

3.2. Nouvelle Architecture d'un Agent

Les définitions que nous avons formulé au niveau des sections précédentes nous mènent à définir une nouvelle architecture d'Agent. Dans un Réseau de Petri à Agents, chaque jeton (Agent) doit être doté d'un certain nombre de propriétés et de caractéristiques. En respectant l'autonomie, l'intelligence, la réactivité, l'hétérogénéité et la sociabilité, nous proposons, un nouveau modèle et une nouvelle architecture d'Agent cognitif. Ce modèle ne présente pas seulement la structure interne d'Agent *Ai* mais aussi son emplacement dans un Environnement *Envj*, son comportement, ses liens avec les autres Agents, son historique d'actions et son degré d'appartenance.

L'exploitation de ce modèle nous permet de décrire et d'analyser facilement le fonctionnement des Systèmes Multi Agents.

Nous définissions une nouvelle architecture : *Architecture Agent Petri (AAP).*

D'une manière claire, Nous définissons l'Architecture Agent Petri par le modèle suivant :

ident			
Envj	*d*	*df*	
Pk		*Pp*	
Ar		*F*	
t	*Per*	*Valeur*	*Inter*
Historique			

Tableau 5. Architecture Agent Petri (AAP)

ident : indice de l'Agent en question,

Envj : nom de l'Environnement,

d: degré d'appartenance de l'Agent Ai dans son Environnement *Envj*,

df : nombre de fois de franchissement de l'Agent Ai; cette valeur sera incrémentée après chaque franchissement,

Pk : place actuelle de l'Agent,

Pp: ensemble de places déjà visité par l'Agent Ai,

Ar : ensemble d'Agents qui sont en cours de communication avec Ai,

F : valeur retournée par la fonction relation Agent,

t : numéro de transition qui est en cours de franchissement,

Per : Valeur de *Per* de la fonction Agent *Ft*,

Valeur : contenue de valeur de la fonction Agent ; elle décrit l'action à réaliser,

Inter: valeur d'*Inter* de la fonction Agent.

Ce modèle offre la possibilité de décrire instantanément le comportement d'un Agent tout en indiquant les actions en cours de réalisation ainsi que leur historique. L'importance majeure de ce modèle est au niveau applicatif ou implémentation. Nous

pouvons interpréter facilement le comportement d'un Agent par analyse fonctionnelle de son architecture *AAP*.

3.3. Sémantique de RdPA

3.3.1. Marquage d'un RdPA

Un marquage d'un Réseau de Petri à Agents *R* est une famille indexée par *P*. C'est un vecteur colonne dont la valeur de $i^{ème}$ composante est le nombre de marques dans la place *pi* à un instant donné. Un Réseau de Petri à Agents marqué est un couple *<R, M0>* où *R* est un Réseau de Petri et *M0* est un marquage de *R* appelé marquage initial. Une place *P* est une pré-condition d'une transition *t* s'il existe un arc orienté de *p* vers *t*. Symétriquement, *p* sera une post condition de *t* s'il existe un arc reliant *t* à *p*.

Le marquage d'un Réseau de Petri à Agents évolue à chaque activation d'une transition. Un tel événement est régi par des règles de franchissement. Une transition ne peut être activée que si le marquage de l'ensemble des places pré condition l'autorise. A l'activation d'une transition, il y a consommation de nombre de marques adéquat dans les places pré condition et production de marques dans les places post condition.

Si le marquage *Mi* est accessible à partir du marquage *M0* après franchissement de la séquence de transition *S=(S1, S2, …, Si), alors* $Mi=M0+C\bar{S}$

Où *S* est le vecteur de taille *n* dans lequel chaque *Sj* représente le nombre de fois où la transition *tj* est franchie dans la séquence *S*.

Dans le Réseau de Petri à Agents, le franchissement d'une transition implique le changement de comportement des Agents et de l'Environnement. Une séquence de franchissements donne d'une manière informelle l'historique de relations entre deux ou plusieurs Agents. L'intelligence d'un tel Agent se base donc sur sa capacité d'interpréter cette séquence. En effet, à la prochaine transition l'Agent doit mettre en considération les transitions visitées c'est à dire les actions déjà réalisées. Donc, le marquage d'un *RdPA* présente une description dynamique des Agents dans un *SMA* et une aide à la prise de décisions.

3.3.2. Règles de Franchissement d'un RdPA

Une transition ne peut être activée que si le marquage de l'ensemble des places Pré l'autorise. Si *t* est franchissable pour le marquage *M*, le franchissement (tir) de *t* donne le nouveau marquage *M'* tel que :

$$\forall p \in P,\ M'(p) = M(p) - Prè(p, t) + Post(p, t)$$

Cette condition constitue la première règle. Nous utilisons également les notations :

$$M' = M - Pre(\ , t) + Post(., t)$$

$$M\ (t > M')$$

$$M\ (t) \rightarrow M'$$

Ainsi, soit *R* un Réseau de Petri à Agent, *M* un marquage de *R*, on dit qu'une transition *t* de *R* est franchissable à partir de *M* pour l'Agent *Ai* de *C(t)* si et seulement si :

$$\forall p \in P, M(p) \geq Pr\acute{e}(p,t)(A_i, A_j, \ldots, A_n)$$

- On note *M [t (Ai, Aj, …, An)>* le franchissement de *t* à partir de *M* pour l'ensemble d'Agents *Ai, Aj, …, An.*
- Si *t* est franchissable à partir de *M* pour l'ensemble d'Agents *Ai, Aj, …, An*, le marquage *M'* est obtenu par ce tir est défini par :

$$\forall p \in P, M'(p) = M(p) + Post(p,t)(A_i, A_j, \ldots, A_n) - Pr\acute{e}(p,t)(A_i, A_j, \ldots, A_n)$$

- On note *M [t (Ai., Aj…, An)> M'* le franchissement de *t* à partir de *M* pour l'ensemble d'Agents *Ai, Aj, …, An.*
- Si, à partir d'un marquage *M*, plusieurs transitions sont candidates, n'importe quelle transition d'entre elles peut être activée.

Chaque transition associe une ou plusieurs conditions faisant intervenir des variables formelles associées aux fonctions correspondantes. Ces conditions sont manipulées par la fonction *Prj* pour la première transition, c'est-à-dire que le tir de *t0* dans un *RdPA* est conditionné par *Prj=1*. Donc ceci constitue la deuxième règle de franchissement.

Une transition *ti* est franchissable si et seulement si la fonction de relation Agent d'ordre *n* l'autorise, dont *F (Ai, Aj)* doit être égale à *1*. La valeur booléenne renvoyée par cette fonction représente, si elle est validée, la règle numéro trois.

Les trois règles déjà données constituent, principalement, les règles à suivre pour franchir une transition. L'apport majeur apporté par ses règles est de contrôler d'une manière efficace les relations entre les Agents, les actions à réaliser dans un Environnement et l'évolution du système à modéliser.

- Il y a toujours un initiateur d'une conversation. C'est celui qui a commencé une conversation en envoyant le tout premier message. Un marquage initial est défini de telle sorte que le rôle initiateur puisse toujours franchir la toute première transition. Dans notre cas, une conversation est initiée par une demande de connexion entre un Agent émetteur et un autre récepteur.
- Tous les rôles ont des transitions de terminaison soit attribuées à un seul Agent, soit communes. Ces transitions déterminent tous les cas possibles de fin de l'interaction (échec, succès, etc.)
- On associe à chaque transition une fonction étiquetée renseignant sur les actions exécutées dans le *SMA* (envoi, réception de message, etc.) ou les actions locales aux Agents pour apporter une sémantique supplémentaire à notre modèle.
- Un Agent veut communiquer avec un autre, il lui envoie une demande de connexion. Ce dernier répond positivement en s'engageant dans le même Environnement que le premier. Il répond négativement s'il est déjà engagé dans un autre Environnement ou ne pouvant pas entrer en communication avec celui-ci.

3.4. Réseau de Petri à Agents à partir des Exemples

Pour mieux comprendre le fonctionnement de *RdPA*, nous présentons deux exemples qui valident notre approche. Ces exemples allant du plus simple au plus complexe. Le premier exemple consiste à modéliser la tâche d'impression. Tandis que le deuxième est celui de modélisation de l'algorithme de placement dynamique des tâches sur des machines parallèles (processeurs).

3.4.1. Exemple 1 : Modélisation d'une Tâche d'Impression

Le système est composé par les deux Agents un *CPU* et une Imprimante. Nous nous proposons de modéliser ce système par deux types de *RdP* à savoir le *RdP PT* et notre *RdPA*. La **Figure 28** présente une modélisation *par un RdP* Place/ Transition.

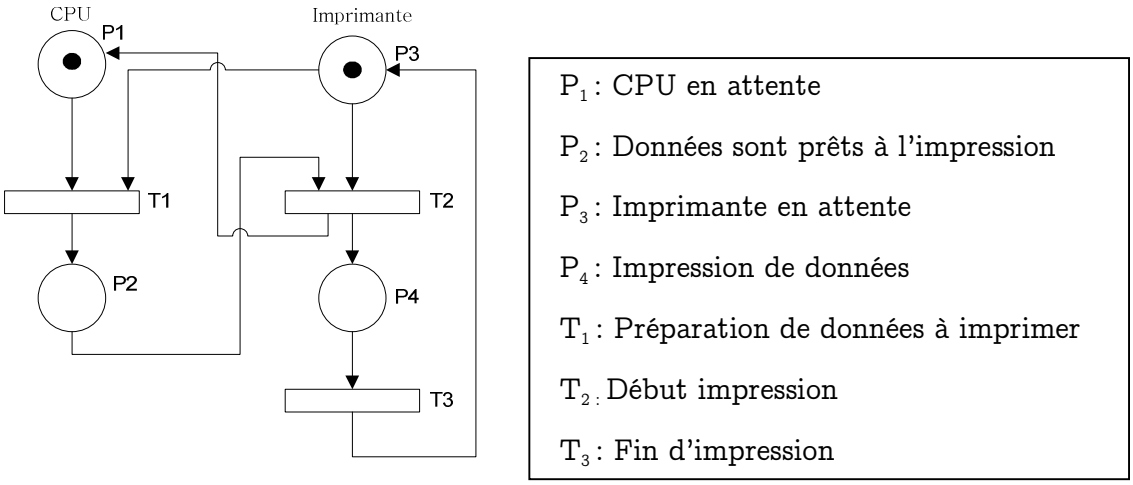


Figure 28. Modélisation de la Tâche d'Impression par *un RdP PT*

Par ailleurs, la **Figure 29** présente une modélisation en *RdPA* du même système d'impression.

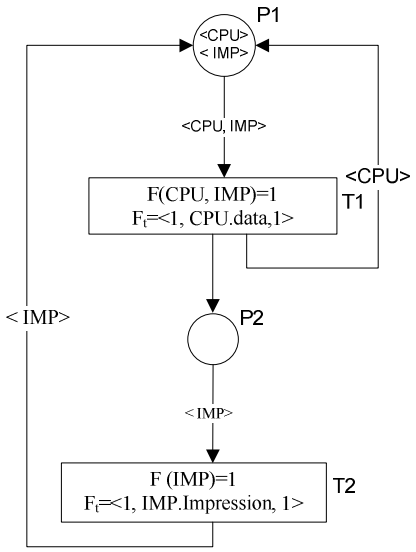


Figure 29. Modélisation de la Tâche d'Impression par un *RdPA*

Il y a réduction (pliage) au niveau de la taille du Réseau de Petri. En effet le nombre de places devient *2* au lieu de *4* et le nombre de transitions devient *2* au lieu de *3*. Le marquage initiale *M0* du premier cas est *M0= (1, 1, 0, 0)*. Pour le deuxième cas *M0= (2, 0)*. Ceci implique aussi qu'on peut obtenir un arbre de marquage réduit.

Cette dernière information permet d'aboutir à une vérification précise des propriétés du modèle (Vivacité, borné, etc.). Ainsi, le modèle en *RdPA* décrit d'une manière intelligente l'état et le comportement du système avec une présentation graphique simple.

Le modèle en *RdPA* est plus expressif que celui en *RdP PT*. En effet, nous distinguons, clairement et sans ambiguïté, les différentes entités du système (Agent *CPU* et Agent Imprimante). Les transitions sont étriquées par les fonctions nécessaires pour franchir les transitions.

3.4.2. Exemple 2 : Algorithme de Placement Dynamique des Tâches

Face aux inconvénients d'un algorithme centralisé, où chaque processeur perd son autonomie à cause de l'existence d'un processeur central chargé de la gestion des tâches. Nous pensons qu'un algorithme entièrement distribué est plus adapté à ces architectures avec sa grande résistance aux pannes et aux reconfigurations du système. Les processeurs jouent le même rôle en exécutant le même processus d'allocation ce qui permet de bien exploiter le parallélisme.

Le contrôle (la détermination d'un processeur d'exécuter la tâche) sera pris en charge par la coopération des processeurs situés sur chaque processeur du réseau.

D'après (Jaillet & Krajecki , 2007), nous pouvons distinguer trois heuristiques distribuées dans l'approche coopérative pour le transfert des tâches : la première applique des politiques à l'initiative de l'émetteur « forward ». La seconde applique des politiques à l'initiative du destinataire «reverse», où un processeur interroge régulièrement les autres processeurs. Tandis que la troisième est dite « hybride », qui est un compromis entre les deux algorithmes précédents.

Parmi ces algorithmes distribués, nous avons choisi l'algorithme « forward » pour le décrire, en premier lieu, par un Système Multi Agents. Puis, en second lieu, nous nous le modélisons par le $RdPA$.

Selon (Leinberger, Karypis, Kumar, & Biswas, 2000) et (Mancini & Petit, 2010), l'algorithme « forward » donne des performances uniformément meilleures quand la charge globale du système (taux de surcharge) est petite ou moyenne. Ceci s'explique par le fait qu'en cas de grande charge, le nœud surchargé perd du temps à chercher sans succès des nœuds faiblement chargés. Alors que les algorithmes « reverse » sont préférables quand la charge est grande, parce que si la charge globale devient petite, les nœuds faiblement chargés deviennent nombreux et inondent le système de demande de tâches.

Les deux modules qui composent cet algorithme sont :

- Un module information,
- Un module décision.

3.4.2.1. Module Information

- Nous estimons la charge locale d'un processeur par son nombre de tâches activées (Nc). Chaque processeur maintient sa charge locale avec un coût réduit de calcul puisqu'on peut l'évaluer rapidement.
- Pour l'évaluation de charge locale d'un processeur, nous utilisons les deux seuils Sc et Uc avec ($Uc < Sc$), si Nc est supérieure à Sc alors le processeur est considéré comme surchargé et dans le cas où Nc est inférieur à Uc alors le processeur est supposé non chargé. Il est supposé indéterminé si Nc est compris entre Uc et Sc.
- Afin de minimiser le coût de communication entre les processeurs et profiter de mémoire propre de processeur, chacun d'eux est constitué de trois listes triées:
    - S_LISTE: Liste des processeurs surchargés. C'est l'ensemble des processeurs tels que $Nc>Sc$.
    - U_LISTE: Liste des processeurs non chargés. C'est l'ensemble des processeurs tels que $Nc<Uc$.

- I_LISTE: Liste des processeurs qui n'appartiennent ni à la première ni à la seconde liste. C'est l'ensemble des processeurs tels que $Uc<Nc<Sc$.

- Pour l'échange des informations d'états entre les processeurs, nous appliquons l'approche d'échange relatif où chaque processeur informe les autres de son état, s'il y a changement dans celui-ci.

3.4.2.2. Module Décision

Pour effectuer le placement dynamique des tâches, nous utilisons l'algorithme "forward" qui est activé chaque fois qu'une tâche Ti est prête pour l'exécution. Si le nombre de tâches activées (Nc) du processeur en question (Pc) est supérieur à un seuil Sc (Pc est surchargé), l'algorithme procède de la manière suivante:

1. Une demande de transfert est envoyée au premier processeur de la liste $U\_LISTEc : Pq$.
2. Pq répond par "un accusé de réception" représentant son état courant (surchargé, non chargé ou indéterminé) et met à jour des listes en fonction du nouvel état de Pc.
3. Si cette réponse est favorable (Pq est toujours non chargé), alors la tâche est transférée vers Pq et les listes S_LISTE, U_LISTE et I_LISTE sont mises à jour en fonction du nouvel état de Pq.
4. Dans le cas contraire, aller à (1) en passant au deuxième élément de la liste U_LISTEc et répéter la même procédure.

3.4.3. Caractéristiques Comportementales de Processeur

Nous pouvons montrer qu'un processeur peut être vu comme étant un Agent Cognitif. Les caractéristiques peuvent être divisées en deux classes : comportementale et dynamique.

- Il effectue, gère et contrôle la totalité des procédures et des enchaînements, même si parfois il délègue certaines tâches à des processeurs spécialisés:
  - un processeur exécute la tâche la plus prioritaire dès qu'elle est prête et qu'il ne peut pas retarder s'il n'a rien d'autre à faire: *Réactif*.

- le processeur contrôle son état interne et perçoit son Environnement. Il ne reste pas inactif s'il existe une tâche prête à être exécutée: *Autonome*.
- Il a la possibilité d'effectuer des calculs pendant qu'il communique. Il est possible d'exécuter des tâches pendant l'envoi et la réception des messages : *flexible*.
- Le processeur assure le traitement et l'exécution des instructions d'un programme de plus en plus complexe: *Intelligent*.

– Le processeur prend en compte les types des tâches (sa date de réveil, son délai critique, sa période d'activation, etc.) et les contraintes sur l'Environnement d'ordonnancement (par exemple si les priorités sont fixes /dynamiques au niveau des tâches, si les préemptions sont admises, etc.). Il doit satisfaire toutes les échéances : *Proactif*.

– Il assure la communication de ces tâches avec les tâches placées sur les autres processeurs. Il interagit avec les autres processeurs quand la situation l'exige afin de réaliser ses tâches: *Social*.

Donc, nous pouvons déduire les caractéristiques suivantes :

– *Réactif* : le premier processeur de la liste non chargée *Pq* répond par un accusé de réception indiquant son état actuel.

– *Autonome* : le processeur contrôle son état interne (surchargé ou bien non chargé), il teste son nombre des tâches activées *Nc* par rapport à un seuil *Sc*.

– *Flexible* : c'est au niveau de traitement de tâche la plus prioritaire. Si le processeur reçoit une tâche très prioritaire, il interrompt la tâche en cours car sa priorité est moindre. Ainsi il met à jour les listes s'il y a changement d'état d'un de ses processeurs.

– *Intelligent* : Il effectue, gère et contrôle ces tâches. De plus, le choix de processeur pour le transfert n'est pas aléatoire. Le processeur surchargé interroge une liste trié de processeur non chargé *U_ LISTE* (il commence par le premier processeur qui est le moins chargé).

- *Proactif* : s'il est surchargé ($Nc>Sc$), il envoi immédiatement une demande de transfert au premier processeur Pq de la liste non chargée U_LISTE.
- *Social* : le processeur interagit avec les autres processeurs afin de compléter ces tâches. S'il est surchargé, il interroge les processeurs de la liste non chargée (U_LISTE) pour transférer sa tâche prête Ti vers le processeur le moins chargé.

Au cours d'exécution de l'algorithme de placement dynamique des tâches, il y a quelques caractéristiques ou variables du processeur qui peuvent être changées:

- Le nombre de tâches dans chaque processeur et leurs durées d'exécution peuvent être varié.

- Le processeur modifie ces croyances pour répondre aux besoins des autres processeurs afin d'assurer une meilleure interaction entre eux. S'il est non chargé, il peut être un récepteur d'une tâche. S'il est surchargé, il peut accepter une tâche d'un autre processeur déjà surchargé ou bien un émetteur afin de diminuer sa charge interne.

- Le changement de son état après le transfert d'une tâche de l'état *surchargé* à l'état *non chargé* ou état *indéterminé*.

- Le processeur doit percevoir les informations qui lui proviennent de son Environnement afin de mettre à jour sa base de connaissances. Ceci se fait par la mise à jour de ces listes (S_LISTE, U_LISTE et I_LISTE) à chaque changement d'état d'un processeur parmi ceux en liaison avec lui.

3.4.4. Caractéristiques de Système Multi Processeurs(*SMP*)

Un Système Multi Processeurs organise plusieurs processeurs qui peuvent s'exécuter en parallèle. Nous considérons un SMP comme étant un SMA. Nous pouvons en déduire et prouver les propriétés suivantes :

- La coopération : coopérer pour atteindre un but commun. Dans un Système Multi Processeurs, plusieurs processeurs fonctionnent en parallèle afin

d'obtenir une puissance de calcul plus importante que celle obtenue avec un seul processeur ou bien afin d'augmenter la disponibilité du système (en cas de panne d'un processeur). Les processeurs coopèrent ensemble afin d'améliorer la performance du système en cas de charges inéquitable des tâches. En fait, les processeurs interagissent en vue de partager la liste des tâches c'est-à-dire le processeur peut déléguer la tâche qu'il ne peut pas résoudre à un autre processeur dont il connaît les compétences.

- La coordination : la coordination entre les processeurs apparaît dans l'allocation de ressources rares et la communication de résultats intermédiaires et de paramètres.
- La négociation : la négociation entre les processeurs consiste à ordonner ces interventions et tenir compte des différents points de vue, de manière à ce que le groupe aboutis à une décision qui est la détermination de processeur destinataire qui va réceptionner la tâche à transférer:
  - le processeur émetteur Pc envoie une demande de transfert d'une tâche au premier processeur Pq de la liste non chargée U_LISTE.
  - le processeur Pq évalue cette demande, s'il est toujours non chargé alors il accepte la tâche.
  - sinon, le processeur Pq refuse cette demande et le processeur Pc passe au deuxième élément de la liste non chargée et répète la même procédure.

3.4.5. Interprétations

Le processeur est une entité autonome interactive qui connaît son état interne surchargé ou non. Il possède également une représentation de son Environnement (il connaît l'état des autres processeurs). Il est doté des capacités de raisonnement qui lui permettent de prendre la décision d'émettre ou bien de recevoir des tâches et de mettre à jour les listes s'il y a modification dans l'état de l'un des autres processeurs. Ainsi, les processeurs coopèrent afin de déterminer le processeur le plus capable à recevoir la tâche. En effet, le but commun de tous les processeurs est l'équilibrage de charge afin d'améliorer les performances du système, en répartissant la charge uniformément sur tous les processeurs.

D'après ce qui précédé, nous avons réussi à démontrer qu'il y a une coïncidence entre les caractéristiques d'un Agent (réactif, autonome, flexible, intelligent, proactif et social) et ceux d'un processeur. Les caractéristiques d'un Système Multi Processeurs peuvent être vues comme celle d'un Système Multi Agents (coopération, coordination et négociation).

3.4.6. Modèle RdPA de l'Algorithme « forward »

La **Figure 30** présente la modélisation de placement dynamique des tâches sur des processeurs parallèles par le biais de *RdPA*.

Nous représentons l'Agent et son autonomie dans la communication avec d'autres Agents de son environnement ou d'autres environnements tout en gardant une représentation graphique assez simple et compréhensible.

Le modèle présente une réduction remarquable de nombres de places et de transitions par rapport aux autres modèles *RdP* classiques. Les différents conditions, lors de franchissement d'une transitions respecte le sémantique des *RdPA*.

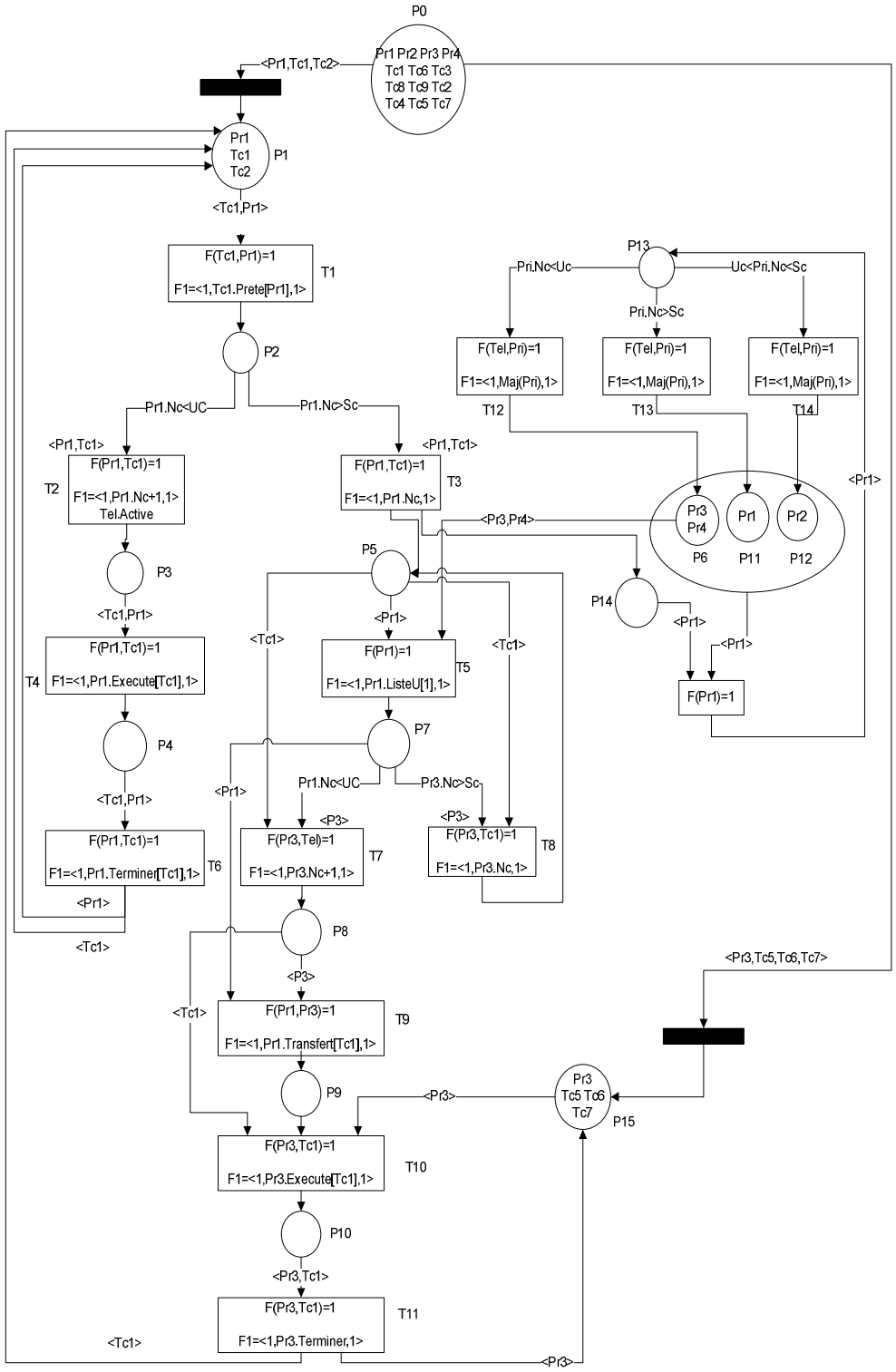


Figure 30. Modèle *RdPA* de l'Algorithme « *forward* »

3.5. De Systèmes Multi Agents Vers les Réseaux de Petri à Agents

Après la mise en place de notre modèle, nous présentons sous forme d'un tableau (**Tableau 6**) une correspondance entre l'approche Multi Agents et les Réseau de Petri à Agents selon des caractéristiques bien déterminées.

Caractéristiques	Système Multi Agents	Réseau de Petri à Agents
Nom	Agent	Jeton
	Etat du système	Place
	Ensemble de règles	*1-* $\forall p \in P\ M'(p)=M(p)\ Prè\ (p,t)+\ Post\ (p,t)$ 2- Pré condition ($Prj=1$) *3-* $F\ (Ai,\ Aj)=1$
	Ensemble de relations et d'actions	Transition
	Agent Administrateur	Agent Modérateur Total
Classe	Agent Réactif, Agents Cognitif, et Agents Hybride	Agent non Modérateur, Agent Modérateur et Agent Modérateur Total
Autonomie	Interaction entre les Agents	Fonction relation d'ordre 2 ou n : $F\ (Ai,\ Aj)=b$
Réactivité	Agent - Agents	Fonction Agent : $Ft=<Per,\ Valeur,\ Inter>$
Hétérogénéité	Agent - Environnement	Fonction d'appartenance relative à un Agent : $Apai=Apa(Ai,\ Envj,\ b,\ d)$
Sociabilité	Environnement - Agents	Fonction d'appartenance relative à un Agent ($Apai$) et à son Environnement : $Apej = Ape(Envj, \bigcup_{i=1}^{i=nk}(Ai, di))$
L'intelligence	Comportement, capacité d'interaction,	Exploitation de valeurs (*df, d, Valeur, et Historique*) donné par l'Architecture Agent Petri (AAP).

Tableau 6. Correspondance entre SMA et $RdPA$

3.6. Conclusion

Dans ce chapitre, nous avons donné toutes les définitions en relation avec notre nouveau modèle. Ce modèle donne naissance à un nouveau formalisme appelé Réseau de Petri à Agents ($RdPA$) dont l'Agent est représenté par un jeton. Les définitions issues de ce formalisme nous aident à définir aussi une nouvelle architecture des Agents. Cette architecture est nommée Architecture Agent Petri (AAP) qui décrit d'une manière efficace l'état interne et le comportement dynamique de l'Agent dans un SMA. L'association des APP de différents Agents permet une vision globale de fonctionnement de SMA. Nous avons défini la sémantique de $RdPA$ et les règles de franchissement d'une transition. Notre contribution prouve que le $RdPA$ a un pouvoir expressif permettant de bien modéliser le comportement des Agents au sein d'un Environnement.

Vue l'importance et la complexité de l'Interaction entre les Agents, nous envisagions d'approfondir notre recherche pour couvrir cette propriété fondamentale dans les SMA. Donc, Il faut garantir le bon déroulement de communication et de coopération. Ceci doit être parfaitement contrôlé par les protocoles d'Interaction. Notre réflexion est orientée vers la création d'un modèle en $RdPA$ pour modéliser et pour vérifier ce type de protocole.

4

Chapitre 4 : Modèle RdPA pour l'Interaction

Jusqu'à nos jours, il n'existe pas un formalisme unique décrivant les conversations entre les Agents. Chaque formalisme met l'accent sur un aspect particulier dans la modélisation de la communication. Notre travail n'échappe pas à cette règle. C'est pourquoi nous avons commencé par définir le formalisme de Réseaux de Petri à Agent dans le chapitre précédent, et nous présentons dans la suite de ce mémoire notre modèle pour la modélisation des Interactions dans un SMA à travers des protocoles et des performatives de communication du langage ACL adopté. Du moins aucune contribution innovante une quelconque méthode qui traiterait, à la fois la modélisation et la vérification d'Interactions entre les Agents dans un SMA.

4.1. Hypothèses et Cadre Général

Avant de détailler le modèle, nous précisons les hypothèses et le cadre général :

- Le *SMA* que nous considérons est constitué d'un ensemble d'Agents Cognitifs, s'exécutant en parallèle.
- Les Agents se communiquent par envoi de messages synchrones (l'émetteur est bloqué jusqu'à ce que le destinataire ait reçu le message, ou jusqu'à la réception d'une réponse éventuellement avec un délai maximal d'attente).
- Pour se comprendre et pouvoir communiquer parfaitement, nous supposons aussi que :
  - les Agents se partagent une ontologie commune,
  - ils sont dotés de mémoire (chaque Agent garde une trace à chaque fois qu'il exécute un évènement),
  - ils sont intelligents (chaque Agent dispose d'une stratégie de résolution et d'un mécanisme de raisonnement), pour retenir et exploiter le contenu et l'historique de leurs conversations.

4.2. Etapes de Modélisation

La spécification d'une conversation entre les Agents doit contenir plusieurs informations non seulement sur la conversation elle-même, mais aussi sur les Agents communicants.

En premier abord, le séquencement des messages doit être spécifié informellement en utilisant les « uses case », les chronogrammes et les diagrammes de séquences *UML* (Unified Modeling Language). Formellement, cette spécification est basée sur l'utilisation de plusieurs formalismes tels que les Réseaux de Petri à Agents dans notre cas. Ce formalisme combine les avantages des Réseaux de Petri classiques (comme le pouvoir d'expression et la représentation de la concurrence) et les caractéristiques des Systèmes Multi Agents dans le sens où un Agent peut s'engager dans plusieurs conversations et interagir avec d'autres Agents.

Ensuite, les rôles que les Agents jouent dans une conversation doivent être énumérés. De nombreuses conversations seront des dialogues, et préciseront que deux rôles. Toutefois, les conversations avec plus de deux rôles sont aussi importantes, représentant la coordination dans la communication entre plusieurs Agents dans un but commun. La modélisation des Interactions ou d'un protocole de communication dans un Système Multi Agents peut être retracée par les étapes suivantes :

Etape 1 : déterminer les différents Agents ayant un rôle dans la conversation implantée par le protocole.

Etape 2 : déterminer les différents états des Agents tout au long du processus de communication avant et après le franchissement d'une transition.

Etape 3 : déterminer les différentes conditions de garde de franchissement de chaque transition et les résultats de leur évaluation.

Etape 4 : déterminer la communication entre les Agents ; les messages que chaque Agent peut accepter ou envoyer dans ses différents états.

Etape 5: déterminer l'état initial du protocole et ses différents états de terminaison.

4.3. Modèle RdPA

L'idée sous-jacente des Réseaux de Petri à Agent est précisément de pouvoir bien représenter l'Agent et son autonomie. La communication se fait avec d'autres Agents de son Environnement ou d'autres Environnements tout en gardant une représentation graphique assez simple et compréhensible.

Le modèle de *RdPA* est constitué comme d'habitude de transitions, de places et d'arcs. Les transitions correspondent aux actions qui peuvent être réalisées. Les places sont les variables de l'espace états contenant des jetons correspondant aux Agents. Les arcs déterminent, selon leur orientation, les conditions d'activation d'une transition et son effet sur l'état.

Chaque place contient des jetons (Agents) et représente l'état des Agents mais peut aussi être utilisée pour des raisons techniques de synchronisation (par exemple time

out). Les variables étiquetant les arcs sont des paramètres définissant le déplacement des Agents lors du franchissement d'une transition et les règles de franchissement des Transitions.

Rappelons toutefois que les idées de base de *RdPA* sont les suivantes :

- Les Agents peuvent entrer en communication s'ils appartiennent à un même Environnement, par souci de simplification. Une fonction ou précondition de franchissement a été définie permettant de vérifier cette contrainte (*Prj=1* si la contrainte est satisfaite et *Prj=0* sinon).
- Par convention, un Agent est Modérateur total dans son Environnement s'il a le degré d'appartenance le plus élevé. Le modérateur total est celui qui gère la communication entre les Agents. On se sert de la notion de modérateur total ici lorsqu'on a plus de deux Agents en communication.
- On attache à chaque place et à chaque transition l'ensemble des Agents pouvant y être présents ou franchir. Un jeton représente un Agent dans notre modèle.
- La valuation d'un arc n'est ni un entier comme les *RDP P/T* ni une fonction comme dans les *RdPC* mais c'est un uplet représentant l'Agent (*<A1>*) ou les Agents qui peuvent franchir la transition (*<A1, A2>*) ou bien les messages échangés (*<A1.m>*) pour dire que le message *m* est envoyé par l'Agent *A1*.
- On suppose que si un Agent est engagé dans un Environnement c'est qu'il est en communication avec un autre Agent. Il quitte alors l'Environnement soit à la fin de la conversation soit s'il veut, tout simplement, arrêter la communication. Cette supposition nous permet de savoir si un Agent est occupé et ceci par une simple évaluation de la fonction *Prj*.

4.3.1. Association des Noms aux Jetons

Dans le but de différencier les jetons, on leur associe des noms (*A1, A2,…, An*). En conséquence, on associe à chaque place l'ensemble des jetons qui peuvent y être

présents. De même, on associe aux transitions les jetons qui peuvent les franchir pour être évalués lors du franchissement.

4.3.2. Valuation des Arcs

Dans un *RdPA*, les arcs ne sont plus valués par des entiers ou des fonctions. En effet, pour chaque transition, il faut préciser les jetons à franchir qui sont portés par les arcs d'entrée. A un arc, est associé un *n*-uplet des jetons qui doivent être retirés de la place quand la transition est franchie et rajoutés à celle de sortie selon la valuation des arcs de sortie. Etant donné que les jetons dans notre modèle *RdPA* représente des Agents, on trouvera les arcs valués par le nom de l'Agent $<Ai>$ ou bien le message envoyé par cet Agent $<Ai.m>$ car dans notre modèle ce sont les Agents qui circulent et les messages qu'ils s'envoient.

Par exemple, dans le cas le plus simple, un Agent *A1* dans une place *P1* va franchir une transition *T1* pour s'engager dans un Environnement *Env*. Lors du franchissement, la fonction *Prj* va être évaluée pour vérifier si *A1* est déjà engagé dans un autre Environnement.

4.3.3. Intégration des actes de langage FIPA-ACL

Un Agent, pour parvenir à son but, peut demander à un autre par un acte de langage, une aide ou d'une manière générale, une action. L'Agent, ayant reçu ce message, se doit de le comprendre et d'agir en conséquence selon ses connaissances, ses buts et sa disponibilité qui conditionnent sa réponse à la demande faite par le premier Agent. Il existe une réponse de non-compréhension due à une faute de règle de transmission même si on suppose généralement dans notre cas qu'on dispose d'un canal de communication fiable pour ne pas entrer dans les détails des erreurs de transmission.

4.3.4. Structure des Messages

Les Agents communiquent entre eux en échangeant ce qui peut représenter des actes de langage, c'est-à-dire des primitives *ACL* (*request*, *inform*, etc.).

Dans notre modèle, les fonctions *F (Ai, Aj)* et *Ft (tk)= <Per, Valeur, Inter>* nous permettent de définir les deux Agents en communication et de spécifier les messages échangés entre eux. Ainsi pour dire que deux Agents *A1* et *A2* communiquent et que l'Agent *A1* envoie une demande (*request*) à l'Agent *A2,* il suffit d'indiquer dans la transition *T1* par exemple que *F(A1, A2)=1* pour garantir que la communication est toujours établie et *Ft (t1)= <1, A1.request, Inter>* pour dire que l'Agent *A1* a envoyé une demande à l'Agent *A2. Inter* est une valeur de validation pour acquitter la réception de la requête. Autrement dit, si l'Agent *A2* a bien reçu la demande de *A1* il va lui répondre dans la transition *T2* par *Ft (t2)= <1, A1.request, 1> ou Ft (t2)= <1, A1.request, 0>* s'il n'a pas bien reçu le message. En cas de refus, *A2* peut répondre par *refused* dans la transition *Ft (t3)= <1, A2.refused, 0>* ou *Ft (t3)= <1, A2.agree, 0>* en réponse à sa demande.

4.3.5. Traitement des Messages

Le traitement des messages dans le cadre d'une Interaction entre Agents repose sur un ensemble de règles décrites à travers des protocoles d'Interactions Multi Agents. (Charif & Sabouret , 2006) ont exposés, en détails, qu'un message initiateur diffère d'un message reçu dans une Interaction déjà déclenchée. Ceci est dans le sens où l'Agent récepteur du message ou émetteur doit traiter chaque message ou requête séparément, construire la réponse et la stocker tout en considérant les conversations déjà faites et les messages échangés.

Lorsqu'un Agent reçoit un message, deux situations sont possibles :

- Il reconnait l'émetteur : il est capable de comprendre le message et de construire la réponse,
- Il ne reconnait pas l'émetteur et n'est pas capable de comprendre le message ni de construire la réponse.

Un Agent traitant un message a forcément la capacité de le comprendre, de le stocker, d'envoyer la réponse à l'Agent émetteur après l'avoir reconnu. Dans notre cas, les Agents sont des Agents considérés cognitifs communicants, capables de percevoir,

partiellement leur Environnement et capables de construire, d'envoyer et de recevoir des messages.

4.3.6. Contrôle des Conversations

Ce niveau concerne les mécanismes qui assurent le respect des règles des protocoles. Le contrôle du respect des règles peut être réparti (chaque participant veille, en ce qui le concerne, à respecter ou faire respecter les règles) ou assuré par un superviseur, par l'infrastructure ou l'Environnement d'exécution des Agents rendant physiquement impossible le non-respect des règles de protocole. Cette question concerne aussi bien les contraintes de distribution (authentification des Agents, limitation du nombre de participants, etc.) que celles du comportement (ordonnancement des interventions, activation de time-out, etc.).

Les règles de protocole sont des contraintes générales sur les séquences des messages sémantiquement logiques menant à un but. La cohérence du dialogue est, ainsi, assurée par ces contraintes. Dans notre modèle *RdPA*, nous utilisons les notions de places et de transitions pour représenter les liens de causalité qui existent entre elles et les points de synchronisation afin de bien structurer les actes de communication du protocole et modéliser le contrôle.

Le déroulement d'une conversation met en jeu, d'une part, des interventions qui sont effectuées par les entités qui participent à cette conversation, et d'autre part un contrôle qui vérifie que ces interventions respectent les règles du protocole. La réalisation des conversations est clairement à la charge des entités participantes. Mais, il n'y a aucune nécessité que le contrôle lui aussi, qui est traduit par le code de synchronisation, soit réparti dans les entités participantes. Il est donc possible dans l'implantation du protocole de dissocier le contrôle et la conversation et de les associer à des entités spécialisées que nous avons, déjà, appelées *Modérateurs*.

4.4. Exemples de Protocoles

La modélisation d'un protocole avec le formalisme *RdPA* se fait d'une manière itérative et sur plusieurs étapes. Initialement, le modèle *RdPA* est simple et couvre les parties essentielles du système en ignorant d'autres aspects jugés auxiliaires. La portée du modèle est après étendue et plus de détails sont pris en considération et des erreurs sont corrigées pour générer notre modèle final.

Un formalisme est jugé fiable s'il garantit quelques propriétés importantes telles que la synchronisation, la concurrence et la réutilisabilité. Il est donc naturel de composer des modèles de protocoles complexes à partir de protocoles simples ou d'un ensemble de protocoles reliant leurs éléments par des arcs et des places de synchronisation.

C'est pour cela que nous essayons, au début, de donner notre modèle d'ouverture de connexion entre deux Agents *A1* et *A2*. Ce protocole est dit « élémentaire » et vise à ouvrir une connexion entre deux Agents et il peut être réutilisé dans les autres exemples. Dans ce modèle, *A1* envoie une demande de connexion (communication) à *A2* en lui envoyant le message «*request*» (*F (A1, A2)*). Ce dernier, après réception de cette demande, peut accepter la communication en lui envoyant un message «*agree*» ou envoyer un message refuse dans le cas de refus (m'= {agree, refuse}). Le protocole se termine par la réception de la réponse «*agree*» et l'engagement d'*A2* dans l'Environnement et les deux Agents franchissent leur transition de fin (*Succès : T7*) ou la réception d'un message «*refuse*» par *A1* (*Echec : T8*) comme illustré dans la **Figure 31**.

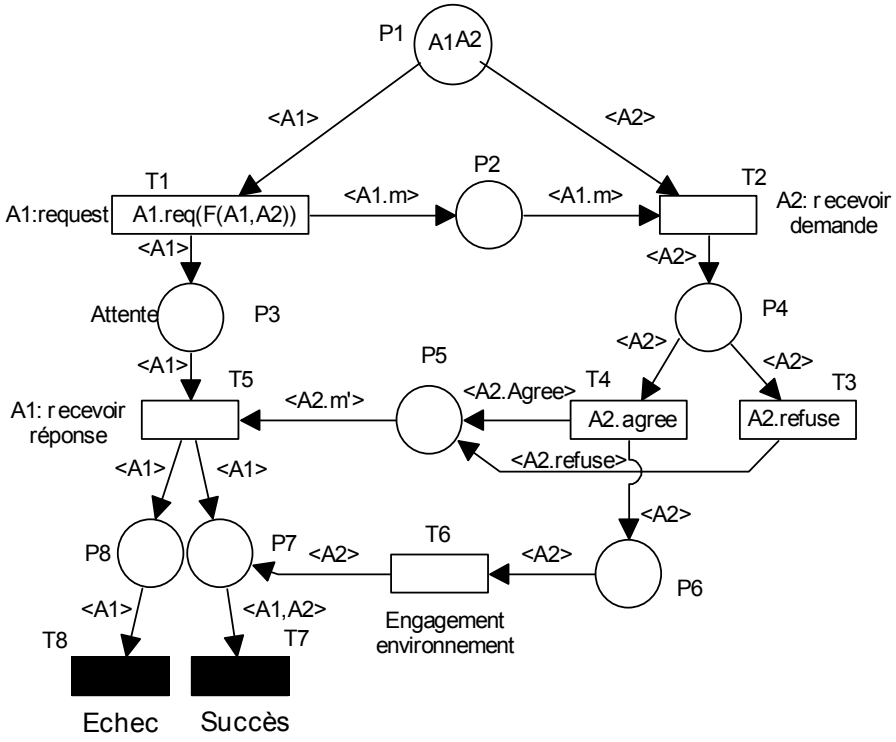


Figure 31. Ouverture de Connexion entre deux Agents

Dans le cas d'échec de connexion, dû à un refus de la part de *A2* soit parce qu'il ne veut pas entrer en communication avec *A1* soit parce qu'il est déjà occupé. Nous pouvons envisager des points de reprise dans notre modèle en *RdPA*. En effet, *A2* envoie le message *<A2.refuse>* et peut revenir à son état initial. *A1*, après avoir reçu ce message, revient lui aussi dans son état initial et une autre instance du protocole peut être déclenchée.

Maintenant que nous avons établi la définition formelle de notre formalisme *RdPA* et le rôle de chaque élément dans le modèle et sa sémantique, nous pouvons suivre les étapes citées ci-dessus pour modéliser des protocoles avec *RdPA* pour démontrer sa puissance expressive. Nous optons à des standards *FIPA* car ce sont des protocoles qui ont été amplement cités dans la littérature à chaque fois qu'un nouveau formalisme de modélisation est proposé. Nous commençons par deux protocoles simples définis dans

FIPA qui sont « *inform* » et « *request* ». Nous représentons, ainsi, une variante de protocole *FIPA-Contract Net* impliquant plus de deux Agents.

4.4.1. FIPA-Inform

C'est un simple acte communicatif pour passer une information d'un Agent à l'autre. Il fait interagir deux Agents : un Agent initiateur *A1* envoie un message « *inform* » (*T1*) à un autre Agent destinataire *A2* qui reçoit et traite le message (*T2*). La conversation se termine quand les deux Agents franchissent leurs transitions de fin (*T4*) et (*T5*) comme illustré dans la figure ci-dessous.

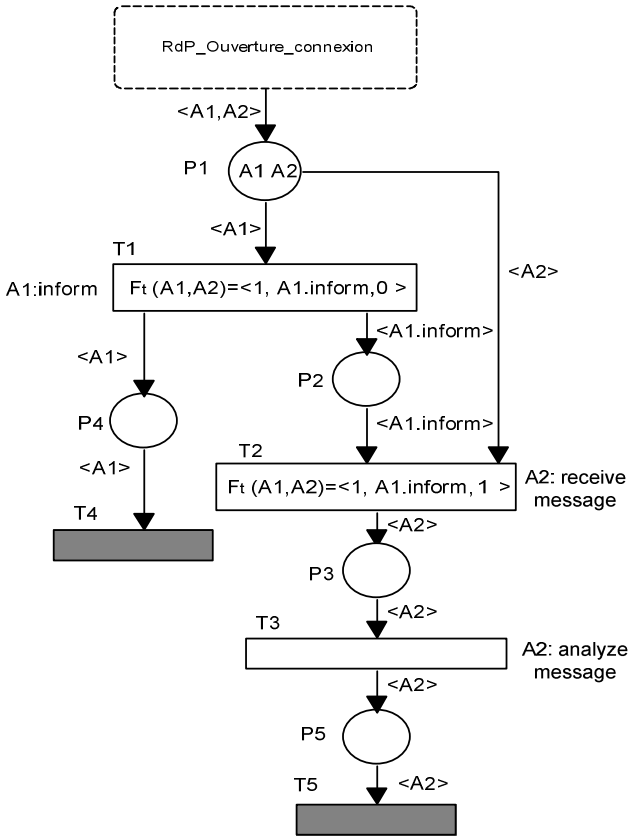


Figure 32. Le Modèle *RdPA* du Protocole *FIPA-Inform*

Nous avons supposé dans ce protocole que les deux Agents sont déjà en communication car nous avons déjà modélisé l'ouverture de connexion ci-dessus et on

a juste mentionné un appel au protocole élémentaire *RdPA_ouverture_connexion* et cela rappelle bien entendu le principe de réutilisabilité.

A1 envoie le message « *inform* » en utilisant la fonction *Ft1 (A1, A2)=<1, A1.inform, 0>* qui précise que les deux Agents en communication sont *A1* et *A2*, l'émetteur est *A1* et le destinataire est *A2* et que le message envoyé par *A1* est « *inform* ». La réception du message est validée par la valeur *1* dans le troisième champ de la fonction *Ft(A1, A2)* lors de la réception.

Avec la fonction *Ft()* nous pouvons modéliser l'envoi d'un message « *inform*» à plusieurs Agents en gardant toujours la même syntaxe : les destinataires sont mentionnés entre parenthèses et l'émetteur est *A1*, par exemple pour informer *A2*, *A3* et *A4* on aura : *Ft1 (A1, A2, A3, A4)=<1, A1.inform, 0 >*.

4.4.2. FIPA-Request

L'idée est de représenter un protocole de communication entre deux Agents *A1* et *A2*. Un Agent *A1* demande à un autre Agent *A2* d'exécuter une action *P*. Le récepteur peut consentir ou peut refuser d'exécuter l'action ou encore répondre qu'il n'a pas compris. En cas de refus, l'Agent récepteur est obligé de déclarer la raison du rejet. C'est le protocole *FIPA-Request* comme illustré par la **Figure 33**.

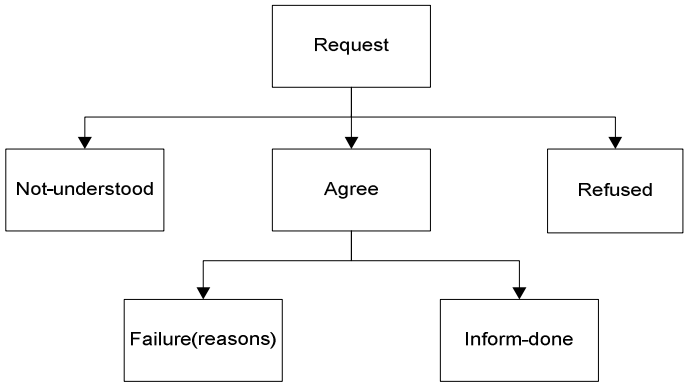


Figure 33. Protocole *FIPA-Request*

La **Figure 34** décrit ce même protocole en utilisant le formalisme de Réseaux de Petri ordinaire (les arcs de la relation des flots sont tous valus à 1). Chaque Agent exécute un Réseau de Petri dont les places correspondent à ses états ou l'état de la conversation et les transitions correspondent à l'envoi et la réception des messages.

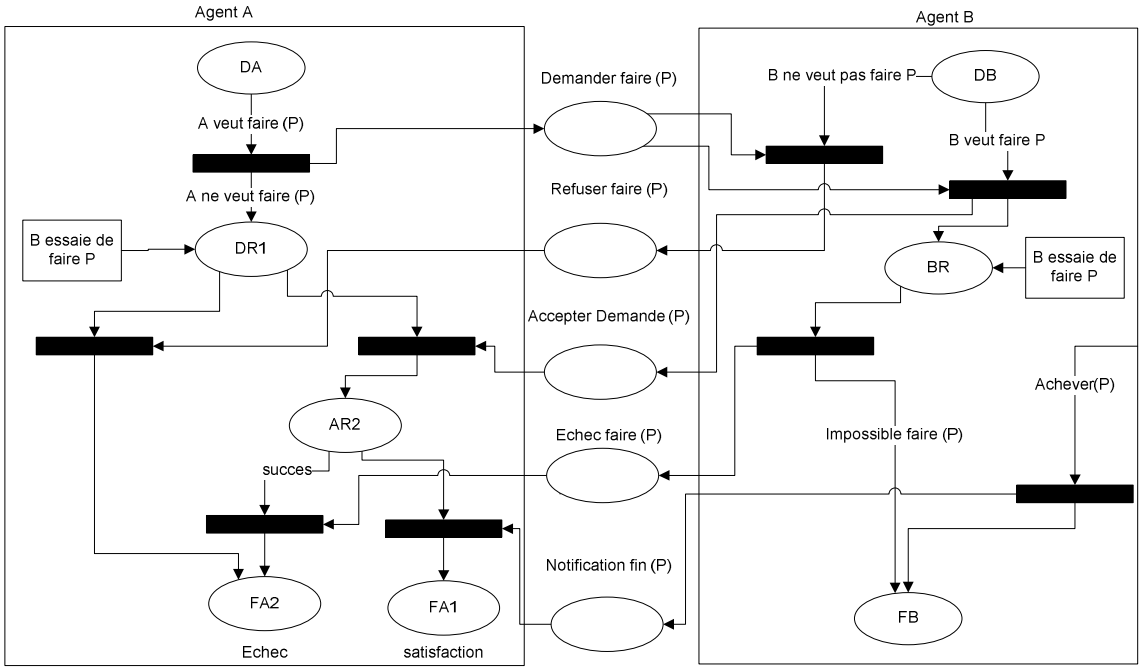


Figure 34. Le Protocole *FIPA-Request* avec *RdP* Ordinaire (Lehmann & Moldt, 2004)

Malgré la simplicité du protocole, plusieurs places, transitions et arcs ont été utilisés pour modéliser l'état de la conversation et des Agents tout au long de leur communication. Dans le modèle ordinaire, le concepteur est appelé à modéliser les deux cas possibles à chaque fois. Par exemple, *B* veut faire *P* et *B* ne veut pas faire *P* ce qui augmente, certes, le nombre des places utilisées, des arcs et des transitions et ne laisse pas explicite le choix de l'Agent car les jetons ne sont pas distingués.

Le but, est alors, de créer un modèle valide pour les deux Agents dans lequel la localisation de la décision de l'Agent doit être explicite, cela est possible avec l'utilisation des jetons (Agents) identifiables par leur noms.

Nous essayons de modéliser ce même protocole par le formalisme de *RdPA* (**Figure 35**) en raffinant à chaque fois notre modèle et en intégrant des primitives du langage *ACL*. *A1* envoie une demande de connexion à l'Agent *A2* avec la primitive « *request* ». *A2* peut accepter la demande, il lui répond dans ce cas avec un message *<A2.agree>*. Le message *<A2.refuse>* est envoyé s'il refuse la demande ou encore *<A2.not-understood>* en cas de non compréhension. Dans le cas d'acceptation d'*A2*, ce dernier essaie de faire *P* et envoie un message *<A2.inform-done>* dans le cas du succès ou un message *<A2.failure>* en cas d'échec de réalisation de la tâche. Toutefois cet échec laisse la possibilité de refaire la tâche par *A2*. Pour ce faire, nous devons rajouter des points de reprise à notre modèle.

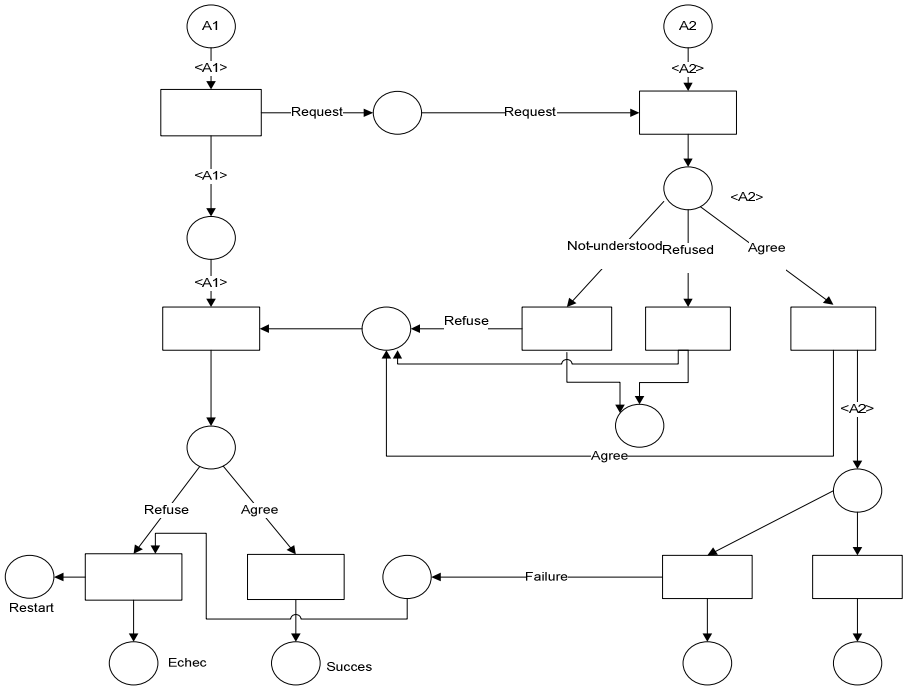


Figure 35. Les Différents Scénarios de Protocole *FIPA-Request*

Le réseau de Petri de la **Figure 35** modélise le protocole qui décrit les états relatifs à l'Interaction entre les deux Agents. Nous distinguons trois situations possibles : *succès*, *échec* suite au refus de la demande et un *échec* dans la réalisation de la tâche. Le modèle spécifie formellement comment l'Interaction entre ces deux Agents se

produit et quelles performatives sont utilisées à chaque étape de la conversation. Nous exposons dans la **Figure 36** le modèle *RdPA* de protocole *FIPA-Request* détaillé avec les messages échangés entre les deux Agents et les fonctions utilisées. Notons que, dans ce modèle et dans les *RdP* en général, il est toujours possible de capturer l'état actuel de la conversation ou de l'Agent par l'intermédiaire de marquage courant par la localisation des jetons (Agents).

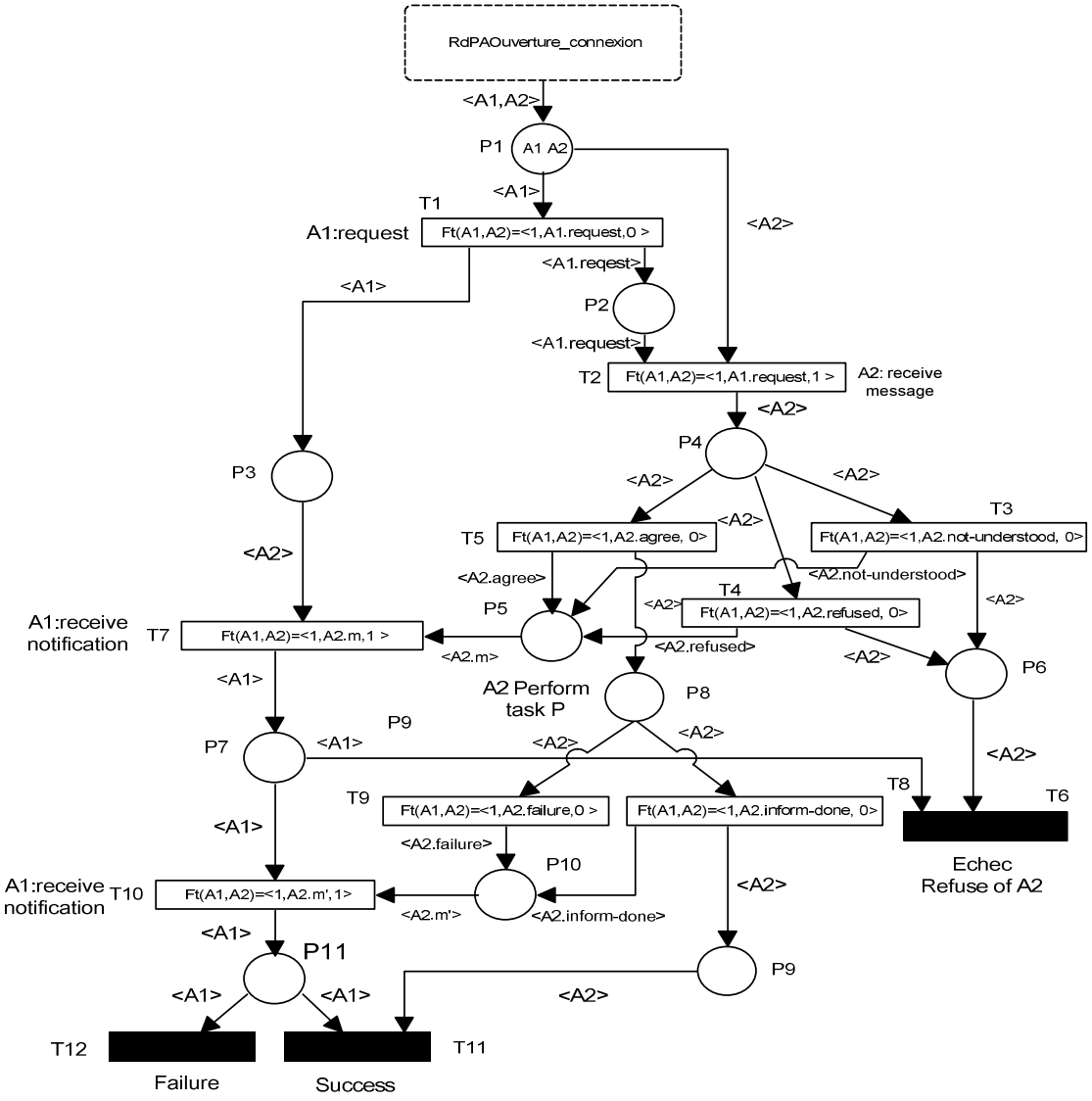


Figure 36. Le Modèle RdPA de Protocole FIPA-Request

Dans ce modèle nous avons supposé que les deux Agents sont en communication et engagés dans le même Environnement. De plus, nous avons fait un appel au protocole élémentaire d'ouverture connexion *RdPA_ouverture_connexion* entre les deux Agents *A1* et *A2*. Ainsi, nous avons enrichi notre modèle *RdPA* de *FIPA-Request* en précisant les différents messages échangés entre les deux Agents selon la structure des messages avec la fonction *Ft(Ai, Aj)*. Ceci en précisant à chaque fois l'émetteur et le destinataire et s'il s'agit d'un message initiateur ou d'une réponse à un message déjà reçu par la valeur de validation de la réception *Inter* de la fonction (Ft()).

Dans les deux cas d'échec, une nouvelle instance du protocole peut être déclenchée et des points de reprise ou états d'accueil peuvent être rajoutés. En effet, *A2* doit préciser les raisons de refus qui peuvent être soit parce qu'il n'a pas les compétences nécessaires pour faire la tâche, soit simplement parce qu'il refuse de la faire. Dans ce deuxième cas, *A1* peut recommencer une nouvelle tentative de communication.

Pour ce faire, nous devons identifier les deux cas d'échec dans notre modèle. Le premier cas d'échec est dû au refus d'*A2* : les deux Agents vont franchir la transition de fin *T6* et peuvent revenir à l'état initial par l'ajout d'un arc de *T6* vers *P1* portant les Agents en question. Le deuxième cas d'échec est dû à un problème dans la réalisation. *A2* peut décider d'exécuter la tâche. Pour cela, nous ajoutons un arc de la transition *T9* vers *P8*. *A1* peut ré-envoyer la demande à *A2*. Cela est possible par l'ajout d'un arc de la transition *T12* à *P1*.

Notons que les Agents en question sont des Agents cognitifs ayant la faculté de prendre des décisions et d'agir d'une façon autonome tout en suivant les règles de protocole. Un Agent peut rester bloqué dans un état d'attente d'une réponse mais il peut ne pas le faire si cela n'est pas nécessaire car la décision revient à lui seul dans ce cas.

Dans la logique d'amélioration de notre formalisme de *RdPA*, nous pouvons insérer un mécanisme de temporisation qui utilise une fonction *delay()* et une durée maximale *R* au-delà de laquelle l'Agent émetteur sort de l'état d'attente d'une réponse. Cette solution nous permet d'éviter qu'un Agent reste bloqué longtemps dans son attente.

En outre, dans les Réseaux de Petri *P/T,* l'accord doit être explicite par la modélisation de deux places différentes « *agree* » et « *refuse* » pour le canal de communication parce que les jetons sont indiscernables. Cela conduit à une situation de dépendance entre les deux Agents et l'identité du canal de communication. Or, le conflit ne devrait pas être résolu par l'identité du canal, mais par le type de message reçu ou envoyé. Ceci est garanti avec notre modèle *RdPA,* car les jetons sont différenciés par leurs noms et leurs comportements qui ne dépendent pas du canal de communication mais plutôt de leur choix, autrement dits des messages envoyés et reçus.

4.4.3. FIPA-Contract Net

Dans l'exemple suivant, nous essayons de montrer le pouvoir expressif de *RdPA* dans la modélisation des protocoles faisant intervenir plusieurs Agents tel que le protocole *FIPA-Contract Net*. Dans ce protocole, un Agent modérateur choisit un autre Agent pour effectuer une tâche. Ceci par diffusion d'un message de demande d'exécution d'une tâche *P* à tous les Agents de son Environnement et en attendant les réponses de ces derniers.

Notre but n'est pas seulement la modélisation du traitement local de l'Agent. Nous avons attribué à l'Agent modérateur la possibilité de choisir la première réponse positive et de refuser toutes les réponses qui viennent après, comme il peut annuler la négociation pendant la conversation comme illustré dans la **Figure 37**.

Nous distinguons plusieurs scénarios possibles :

- Si tous les Agents refusent l'offre du modérateur alors « *échec* »,
- Il existe une réponse positive. Dans cette situation trois cas sont aussi possibles :
  - si le modérateur accepte l'offre (*accept-proposal*) alors « *succès* »,
  - si le modérateur annule la négociation (*cancel*) alors « *échec* »,
  - le modérateur refuse toutes les offres alors « *reject* ».

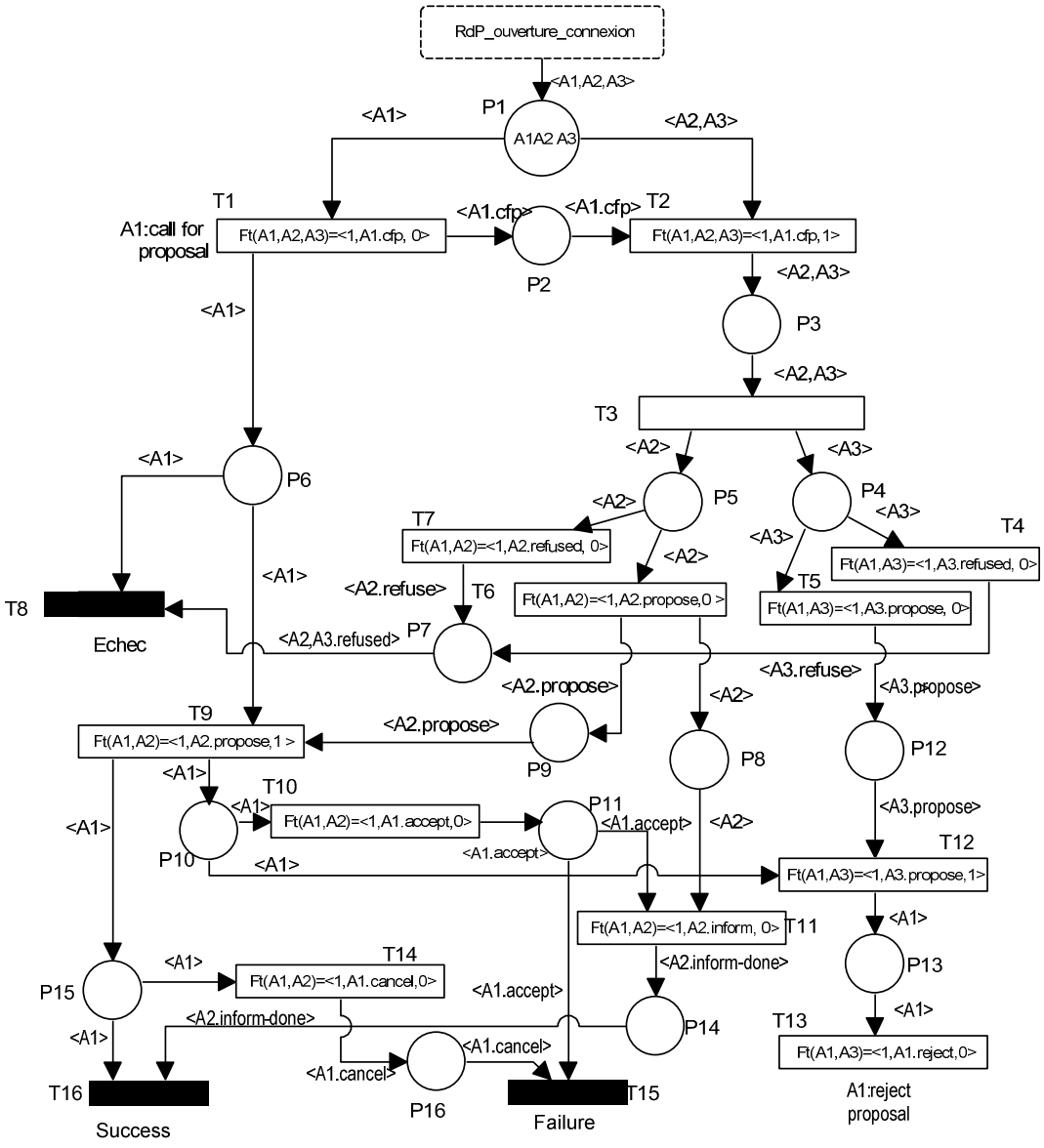


Figure 37 . Le Modèle *RdPA* de Protocole *FIPA-Contract Net*

Le fonctionnement de notre réseau est défini par les règles de franchissement des transitions permettant d'obtenir les séquences d'actions possibles dans notre protocole. Par exemple, le franchissement de la transition *Réception_ appel* (*T2*) dont l'arc est valué par *<A2, A3>* suppose la présence des deux Agents dans la place en amont. Cela facilite la création du graphe d'accessibilité de notre modèle lors de sa validation.

Le *RdPA* synthétise l'ensemble des spécifications nécessaires pour le suivi d'une Interaction en modélisant l'ensemble des évènements observables lors de l'Interaction et les relations de causalité à travers les places et les transitions.

Avec le *RdPA,* l'autonomie et la prise de décision de l'Agent est modélisée beaucoup plus explicitement qu'avec les autres formalismes de spécification. De plus, la valuation de l'arc indique l'Agent qui va exécuter l'action de la transition en cours et les conditions de sa réalisation.

Il est utile de souligner qu'une exécution conduit à un blocage du protocole suite à l'annulation par le modérateur (*cancel*), au cours de la réalisation de la tâche de la part de l'autre Agent. Ce blocage provient de la conception du protocole et non pas de sa modélisation. Dans notre modèle, ce cas est considéré comme un cas d'*échec* de la conversation.

Evidemment, les cas imprévus doivent, eux aussi être pris en considération. Durant la conversation, une action ou message inattendu ne doit pas bloquer les Agents ou empêcher le bon déroulement de leur conversation. Ceci nous mène à réfléchir à intégrer la gestion des exceptions dans le formalisme de *RdPA* pour garantir la cohérence des protocoles même si cette cohérence est bien assurée avec la modélisation des transitions de terminaison de la conversation.

4.4.4. Avantages de Modèle RdPA d'Interaction

L'un des avantages des *RdPA* est sa double représentation graphique compréhensible et ses fonctions mathématiques. Cela signifie qu'ils combinent une dénotation claire qui peut être visualisée dans la représentation graphique avec une sémantique opérationnelle qui peut être exécutée. Dans le *RdPA*, le modèle et la mise en œuvre se confondent en une représentation simplifiée du système. Les modèles peuvent déjà être exécutés pour la simulation. Cela, peut être un avantage pour la compréhension de l'Interaction et la vérification de sa bonne réalisation. Néanmoins, les Réseaux de Petri à Agents peuvent également être utilisés pour la modélisation abstraite qui peut

montrer des aperçus ou détails (raffinement) du système. Traditionnellement, les Réseaux de Petri ont été principalement utilisés pour la modélisation.

Un autre avantage est l'expressivité des Réseaux de Petri à Agents qui sont assez puissants pour décrire n'importe quelle exécution, même les processus concurrents et distribués peuvent être décrits, grâce aux contraintes associées aux franchissements des transitions. Ceci donne aux *RdPA* un avantage sur les langages de programmation classiques et les autres formalismes de modélisation, voire même des autres extensions des *RdP*.

L'apport majeur de modèle *RdPA* d'Interaction est d'une part, son pouvoir de mettre en avant l'autonomie des Agents et l'Interaction entre eux en précisant les messages échangés lors de la conversation et d'autre part, de simplifier la compréhension de l'état interne de l'Agent en suivant son comportement lors de l'interaction.

En plus, une amélioration peut être faite sur le modèle *RdPA* afin d'améliorer son pouvoir expressif. Il s'agit d'ajouter un mécanisme de temporisation et une gestion d'exceptions pour assurer une bonne exécution des Interactions et éviter tout blocage des Agents.

4.4.5. Raffinement du Modèle

Une question reste en suspens: quand avoir le bon modèle? Autrement dit, à quelle étape doit-on arrêter la modélisation et dire qu'on on a atteint notre but de modélisation.

En fait, si le modèle est compréhensible et atteint le but pour lequel il a été conçu, alors, il est efficace et il peut aussi être le bon modèle. Ici, un critère supplémentaire doit être indiqué : le modèle doit être sujet d'une vérification et d'une validation pour s'assurer que ce qu'on a obtenu est bien celui souhaité. Ceci est possible par la vérification de l'existence ou la présence des propriétés souhaitées et l'absence de celles non désirées. Cette vérification fera l'objet de la section suivante.

4.5. Vérification et Validation du Modèle

Rappelons que nous avons présenté, dans les sections précédentes, notre modèle *RdPA* pour l'Interaction en définissant les éléments essentiels du modèle et sa sémantique. Nous avons proposé la modélisation de quelques protocoles standards de *FIPA* et nous avons mis en avant les avantages de notre modèle en *RdPA* permettant une grande expressivité. Nous montrons que les *RdPA* facilitent la vérification et la validation formelle des propriétés sur le modèle obtenu. Une fois que le modèle est construit, il devient nécessaire de déterminer si ce modèle est valide. Ceci en établissant un plan d'expériences de ce modèle, afin d'en étudier son comportement, fondé généralement sur un cycle de vérification /validation pour vérifier l'absence des propriétés indésirables et la présence de celles souhaitées.

4.5.1. Précision Vocabulaire

Notons que la vérification cherche d'abord à répondre à la question : Construisons-nous correctement le modèle? Le but est de prouver la cohérence du modèle, de s'assurer de la bonne utilisation des moyens de modélisation et de rendre compte de la description des exigences qui ont prévalues à l'existence de ce modèle. La validation cherche à répondre à la question Construisons-nous le bon modèle ? Il s'agit ici de s'assurer de la pertinence du modèle, d'une certaine forme de complétude au regard du système modélisé, de ses situations et de ses scénarios d'évolution.

La vérification et la validation des modèles sont considérées parmi les étapes les plus importantes dans le processus de modélisation surtout pour les Systèmes Multi Agents ouverts.

4.5.2. L'Importance de la Vérification et la Validation

La vérification et la validation consistent à s'assurer que la spécification dispose d'un certain nombre de propriétés. En effet, des comportements anormaux, dus à une spécification incomplète, ou à des erreurs de conception, peuvent d'ores et déjà être perçus en exécutant la spécification.

Depuis les travaux de (Martial, 1992) la validation du modèle est devenu une étape de base pendant la modélisation du protocole d'Interaction. Des Réseaux de Petri ordinaires ont été utilisés pour vérifier le modèle de protocole conçu et assurer certaines propriétés.

Pour vérifier dans quel sens un système est valide, il faut définir formellement les comportements désirés et les comportements non désirés de notre système.

4.5.3. La Portée de Validation dans le Modèle RdPA

En ce qui concerne la portée de la validation, nous nous sommes intéressés à valider notre modèle *RdPA* d'Interaction. Plusieurs travaux ont déjà été menés sur l'utilisation de Réseaux de Petri, en général, pour valider le modèle de conversation d'Agents. Le résultat du travail de (Burkhard, 1993) qui a confirmé que tout modèle de *RdP* peut être transformé en *RdP ordinaire* justifie également notre choix pour le modèle de validation. Les propriétés liées au modèle d'Agents peuvent être obtenues uniquement par l'intermédiaire d'un modèle global du système.

Il est à noter qu'à un modèle *RdPA* on peut associer un Réseau de Petri *ordinaire* ayant un comportement identique vu que notre espace d'états ainsi que le nombre d'Agents sont finis. Ceci nous permet d'exploiter les outils assez disponibles de vérification et validation pour les Réseaux de Petri ordinaires.
De ce fait, nous allons traduire le modèle décrit en *RdPA* en Réseau de Petri *ordinaire*, selon une règle de transformation, c'est-à-dire « si l'Agent est dans un *état_x*, et s'il reçoit un *message_i*, alors il envoie un *message_j* et il passe dans un *état_y* ». Nous avons pu transformer la structure qui représente la transition des états du protocole de communication d'Agents en un modèle de validation en Réseaux de Petri *ordinaire*, sans difficulté, sur lequel nous allons valider des propriétés, comme par exemple la vivacité et le non-blocage.

4.5.4. Méthodes de Validation dans un RdPA

Au début de ce chapitre, nous avons défini un modèle formel d'Interaction entre les Agents dont les modèles représentent des similitudes avec les systèmes à évènements discrets. Ces modèles doivent ensuite être analysés. De nombreux travaux ont été consacrés à l'analyse des Réseaux de Petri et plusieurs méthodes ont été utilisées à ce propos. Dans ce qui suit, nous passons certaines d'entre elles en revue.

Notre synthèse sur les méthodes de validation a montré les trois méthodes de validation les plus couramment utilisées:

- *Méthodes faisant intervenir l'algèbre linéaire* : ils analysent les Réseaux de Petri à partir de leurs matrices d'incidence et de l'équation fondamentale qui fait évoluer le marquage dans le réseau. Elles sont efficaces pour étudier les propriétés structurelles. En général, avec ces méthodes, nous ne pouvons trouver que des conditions nécessaires ou suffisantes d'accessibilité de marquages.

- *Techniques de réduction* : l'idée de ces méthodes est de trouver des règles de réduction telles que le réseau réduit qui garde les mêmes propriétés que le réseau initial. On peut ainsi appliquer sur le modèle de taille réduite les méthodes de la troisième catégorie.

- *Méthodes basées sur le graphe des marquages accessibles* : concernent les méthodes énumératives de tous les marquages de réseau et permettent de vérifier des propriétés de Réseaux de Petri. Elles conviennent à tout type de Réseaux de Petri sauf qu'elles peuvent engendrer une explosion combinatoire.

Par le biais des RdP, nous nous intéresserons qu'à l'analyse d'accessibilité.

Le principe de cette technique est l'énumération de toutes les interactions possibles entre les entités communicantes. Cette énumération génère des états globaux de protocole où chaque état global donne l'état du protocole après avoir exécuté une opération. A la fin de cette opération, on obtient un graphe d'accessibilité du

protocole. Les erreurs du protocole sont alors détectées par la vérification de chaque état global composant ce graphe (Tari & Arora, 2007). Les états et la structure de ce graphe sont ensuite examinés à la lumière d'un certain nombre de propriétés : interblocage, réception non spécifiée, pour valider le protocole d'interaction mis en œuvre. Pour vérifier la validité du protocole, deux types de propriétés doivent être contrôlées : des propriétés générales que l'on retrouve dans tous les protocoles comme des problèmes d'interblocage et des propriétés fonctionnelles qui sont dépendantes de la fonction du protocole.

L'analyse d'accessibilité standard est une technique très utile, simple qui permet la détection de toutes les erreurs que peut contenir une spécification de protocole. Cependant, elle souffre du problème de l'explosion combinatoire. Le nombre total des états globaux d'un protocole est exponentiel en termes du nombre d'entités composant le protocole ainsi que le nombre d'états locaux de chaque entité. Ceci qui a limité leur application à des protocoles simples.

Le passage à un nombre supérieur d'Agents dans une conversation n'entraine qu'une augmentation du nombre de copies des protocoles et des chemins supplémentaires dans le graphe des états accessibles.

4.5.5. Les Propriétés Structurelles et Comportementales d'un RdPA

Une des principales qualités des Réseaux de Petri est que ces derniers proposent des techniques permettant d'analyser et prouver des propriétés sur le modèle du système conçu. Pour les protocoles, la détection d'inconsistance dans la séquence d'échange de messages et la détermination de la vivacité de tous les états possibles est un aspect crucial du développement, étant donné la quantité élevée de messages qu'un protocole utilise habituellement. Néanmoins, certaines de ces propriétés sont générales dont le sens où elles sont significatives quel que soit le protocole considéré. Nous en citons à titre d'exemples :

- Les conversations d'un protocole d'Interaction sont destinées soit à revenir périodiquement à un état de reprise soit à se terminer dans un état objectif.

Dans un premier temps, nous devons montrer que l'état de reprise ou début d'un nouveau cycle, est toujours accessible par une séquence d'interventions appropriées. Ceci est quel que soit l'état atteint à partir de l'état initial. Dans un deuxième temps, nous devons vérifier que l'état objectif est accessible et que c'est le seul état bloquant.

- Pour chaque intervention, il existe un état accessible à partir duquel elle peut être réalisée au moins une fois.
- Toute requête reçoit une réponse pour que les interventions nécessaires à l'élaboration de cette réponse soient effectuées.
- Les Agents impliqués dans une conversation atteignent tous les états de fin respectifs et franchissent leurs transitions finales respectives.
- Tous les états sont atteignables à partir de l'état initial et l'utilisation de tous les messages du protocole est assurée.
- Le protocole ne doit pas engendrer un processus avec un nombre infini d'états.
- D'une façon générale, il est possible de construire le graphe d'états d'un réseau, c'est-à-dire l'ensemble de tous ses états accessibles. De même, il est possible de construire l'ensemble de toutes les séquences de transitions qui permettent d'atteindre un état bloquant à partir d'un état initial.

Par ailleurs, nous pouvons souligner l'existence d'autres propriétés de base qui sont, d'une part, des conditions fortes de la validité de spécifications faites et, d'autre part, sont spécifiques à un protocole donné dans la mesure où elles correspondent aux règles comportementales de ce protocole. A travers l'exemple du protocole *FIPA-Request* (**Figure 34**), nous pourrons vérifier les propriétés suivantes, qui ne sont finalement qu'un énoncé formel de la description du protocole donné :

- *« A1 : request »* ne peut être effectuée que si les deux Agents sont déjà en communication c'est-à-dire *F (A1, A2)=1,*
- Lorsque *« A2 : agree »* a été réalisée, *« A2 : refuse »* et *« A2 : not-understand »* ne peuvent plus l'être,

- Le protocole ne se termine avec la transition de terminaison « succès » que lorsque l'Agent *A1* reçoit une notification « *inform-done* » de la part d'*A2*.

De plus, l'introduction de la notion de transitions terminales apporte un grand intérêt pour l'analyse et la vérification. En effet, tout d'abord on doit s'assurer que les séquences du réseau soient finies. Ensuite, on s'intéresse aux séquences finies maximales du réseau dont l'interprétation peut se faire comme suit :

- Une séquence finie qui se termine avec une transition de fin signifie que la conversation a été menée jusqu'au bout et qu'elle est sémantiquement correcte,
- Une séquence finie non terminée par une transition de fin signifie un blocage dans le protocole ; donc une erreur de conception,
- Une séquence infinie signifie qu'il existe une boucle dans le protocole et que la conversation n'arrive pas à s'achever.

Les méthodes d'analyse présentées dans ce paragraphe permettent de vérifier les propriétés que nous introduirons dans la section suivante.

Pour cela, nous devons distinguer deux grandes familles de propriétés pour un *RdPA* et qui sont les propriétés structurelles et les propriétés comportementales :

4.5.5.1. Propriétés Structurelles

Ces propriétés sont indépendantes du marquage initial du réseau. L'étude de ces propriétés est importante lorsque nous concevons un protocole dont le nombre d'Agents participants n'est pas connu.

La validation structurelle implique la possibilité d'étudier les propriétés structurelles et fonctionnelles des protocoles d'Interaction en tant que modèles. Ainsi, nous pouvons vérifier pour un *RdPA* des propriétés telles que :

- *L'absence du blocage* : les Agents impliqués dans une conversation atteignent tous leurs états de fin respectifs,

- *L'absence du cycle ou de séquences infinies* : tous les Agents franchissent leurs transitions finales respectives,

- *La vivacité* : tous les états sont atteignables à partir de l'état initial et l'utilisation de tous les messages du protocole est assurée. Cette propriété peut être vérifiée, de manière structurelle ou non par le développement de tout ou d'une partie du graphe d'états,

- *Le caractère borné du modèle* : le protocole ne peut engendrer un processus avec un nombre infini d'états.

Nous prenons le modèle de protocole *FIPA-Request* (**Figure 36**) modélisé. Nous énonçons la propriété et nous interprétons sa validité sur ce modèle. Pour ce faire, nous avons commencé par construire notre modèle de validation de protocole dans le cas de succès (**Figure 38**). Nous commentons dans la suite les propriétés que vérifient ce modèle et les erreurs qu'on pourra détecter après la vérification.

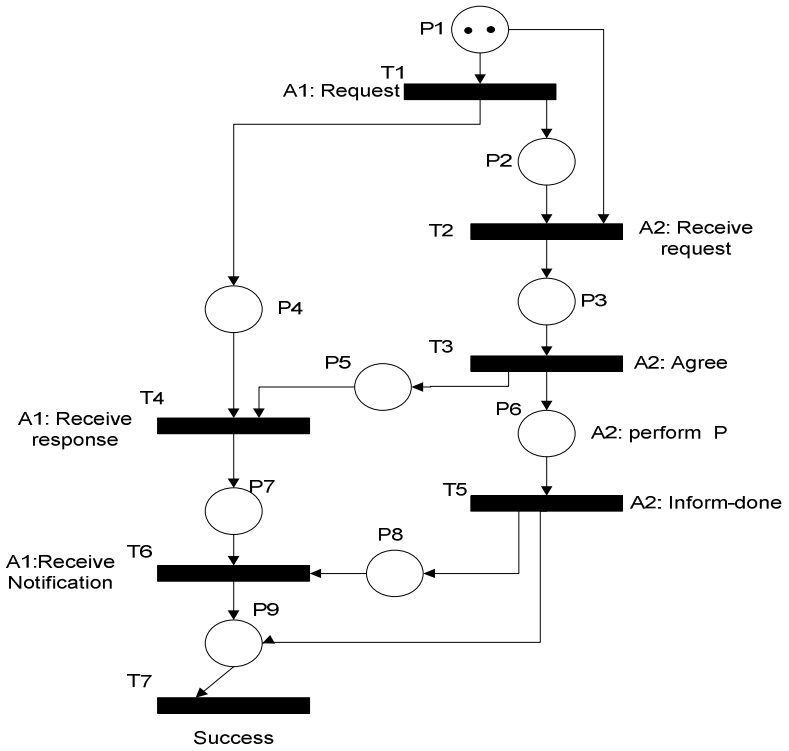


Figure 38. Le Modèle de Validation de Protocole *FIPA-Request* en cas de Succès

Une première vision du modèle révèle des simplifications et des réductions à faire en tenant compte qu'on a un seul scénario celui du succès de la conversation. En premier abord, on constate que la place *P1* est une place d'entrée pour les deux transitions *T1* et *T2*. En réalité, nous avons opté, dans le modèle *RdPA* de protocole « *request* », de représenter un arc de *P1* vers *T2* portant *<A2>*. L'Agent *A2* n'intervient réellement que dans cette transition(*T2*) pour valider la réception du message *<A1.request>* et cette représentation n'affectera en aucun cas le comportement de notre modèle *RdPA* où les Agents sont identifiables par leurs noms. En effet, dans ce modèle, on a lors du franchissement de *T1*, un tir de l'Agent *A1* de la place *P1*. Lors du franchissement de *T2,* on a un tirage d'*A2* porté par l'arc menant de *P1* vers *T2* comme illustré dans la Figure 36.

En outre, nous pouvons omettre la place *P9* dans le modèle de validation et cela ne changera rien dans le réseau car la réception de la notification (*T6*) ne concerne que le cas de succès.

Vivacité Structurelle

Dans un protocole d'Interaction, la vivacité structurelle garantit l'existence d'un état initial tel que quel que soit l'état accessible, on pourra toujours exécuter au moins une opération. La vérification de cette propriété nous garantit que la conversation entre les Agents évolue et ne tombe pas dans un état bloquant étant donné notre marquage initial.

Dans notre modèle de *FIPA-Request*, la première action *A1* : « *request* » est toujours accessible quel que soit le marquage initial du réseau du moment où *A1* est présent dans *P1*.

Répétitivité

Cette propriété garantit l'existence d'un contrôle de manière que toutes les opérations peuvent être exécutées une infinité de fois.

Cette propriété n'est vraiment pas intéressante dans notre étude des propriétés car on s'intéresse à étudier une seule instance de protocole. Mais, elle peut être vérifiée dans

notre modèle *RdPA* de *FIPA-Request* si on n'a pas associé des transitions de fins communes aux Agents renseignant sur l'état de terminaison de la conversation (*succès*, *échec*) après l'exécution de la conversation.

Consistance

Dans les protocoles d'Interaction, la consistance assure l'existence d'un contrôle tel que toutes les opérations sont exécutables. Par exemple un Agent revient périodiquement à son état initial après la fin de la conversation.

Certes, nous pouvons avoir un modèle consistant pour notre variante de *FIPA-Request* en ajoutant un arc menant de la transition de fin à l'état initial. Ceci est notamment faisable, mais n'est pas sémantiquement tout à fait juste. En effet, un Agent, en entrant dans une conversation (recevant et envoyant des messages), ne peut pas revenir à son état initial vu que chaque message reçu ou émis conduit à des modifications et à des prises de décision relatives à ce dernier.

Le modèle *RdPA* de *FIPA-Request* est consistant en tenant compte qu'on n'a des Agents à la place des jetons simples.

Bornitude Structurelle

Dans le protocole de communication entre Agents, la bornitude structurelle garantit que le nombre d'occurrences des objets modélisés reste limité quels que soient les états initiaux, c'est-à-dire qu'il n'y aura pas une accumulation des jetons.

Dans notre modèle, cette propriété est assurée par l'ajout de transitions de terminaison que les Agents franchissent à la fin de la conversation ce qui n'entraine en aucun cas une accumulation des Agents ni des messages dans notre modèle.

4.5.5.2. Propriétés Comportementales

Ces propriétés dépendent de la structure du Réseau de Petri à Agent et du marquage initial, basées essentiellement sur la matrice d'incidence et sur l'équation fondamentale qui fait évoluer le marquage.

Les matrices caractérisant notre modèle *FIPA-Request* de validation (**Figure 38**) sont :

T

[T1 T2 T3 T4 T6 T7]

M0	P	W-	W+	W
2	P1	1 0 0 0 0 0	0 0 0 0 0 0	-1 0 0 0 0 0
0	P2	0 0 0 1 0 0	1 0 0 0 0 0	1 0 0 -1 0 0
0	P3	0 1 0 0 0 0	1 0 0 0 0 0	1 -1 0 0 0 0
0	P4	0 0 1 0 0 0	0 1 0 0 0 0	0 1 -1 0 0 0
0	P5	0 0 0 0 1 0	0 0 1 0 0 0	0 0 1 0 -1 0
0	P7	0 0 0 0 0 1	0 0 0 1 0 0	0 0 0 1 0 -1
0	P8	0 0 0 1 0 0	0 0 1 0 0 0	0 0 1 -1 0 0
0	P9	0 0 0 0 0 1	0 0 0 0 1 0	0 0 0 0 1 -1

M0 désigne le marquage initial, *W*, *W-* et *W+* correspondent respectivement aux matrices d'incidence, matrice d'incidence avant et matrice d'incidence arrière.

Avec ces matrices, nous pouvons suivre facilement l'évolution de notre réseau depuis son état initial(*M0*) et avant et après le franchissement de chaque transition et nous pouvons comprendre le déplacement des jetons ou des Agents dans notre cas sur la matrice d'incidence. Par exemple, nous pouvons lire sur la matrice d'incidence *W* sur la dernière colonne qu'à la fin de la conversation, après franchissement de la transition de terminaison on a eu un tirage de deux jetons correspondant dans notre modèle *RdPA* aux deux Agents *A1* et *A2.* Nous avons atteint, ainsi, l'état objectif de marquage nul.

Parmi les propriétés comportementales d'un *RdP* nous présentons :

Accessibilité des Marquages

L'accessibilité est une propriété considérablement importante dans l'étude des propriétés dynamiques d'un Réseau de Petri. Dans un protocole de communication des Agents, cette propriété garantit la contrôlabilité de l'Interaction dans le sens où elle permet de conduire la conversation à un état souhaité à partir de l'état initial. Nous construisons le graphe des marquages accessibles de notre modèle *FIPA-Request* (**Figure 39**). Pour ce faire, nous associons à la présence d'un jeton dans une place la valeur *1* et son absence signifie la valeur *0*. Deux situations peuvent alors se présenter :

- Le graphe est fini. C'est la situation la plus favorable car dans ce cas toutes les propriétés peuvent être déduites simplement par inspection de celui-ci,
- Le graphe est infini. Dans ce cas, nous construisons un autre graphe appelé graphe de couverture permettant de déduire certaines propriétés. Nous pouvons également passé par un arbre de couverture.

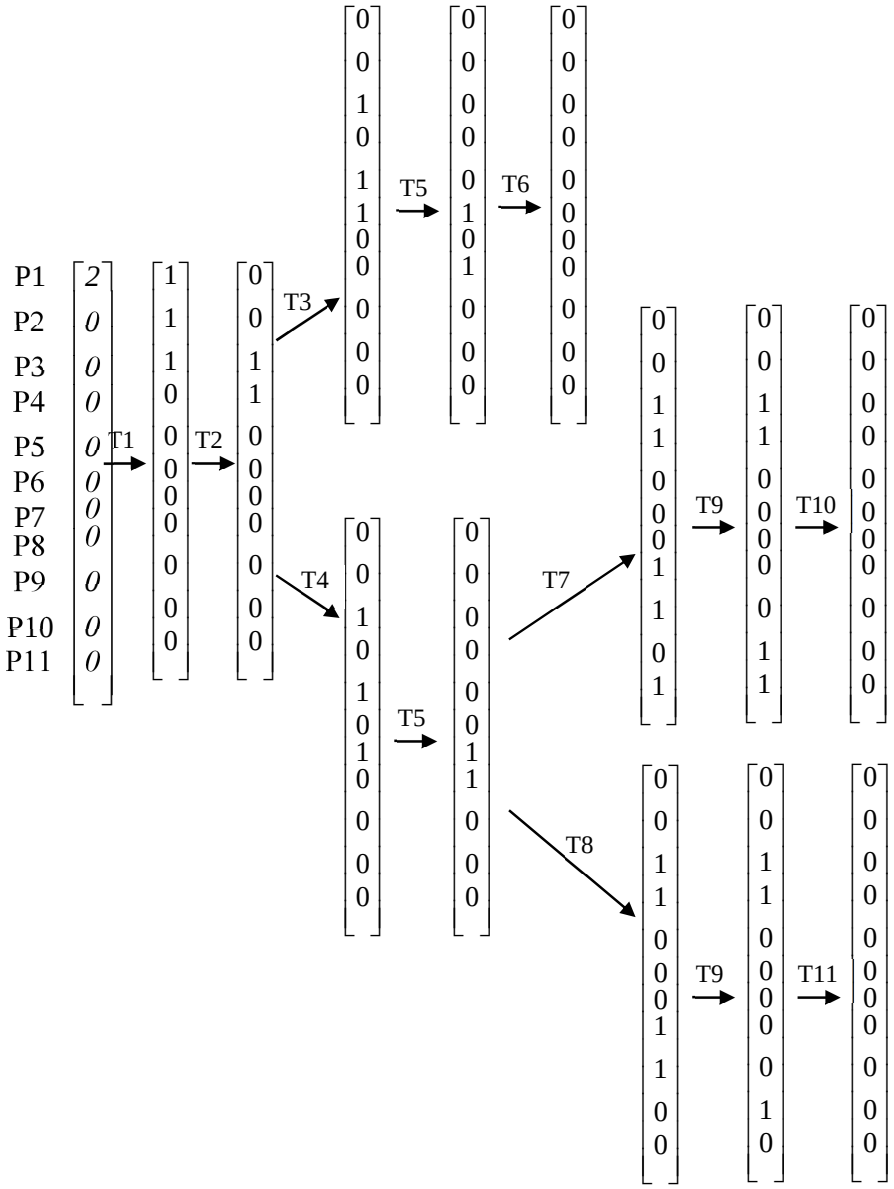


Figure 39. Graphe de Marquages Accessibles de Protocole *FIPA-Request*

Sur le graphe de marquage on peut suivre clairement le déroulement de la communication entre les Agents présentés par les franchissements de toutes les transitions de notre modèle.

On constate qu'aucun état de blocage ne peut survenir et que la conversation est menée jusqu'aux transitions finales ou états objectifs représentés par le marquage nul du réseau.

Sur le graphe des marquages accessibles, nous pouvons même lire les différents scénarios modélisés dans notre modèle de protocole *FIPA-Request* :

- La Séquence T1T2T3T5T6 correspond au scenario d'échec dû au refus de l'Agent à réaliser la tâche,
- La Séquence T1T2T4T5T8T9T11 correspond au scenario d'échec de la réalisation de la tâche,
- La Séquence T1T2T4T5T8T9T10 correspond au scenario du succès représenté par la **Figure 37**.

Caractère Borné et Sain

Le caractère borné garantit que le nombre d'Agents et de messages véhiculant dans les places reste limité à un état initial donné, autrement dit, les messages circulant sont toujours traités et ne s'accumulent pas dans le réseau. L'absence de ce caractère borné peut déclencher un traitement des messages infini dans le réseau alors que ce n'est pas le cas dans les conversations inter Agents.
Cette propriété est bien satisfaite dans notre exemple puisque notre état objectif, avoir un réseau sans jetons, est bien atteint d'où, on aura en aucun cas un envoi infini des messages dans le réseau.

Absence du Blocage

Nous pouvons définir le blocage comme :

- l'exécution d'une infinité d'instructions dans un intervalle de temps borné. Le système reste bloqué alors que le temps global évolue,

- deux Agents *A1* et *A2* se retrouvent simultanément dans un état où *A1* attend une ressource ou un message de *A2* et inversement. Nous appelons cette situation *interblocage*.

L'absence du blocage dans un Protocole d'Interaction garantit qu'au moins une opération est toujours exécutée étant donné un état initial et quel que soit l'état courant de la conversation.

Le graphe d'accessibilité de notre modèle *FIPA-Request* révèle qu'aucun état de blocage ne peut se produire et que les Agents exécutent leur actions jusqu'à la transition de fin quel que soit l'état de terminaison (*échec* ou *succès*).

Vivacité

La vivacité est une propriété assez importante dans le contexte des protocoles d'Interaction. Elle assure que pour un état initial donné, toutes les opérations peuvent toujours s'exécuter. Ceci veut dire que tous les messages du protocole doivent être utilisés bien entendu selon les actions des Agents et que ces derniers exécutent toutes leurs actions comme prévu lors de la spécification. La vivacité est une propriété déductible à partir du graphe des marquages accessibles qui, dans notre cas, montre que notre modèle *RdPA* est bien vivant.

Etat d'Accueil

Dans les conversations entre Agents, nous ne souhaitons pas nécessairement qu'un protocole d'une conversation puisse revenir à l'état initial du dialogue. Par contre, nous souhaitons souvent qu'une conversation puisse revenir à un état donné, que nous appelons état d'accueil, pour des raisons comme par exemple les multiples cycles d'une négociation lors d'une vente aux enchères. Dans notre cas, nous avons pris cette hypothèse dès le début pour modéliser une instance de protocole de *FIPA-Request*. Cependant, nous avons indiqué lors de la modélisation les états d'accueil possibles dans notre modèle, essentiellement dans le cas d'échec de la communication. Ainsi, il est toujours possible de reprendre la communication à partir de ces états jusqu'à la réalisation du but de l'Interaction.

Persistance

En termes de protocoles, un Réseau de Petri persistant correspond à un dialogue dont il ne subsiste pas de transitions conflictuelles ou de prise de décision.

Concrètement, lors d'un appel d'offre *Contract Net*, les multiples franchissements de la transition de réception *Proposal* ne doit pas empêcher l'occurrence d'une transition du traitement prise de décision par le modérateur, une fois le délai d'attente des réponses est dépassé.

Le Réseau de Petri à Agent de protocole *FIPA-Request* est un réseau persistant. En effet, une transition peut être franchie plusieurs fois si les règles du protocole le permettent. Cela est modélisé par le modèle *RdPA* de *FIPA-Contract Net* dans le cas où l'Agent modérateur ayant reçu plusieurs propositions (**Figure 35**).

D'autres propriétés selon le domaine d'application peuvent être vérifiées aussi longtemps qu'elles sont concernées par des états d'Agents et de leurs comportements.

Il est aussi possible d'utiliser les méthodes de simulation automatique qui constituent un outil appréciable pour vérifier la dynamique des protocoles : observer l'évolution d'un protocole au cours du temps, étudier la performance du système et vérifier que le protocole se comporte comme prévu.

4.5.5.3. Synthèse

Les propriétés introduites ci-dessus ont une grande importance si on s'intéresse au Réseau de Petri, au Système Multi Agents, ou à la complexité du réseau. Le Réseau de Petri à Agents décrit un flux de messages qui est, dans la plupart des cas, fini.

Nous pouvons imaginer l'analyse d'un Réseau de Petri à Agents dans le contexte d'un flux de message unique ou en présence d'un générateur des messages représentant tous les flux. Dans ce cas, nous pouvons nous intéresser au caractère borné du réseau et, éventuellement l'accessibilité des états, mais pas à la vivacité. D'autre part, la vivacité sera souvent plus importante dans l'analyse du système global et dans l'étude de ses propriétés dynamiques.

Considérons, par exemple, le modèle de protocole *FIPA-Request*. Compte tenu de certaines séquences de messages, nous pourrions être intéressés à l'accessibilité; dans le cas où la question qui se pose est : est-ce que le marquage initial du réseau permet d'avoir un comportement correct de notre réseau ?

Parmi les propriétés qui peuvent être utiles ici, nous pourrions désigner les marquages finaux ou de terminaison (succès, échec), puis en les désignant comme membres de l'espace d'états, déterminer si toutes les séquences de tests conduisent à ces états objectifs.

4.6. Conclusion

Dans un Système Multi Agents, nous nous sommes intéressés plus aux propriétés dynamiques. Ceci est dans le sens où la vérification de la vivacité nous garantit que le système n'est pas entré dans un état où aucune action ne pourra se produire et l'absence du blocage nous assure qu'aucun élément du système qui est en mesure d'agir ne restera inactif ou bloqué indéfiniment.

Ce sont les propriétés qui nous assurent d'être engagé dans une activité autonome et d'identifier de l'espace d'états et surtout des états objectifs. Ceci nous permet de déterminer si un *SMA* a réussi ou non à atteindre son objectif de l'Interaction.

En tenant compte de la bonne spécification des types de données et des actions, les protocoles ainsi modélisés peuvent être analysés avant leur implantation et guider très tôt certains choix de conception grâce à l'analyse structurelle et dynamique. Même si la démarche de vérification/validation formelle est la seule façon de garantir l'absence d'erreurs, le problème reste la capacité limitée des outils présents sur le marché qui impliquent l'impossibilité de généraliser cette méthode à grande échelle. Certes, ce contrôle peut être largement utile en présence des outils variés de validation. Néanmoins, certaines entités dynamiques peuvent engendrer un comportement indésirable lors de leurs déplacements. Ceci est connu par le processus de *Migration*.

5

Chapitre 5 : Modèle de Migration des Agents Mobiles

La vérification formelle d'un système est une tâche difficile. La complexité est davantage dans le cas d'un système ouvert et dynamique. Dans le cadre de la vérification des Systèmes Multi Agents, la mobilité des Agents est considérée comme étant un facteur fondamental pour exprimer la dynamique du déroulement des actions. Ceci explique le fait que la vérification du processus de migration permet de bien contrôler les entités dynamiques et de définir clairement leur emplacement (Environnement de départ et celui d'arrivé). Dans ce chapitre, nous définissons un nouveau modèle de Migration pour les Agents Mobiles par le biais de Réseaux de Petri à Agents. Le modèle explique la phase de migration des Agents d'un Environnement à un autre.

5.1. Problématique

Les travaux déjà réalisés dans le domaine des *SMA* n'ont pas considéré la vérification de migration des Agents d'une façon intégrée dans le cycle de développement. La plupart des techniques formelles n'ont pas traité la vérification du processus de Migration d'un Agent Mobile. Les travaux de (EI Fallahi, Haddad, & Mazouzi, 2001) et (Celaya, Desrochers, & Robert, 2009) ont présenté une vérification du Modèle général du *SMA*, en particulier, la communication des Agents au sein du même Environnement et non la vérification du processus de migration explicitement.

Dans la modélisation et la simulation de (Moldt & Weinberg, 1997) le modèle de Migration été défini comme étant un outil séparé d'aide pour la plateforme accueillant les Agents. Cet outil n'est pas concrétisé dès le départ avec la totalité du modèle, mais il a été défini séparément. Certaines plateformes d'Agents Mobiles tels qu'*OMG*, *RePast* et *Masson* fournissent le mécanisme de migration mais n'offrent pas sa politique de migration. (Bouzid, Chevrier, & Vialle, 2003) ont proposé un modèle de simulation parallèle basé sur la répartition des conflits survenant entre les Agents, et sur un équilibrage dynamique de charge entre les processeurs par le biais de politique dite « auto-partition dynamique ». Ceci engendre un contrôle insuffisant du comportement de l'Agent et des changements qui lui sont apportés au cours de Migration. Récemment, des travaux font appel à des techniques de vérification pour les applications distribuées. En effet, (Bouali, 2009) a distingué deux niveaux de vérification ; le premier est du côté structurel en examinant les architectures des Agents Mobiles. Le deuxième niveau vérifie les Environnements d'accueil des Agents. Cependant, (Chen, David, Linz, & Cheng, 2008) ont déjà cherché à vérifier la sécurité de Agents visitant des sites différents. En effet, le transfert de code dit « *codeAgent* » peut subir des changements indésirables lors de la migration. Pour résoudre ce problème, les chercheurs ont proposés une technique basée sur la multi vérification coté site envoyant le code et coté site accueillant le code. (Jiannong, Feng, Jian, & Sajal, 2002) et (Bouchoul & Mostefai, 2012) ont montré que la vérification de la migration est intégrée dans le processus de communication. Ceci engendre une ambiguïté pour distinguer les comportements générés localement (au sein du même Environnement) et celle à l'extérieur (inter-Environnement). (Cauiya, 2007), a exprimé une continuité de l'Environnement exécutant des Agents Mobiles. Ceci ne donne pas une « liberté » pour

qu'un Agent choisi son prochain Environnement et le modèle ne se construit pas en fonction des perceptions et des actions actuelles.

Plusieurs travaux dans la littérature tels que, (Huget, 2001), (EI Fallahi, Haddad, & Mazouzi, 2001), (Sibertin-Blanc, Cardoso, & Hanachi, 2001) et (Chen, David, Linz, & Cheng, 2008) ont proposé des techniques qui guident le concepteur dès la spécification jusqu'à la validation d'un *SMA*. Tandis que, le processus de Migration est resté toujours non vérifié qu'au niveau de l'implémentation. (Murata, Nelson, & Yim, 1991) ont proposé un algorithme pour la construction d'un modèle predicat/transition pour les opérations robotiques. Cependant, ce modèle considère que l'interaction entre les Agents est indirecte, lors de leurs migrations, ce qui augmente le nombre d'états d'une manière combinatoire. Ce problème a été soulevé dans les travaux de (Xu, Volz, Loeger, & Yen, 2002). (Leitao, Colomb, & Restivo, 2004) ont proposé un modèle en *RdP* pour le contrôle d'un système de fabrication distribué. Ce modèle favorise une vérification des propriétés liées au fonctionnement de chaque entité du système dans son propre Environnement, mais il n'offre aucun moyen pour analyser la relation Agents avec les autres Environnements. Plus récemment, (Koning & Hernández, 2003), (Pouyan & Reeves, 2004) et (Celaya, Desrochers, & Robert, 2009) modélisent la mobilité des entités composant un système à l'aide de Réseaux de Petri.

Nous constatons à travers ce tour d'horizon sur les méthodes de modélisation et de vérification l'absence d'un modèle efficace qui traite la migration au niveau des *SMA*. L'un de nos objectifs de recherche est de concevoir le dit modèle. Notre travail s'intéresse à la vérification du modèle Migration dans un *SMA*. En fait, nous réalisons une décomposition structurelle du modèle *RdPA*. L'idée est de construire une Interface dite « Interface de Migration », qui favorise une description par modèle de relation entre un Agent Mobile, son Environnement de départ et son Environnement d'arrivé. Pour le contrôle du comportement de chaque Agent, chaque Environnement, et donc la totalité du système, nous appliquons la théorie du modèle Influence/Réaction. En considérant les problématiques déjà indiqué, nous soulevons les deux questions suivantes : comment modéliser la migration des Agents ? Et comment valider ces principales propriétés du modèle après vérification ?

5.2. Modélisation de la Migration d'Agents

L'utilisation des Agents Mobiles actuellement est fortement recommandée. Mais, ceci engendre des problèmes de sécurité, de confidentialité, de gestion et de suivi. Nombreux travaux de développement des plateformes d'Agents Mobiles sont commercialisés, notamment *Aglet, Voyager*, *Odyssey*, *Netlogo*, *Swarm, Mason* et *RePast*. Ces études se focalisent sur le développement d'un Environnement d'Agents Mobiles. Elles concernent la création des Agents, la transmission et la communication locale. Mais la mobilité se traduit dans l'implémentation directe sur cet Environnement en utilisant des multiples tâches (threads). La plupart de ces plateformes ne garantit pas la transmission du vecteur d'état des Agents avant et après la migration puisque la reprise d'architecture doit être réalisée par le concepteur (Treuil, Drogoul, & Zu, 2008). D'autre plateformes des Agents Mbiles décrit la migration d'une manière directe par un code, généralement orienté objet tels que *MADKIT*, *JADE*, *ZEUS* et *JNA*. Malgré que les Agents partagent certaines caractéristiques au niveau comportemental, il est difficile de transformer un modèle d'une plateforme à une autre. Ceci est dû à l'absence d'un modèle générique indépendant de l'implémentation du code.

Nous proposons un modèle (Pattern) à base des *RdPA* décrivant d'une manière simple et claire les composants intervenants lors de Migration (Agent Mobile, Environnement de départ et Environnement d'arrivé). Ces composants évoluent dynamiquement en fonction de liaison entre eux.

Nous proposons l'application du Modèle Influence/Réaction pour décrire les changements comportementaux de ces déférents composants. Rappelons toutefois que ce modèle été proposé par (Ferber & Müller, 1996). Il cherche à séparer « ce qui est produit par les Agents (influence) de ce qui se produit effectivement (réaction) » résultant du couplage des influences. Dans (Fabien , 2007) ce modèle a été utilisé pour la vérification des actions des Agents par simulation. En effet, dans le domaine des *SMA*, pour vérifier puis valider certaines propriétés, le moyen le plus utilisé est la simulation dans une plateforme dédiée. Les systèmes à base d'Agent Mobile sont des

systèmes ouverts d'où la difficulté de simuler des Environnements de différentes natures. En conséquence, le degré de certitude à une telle validation reste insuffisant. Ceci favorise donc une validation formelle. Des modèles en Réseau de Petri ont été souvent menés pour vérifier l'aspect concurrent. Nous distinguons ce type des modèles dans les travaux en domaine de biologie tels que (Pinney, Westhead, & DcConkey, 2003) et (Caouiya, 2007). Malgré l'utilisation des Réseaux de Petri dans nombreux travaux de vérification des *SMA*, beaucoup d'eux n'ont pas clairement montré les aspects liés à la migration des Agents.

5.3. Vérification Formelle des Systèmes Multi Agents

Les méthodes formelles bénéficient d'une littérature très riche et variée selon le système cible et son domaine d'application. Notre travail est focalisé, particulièrement, sur le processus de migration des Agents Mobiles. L'objectif donc est de fournir un modèle décrivant le comportement du système, lors de cette phase, d'une façon formelle, précise et nom ambigüe. En effet, la complexité est introduite par la mobilité des Agents qui sont de divers types et peuvent produire des comportements non désirés. Ceci conduit à une vérification beaucoup plus difficile.

La vérification formelle est l'action de contrôler l'exactitude, c'est à dire démontrer la validité d'un système, contrairement à la vérification par la simulation. C'est une vérification exhaustive, basée sur une théorie mathématique. Elle permet de vérifier tous les cas de figure possibles de réalisation ou faire l'équivalence entre deux descriptions.

Rappelons toutefois, que dans notre étude nous intéressons à la vérification de migration des Agents à travers un modèle. La nature complexe d'une telle tâche dicte le besoin d'utiliser un formalisme ayant un grand pouvoir expressif pour modéliser, analyser, vérifier et valider les changements. Bien que divers langages formels aient été utilisés dans ce propos, nous avons opté pour les Réseaux de Petri à Agents qui ont l'avantage de représenter les notions de concurrence, de synchronisation, de modularité, de réutilisabilité et d'une facilité de conception pour les *SMA*.

5.4. Modèle RdPA de Migration

Le défi est de construire un cadre formel pour garantir la précision et la sûreté de la phase de modélisation puis la vérification du modèle conçu. Ce modèle décrit et contrôle la mobilité des Agents d'un Environnement à un autre.
Une démarche classique dans l'ingénierie des *RdP* consiste à commencer par la création du modèle, trouver le graphe de marquages, déduire le graphe de couverture (si nécessaire) et déduire les propriétés. Bien que la modélisation par un Réseau de Petri offre un mécanisme dynamique de contrôle des systèmes, un développeur non expert dans l'ingénierie des *RdP* trouve une difficulté pour la création du modèle. Pour cela, nous pouvons simplifier le processus de développement par la création d'une matrice dite : « Matrice de Migration », qui peut servir à l'initialisation de l'étape de Vérification.

Nous devons vérifier le marquage initial, le choix de ce dernier n'est pas arbitraire. En effet, nous vérifions tout d'abord qu'un Agent ne doit appartenir initialement à un et un seul Environnement. Ensuite, nous contrôlons son déplacement d'un Environnement à un autre.

Nous définissons le modèle de Migration des Agents Mobiles par le Réseau de Petri à Agents suivant : $R = <Env, MIG, A, Pré, Post, Z, L, Ei>$.

où :

- *Envj* : est un ensemble fini non vide de Places (Environnement de départ *Envd* ou d'arrivé *Enva*),
- *MIG* : est un ensemble fini non vide de Transitions (Migrations),
- *A* : est un ensemble fini non vide de Jetons (Agents Mobiles),
- *Pré : Envd* x *MIG* $\longrightarrow$ *N* une application d'incidence avant de l'Environnement de départ,

- *Post : Enva* x *MIG* $\rightarrow$ *N* une application d'incidence arrière de l'Environnement d'arrivé,
- *Z* : Influence d'Agents,
- *L* : Réaction d'Agents,
- *Ei* : variable état d'un Environnement.

Suite à cette définition nous concevons le modèle en *RdPA* par la **Figure 40** :

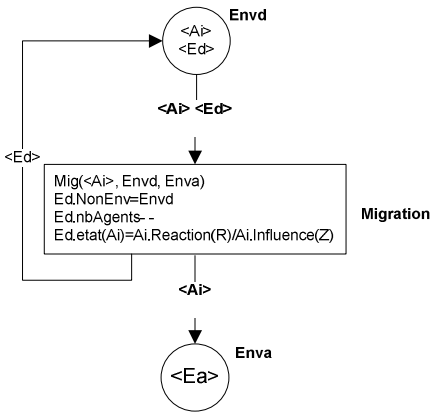


Figure 40. Modèle de Migration en *RdPA*

Analyse du Modèle

Le modèle que nous proposons est décrit comme suit :

- *Env= {Env1, Env2,..., Envk}* : ensemble fini non vide de Places ; chaque Place désigne un Environnement. Avec *k* : le nombre d'Environnements du système,
- Chaque Environnement *Envi* peut être un Environnement de départ(*Envd*) ou d'arrivé (*Enva*),
- *Mig* : ensemble fini non vide de Transitions ; chaque Transition désigne l'étape de Migration. Cette transition représente « une Interface » entre *Envd* et *Enva*,

- La migration d'un Agent *Aj* de l'Environnement *Envd* à un autre *Enva* se traduit par un franchissement de jeton *Aj* d'une place à une autre à travers une transition. Ceci est défini par : *Mig (Ai, Envd, Enva),*
- Pour chaque Environnement *Envi* on associe un jeton *Ei* qui décrit son état et toute information utile tel que :
  - *Ei. Nom_ Environnement,*
  - *Ei. Nombre_ Agents_ Mobiles,*
  - *Ei. GetInfluence(Aj),*
  - *Ei. GetReaction(Aj),*
  - *E. Etat(Aj),*
  - *Ei. Seuil(Aj),* donne la valeur max de satisfaction des actions d'Agent (rendement),
  - *Ei. Accessibility(Aj).*
- L'exploitation de *Ei* permet la vérification d'état de l'Environnement accueillant des Agents Mobiles :
  - au départ : *Ei. GetInfluence(Aj)= Aj. Influence(Z)* ; avec *Z* est l'ensemble d'actions à réaliser par l'Agent *Aj*,
  - après Migration : *Ei. GetInfluence(Aj)= Aj. Reaction(L)* ; avec *L* est l'ensemble d'actions réalisés par Aj pendant sa migration,
  - *Ei. état(Aj)= Aj. Reaction(L)/Aj. Influence(Z)*
  - Si *Ei. état(Aj)<=Ei. Seuil(Aj)* alors *Ei. Accessibility(Aj)=vrai*

La migration d'un Agent *Aj* fait un changement en 3 niveaux :

- *Niveau 1* : Environnement de départ par la mise à jour de sa variable d'état *Ed*,
- *Niveau 2* : Agent Mobile par la réalisation des actions *Aj. Reaction(L)* à partir d'*Aj. Influence (Z),*
- *Niveau 3* : Environnement d'arrivé par la mise à jour de son variable d'état *Ea.*

Donc, un Environnement peut subir *3* changements :

- *Changement 1* : lorsque un ou plusieurs Agents le quitte,
- *Changement 2* : lorsque les Agents sont en réaction dans l'Environnement,
- *Changement 3* : lorsque un ou plusieurs Agents le rejoindre.

Notre modèle est capable de distinguer ces *3* niveaux et ces *3* changements par le biais de Places qui indique le niveau *2* et changement *2*, par les arcs reliés en amant des Transitions indiquant le niveau *1* et changement *1* et par les arcs reliés en aval des Transitions indiquant le niveau *3* et changement 3.

5.4.1. Matrice de Migration

Nous définissions une Matrice dite « Matrice de Migration » présenté par la **Figure 41**. Cette matrice décrit d'une manière simple les Environnements de départ en fonction de ceux d'arrivé lors de Migration des Agents.

	Enva	*Env1*	*Env2*	-	-	-	*Envk*
Envd	*Env1*	*0*					
	Env2		*0*				
	-			*0*			
	-				*0*		
	-					*0*	
	Envk						*0*

Figure 41. Matrice de Migration

- C'est une Matrice carré qui présente en ligne les Environnements de départ et en colonne les Environnements d'arrivé,
- La dimension de la matrice est celle le nombre d'Environnements *k* du système,
- Le diagonale de la Matrice est toujours rempli par des valeurs nulles (l'Environnement de départ est celui d'arrivé : pas de migration).

5.4.2. Vérification de Matrice de Migration

Mathématiquement, un élément *Aj* doit exister sur une seule ligne de la Matrice de Migration. Ceci se traduit par : *Fonction verif_Agent(M, k, Aj)=vrai* ; vérifie qu'à

un moment donné, un Agent *Aj* existe dans un seul Environnement parmi les *k*-Environnements.

```
Type Matrice=Array [1 ..nmax, 1..nmax]of Agents ;
Function verif_Agent(M : Matrice ; k: integer. Aj: Agent): boolean;
Var x,i,c :intger ;
Begin
x:=0 ;
for i := 1 to k do
for c :=1 to k do
if M[i, c]= Aj then x :=x+1 ;
if x>1 then verif_Agent :=false
else verif_Agent := true ;
End;
```

Pour la vérification de totalité des Agents, nous proposons la fonction :

verif_ Mig (M, k)=vrai ; vérifie que chaque Agent de l'ensemble existe dans un seul Environnement.

```
Type vect_Agent=Array [1..nmax]of Agents ;
Function Verif_Mig(M :Matrice ; k :integer ;VA : vect_Agent) : boolean ;
Var i: integer ;
Begin
i :=1 ;
While (Verif_Agent(M, k, Aj)=true) and(i<=k) do
i :=i+1 ;
If i>k then Verif_Mig:=true
 Else Verif_Agent:=false;
End;
```

Pour vérifier l'interface de Migration pour un Agent *Aj*, nous devons savoir les Environnements en amont et en aval : *Procedure recherche_ Env(M, k, Aj)=(i, c)*; donne, respectivement, l'indice d'Environnement de départ(*Envd*) et celui d'arrivé (*Enva*)pour un Agent *Aj*.

```
Procedure recherche_Env(M :Matrice ; k :integer ; Aj : Agent; vari,c :integer ) ;
Begin
If Function Verif_Agent(M, k, Aj)= true then
```

```
 Begin
i :=0 ;
      Repeat
           i :=i+1 ;
          c:=1 ;
            Repeat
               c:=c+1 ;
            Until (M [i, c] = Aj) or (c>k);
       Until (M [i, c] = Aj) or (i>k);
End;
End;
```

5.4.3. Vérification des Propriétés du Modèle de Migration

Différents niveaux d'abstraction peuvent être considérés. D'une part, on peut valider des aspects propres aux Agents et des aspects d'Interaction entre les Agents. D'autre part, on peut s'intéresser à des abstractions du système et valider ses propriétés globales. Dans tous les cas, il est fondamental de définir, à ces différents niveaux, des sémantiques claires et prouvables afin d'envisager la spécification formelle de *SMA* et sa vérification.

La propriété du système représente donc, implicitement, une connaissance que nous souhaitons pouvoir décrire et qui doit pouvoir être vérifiée sur le modèle. Ceci permet de rassurer la cohérence et la pertinence du système attendu avant de créer un prototype.

Dans notre thématique de vérification d'un *SMA*, nous avons privilégié les Réseaux de Petri à Agents vu qu'ils offrent la possibilité de prouver et de vérifier formellement certaines propriétés du comportement du système cible. Ces propriétés sont regroupées en deux groupes. Le premier groupe concerne des propriétés de la structure du *RdPA* et son marquage initial. Ces propriétés sont globales et doivent, en général, être vérifiées pour que le même système modélisé par les *RdPA* puisse être implémenté sans qu'il engendre des problèmes de conflits lors de son exécution. C'est pourquoi, on les appelle "bonnes" propriétés ou propriétés de structure. Le second groupe concerne la possibilité de décomposer un *RdPA* en un sous *RdPA* particulier qui peut ensuite être

utilisé pour exprimer des propriétés spécifiques concernant soit les marquages accessibles, soit les scénarios du comportement (propriétés du comportement).
Les Réseaux de Petri à Agent permettent d'exprimer les contraintes appliquées sur le système, de vérifier des propriétés statiques et dynamiques concernant le comportement. Pour un modèle *RdPA,* on distingue deux sortes de propriétés : celles qui sont relatives à l'état du système, se basant sur le nombre des jetons qui circulent dans le réseau, et celles qui sont relatives à son évolution. Les jetons représentent les Agents. Chaque Place du modèle symbolise un état d'un Environnement, la Transition indique le déclenchement d'un évènement de Migration.

A partir du graphe de marquages accessibles nous déduisons les propriétés suivantes :

5.4.3.1. Réseaux bornés

A travers cette propriété, on cherche à montrer la possibilité pour une place d'accumuler une quantité bornée ou pas de jetons au cours de l'évolution d'un réseau.
Dans chaque place le nombre de jetons ne dépasse pas la valeur na qui présente le nombre d'Agents dans le système.
Chaque place est bornée par la valeur na respectant la règle suivante :

$$na\text{-}bornée \leftrightarrow \forall M \in A(R, M0),\ M(p) \leq na$$

- Toutes les places de *RdPA* sont bornées, donc le réseau lui-même est caractérisé borné, d'où le graphe de couverture et le graphe des marquages sont identiques. Cette propriété de borniture est déductible même sans exécution du réseau.
- Le système restera stable durant son fonctionnement et nous ne pouvons pas avoir une situation de débordement de capacité du système par un nombre infini d'Agents.
- Nous ne pouvons pas avoir un franchissement de deux ou plusieurs transitions en même temps par un seul Agent (un Agent migre vers un seul Environnement à un instant donné).

- Le modèle de Migration ne peut pas être sauf (unaire) où chaque place contient au maximum un jeton (vu qu'un Environnement peut accueillir plusieurs Agents).
- Nous ne pouvons pas avoir des places avec un nombre infini d'Agents s'accumulant dans certaines circonstances. Les Agents formant un système sont, déjà, connus à l'avance.

5.4.3.2. Couverture

Cette propriété concerne l'évolution du nombre de jetons au cours de l'exécution des transitions. Elle teste si ce nombre augmente au cours de temps ou non, et l'évalue par rapport au nombre de jetons initial dans *M0* :

- Le nombre d'Agents et de la liste des Environnements sont gardés constants et stables, et le système peut donc bien s'adapter avec ses Environnements dynamiques.
- Un *SMA* composé de *na* Agents réalisant m processus de Migration doit exécuter notre modèle *m* fois. Donc nous pouvons calculer son temps total d'exécution.

5.4.3.3. Invariants

La notion de l'invariant consiste à avoir un nombre d'Agents qui est constant dans tout le réseau. Elle cherche à affirmer si un modèle admet une composante conservative ou répétitive. Ceci à travers le Vecteur de pondération : *VP= (na, nai, pk)*

où :

na : le nombre total d'Agents,

nai : le nombre d'Agents initial,

pk : l'ensemble de Places.

- Le modèle est conservatif puisque le nombre d'Agents présenté dans les différentes places est différent du nul et il est constant.
- Tout au long de franchissement des transitions, le graphe de marquages ne présente jamais un vecteur de pondération nul.
- L'Environnement peut être changé après chaque Migration.

5.4.3.4. Activité d'un Réseau

La notion d'activité d'un réseau couvre deux classes de définitions. La première concerne l'activité individuelle des transitions. La seconde concerne l'activité globale d'un réseau (indépendamment de transitions particulières). Cette activité est gérée par la quasi-vivacité et par la vivacité.

5.4.3.4.1. La Quasi-vivacité

Cette propriété signifie que, depuis le marquage initial cette transition peur être franchie au moins une fois. Autrement dit, pour un réseau marqué (R, M0) :

$$t \in T \text{ quasi-Vivante}, M \in A(R, M0), M \rightarrow t$$

- Pour le modèle de Migration toutes les transitions possèdent une séquence de transitions permettant de la franchir.
- Cette propriété permet de garantir que n'importe quel état atteint par le système est généré par l'événement de Migration donc le modèle ne comporte pas de branches mortes.
- L'utilisation de ce modèle favorise une continuité entre les Environnements à travers la migration des Agents d'une manière bien contrôlée.

5.4.3.4.2. Vivacité

Cette propriété cherche à vérifier qu'une transition est toujours franchissable quel que soit les transitions déjà franchies. La construction du modèle de migration est basée sur

la vérification de certaines contraintes et conditions qui doivent être satisfaites pour tirer une transition.

5.4.3.4.3. Blocage

On aura la notion de blocage que si à un instant bien déterminé on peut jamais franchir une transition du réseau dans le sens où il existe une ou plusieurs places qui ne sont pas à l'origine d'aucun arc. Nous pouvons affirmer les caractéristiques suivantes :

- Chaque place peut avoir des arcs en entré et d'autres en sortie. Ceci implique que chaque Environnement peut être *Envd* ou bien *Enva*.
- Chaque place donne lieu à au moins une transition qui sera ensuite franchie.
- A tout instant une transition peut être franchie, et chaque place de notre modèle est d'origine d'au moins un arc, d'où nous pouvons affirmer l'absence de blocage dans le modèle.
- La propriété de non blocage augmente la réactivité de la part des Agents et amplifie la coopération entre les Environnements. Le non blocage favorise la cohérence et la robustesse du système, de telle sorte qu'on assure la bonne gestion de ressources aux bons moments et on évite les comportements inattendus.

5.5. Etude de la Gestion de Transport Maritime

Pour valider l'approche proposée nous appliquons le modèle de Migration pour la gestion du problème de transport maritime. Notre objectif est de modéliser le problème de Gestion de Transport Maritime (GTM) par un SMA et le RdPA afin de contrôler le passage des bateaux d'un Port à un autre.

Nous souhaitons modéliser, vérifier et valider le fonctionnement du système (**Tableau 7**) composé par :

- *2* bateaux *B1* et *B2* qui embarquent d'un Port à un autre,
- *4* Ports : *Port1*, *Port2*, *Port3* et *Port4* qui accueillent les conteneurs embarqués sur l'un des bateaux *B1* ou *B2*,
- Des conteneurs qu'on veut les déplacer d'un Port à un autre suivant les demandes suivantes :

conteneurs	Port de départ	Port d'arrivé	Bateau
C1	Port1	Port2	B1
C2	Port1	Port3	B1
C3	Port1	Port4	B1
C4	Port2	Port4	B1
C5	Port1	Port4	B2

Tableau 7. Exemple de Répartition des Conteneurs

Nous pouvons déduire la Matrice de Migration suivante (**Figure 42**)

	Port d'arrivé	*Port1*	*Port2*	*Port3*	*Port4*
Port de départ	*Port1*	*0*	*<B1>*	*0*	*<B2>*
	Port2	*0*	*0*	*<B1>*	*0*
	Port3	*0*	*0*	*0*	*<B1>*
	Port4	*<B1> <B2>*	*0*	*0*	*0*

Figure 42. Matrice de Migration de GTM

Un bateau *Bi* doit exister sur une seule ligne de Matrice de Migration, ceci se traduit par : *verif_ Agent(Ports, 4, B1)=true* ; vérifie qu'à un moment donné, le bateau *B1* existe sur un seul *Port* parmi les *4* Ports du système.

De même pour le bateau *B2* : *verif_ Agent(Ports, 4, B2)=vrai*. Pour la vérification de la totalité des bateaux, nous proposons : *verif_ Mig(Ports, 4)=vrai* ; vérifie que chaque bateau de l'ensemble existe dans un seul Port.

Pour vérifier l'interface de Migration pour un bateau *B1*, nous devons savoir les Ports de départ et d'arrivé : *recherche_ Env(Ports, 4, B1 , Port1, Port2)=(1, 2)*.

Si initialement *Mo (<B1><B2>, 0, 0, 0)* alors les ports visités par le bateau *B1* et *B2* sont donnés respectivement par :

- *Mig(B1)= Port2, Port3, Port4 et Port1*,
- *Mig(B2)= Port4 et Port1*.

Inversement, on cherche les bateaux qui visitent les Ports par :

- *Env(Port1)=B1 et B2,*
- *Env(Port2)=B1,*
- *Env(Port3)=B1,*
- *Env(Port4)=B1 et B2.*

Dans le modèle de migration en *RdPA* (**Figure 43**), les bateaux *B1* et *B2* sont deux Agents Mobiles. Les Ports représentent les Environnements accueillant les bateaux.

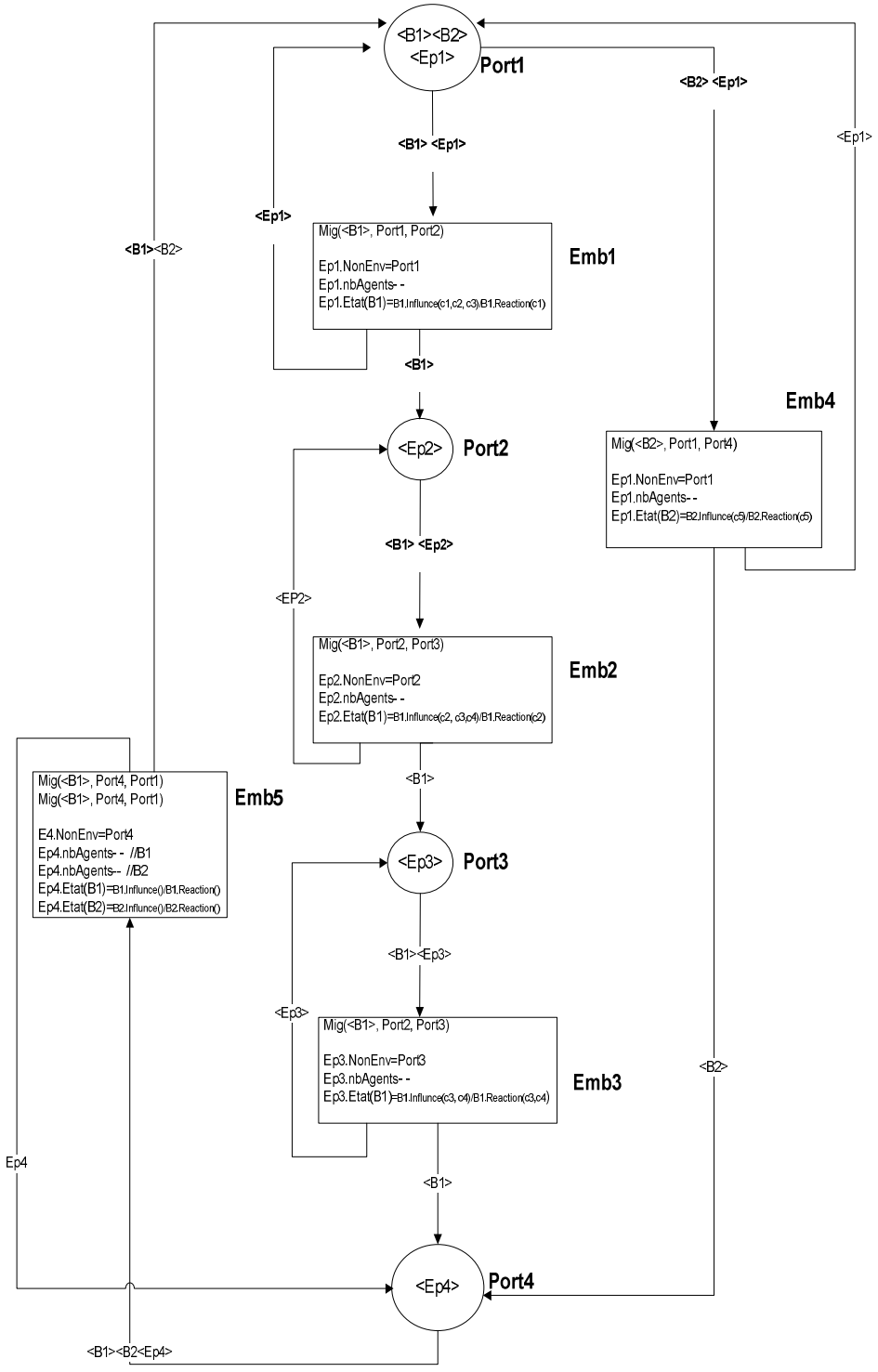


Figure 43. Modèle de Migration des Bateaux

Analyse du Modèle Influence/Réaction :

Port1----------

Ep1.Nom_Environnement=Port1

Ep1.GetInfluence(B1)=Embarque (c1, c2, c3 ; Port2)

Ep1.GetInfluence(B2)=Embarque (c5 ; Port4)

Ep1. Get.Reaction (B1) = Débarque (c1; Port2)

Ep1. Get.Reaction (B2) = Débarque (c5; Port4)

Ep1.Etat(B1)= Débarque (c1, c2, c3; Port2)/ Embarque (c1 ; Port2)

Ep1.Etat(B1)= Débarque (c5; Port4)/ Embarque (c5 ; Port4)

Ep1. Accessibility(B1)=vrai

Ep1. Accessibility(B2)=vrai

Port2----------

Ep2.Nom_Environnement=Port2

Ep2.GetInfluence(B1)=Embarque (c2, c3, c4 ; Port3)

Ep2. GetReaction(B1)= Débarque (c2; Port3)

Ep2.Etat(B1)= Débarque (c2; Port3)/ Embarque (c2, c3, c4 ; Port3)

Ep2.accessibility(B1)=vrai

Port3----------

Ep3.Nom_Environnement=Port3

Ep3.GetInfluence(B1)=Embarque (c3, c4 ; Port4)

Ep3. GetReaction(B1)= Débarque (c3, c4; Port4)

Ep3.Etat(B1)= Débarque (c3, c4; Port4)/ Embarque (c3, c4 ; Port4)

Ep3.Accessibility(B1)=vrai

Port4----------

Ep4.Nom_Environnement=Port4

Ep4.GetInfluence(B1)=Embarque (0;Port1)

Ep4. GetReaction(B1)= Débarque (0; Port1)

Ep4.GetInfluence(B2)=Embarque (0;Port1)

Ep4. GetReaction(B2)= Débarque (0; Port1)

Ep4.Etat(B1)= Débarque (0; Port1)/ Embarque (0 ; Port1)

Ep4.Etat(B1)= Débarque (0; Port1)/ Embarque (0 ; Port1)

Ep4.Accessibility(B1)=vrai Ep4.Accessibility(B2)=vrai

Vérification des propriétés du Modèle de Migration des Bateaux :

Nous construisons le Graphe de Marquages Accessibles (**Figure 44**)

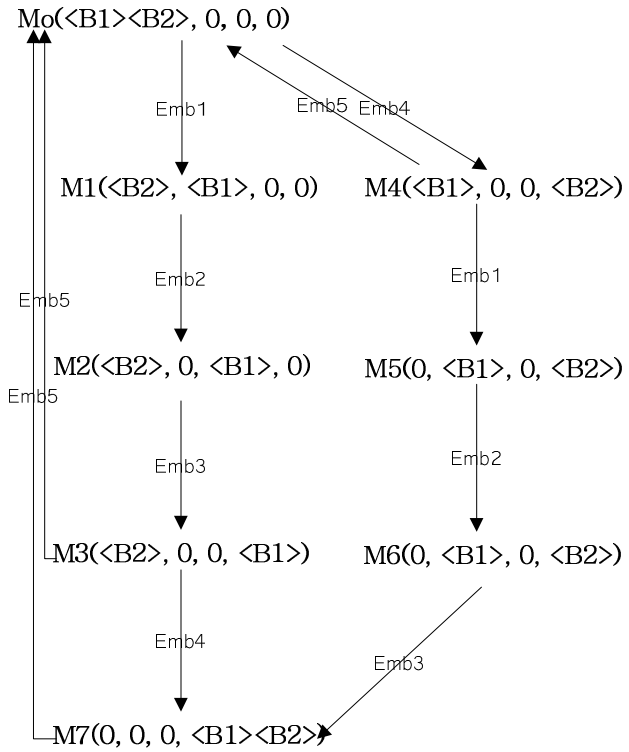


Figure 44 . Graphe de Marquages de Modèle de Migration des Bateaux

A partir du graphe des marques accessibles nous déduisons les propriétés suivantes :

- Le *RdPA* est *2*-bornées. En effet, quelques soit la migration des bateaux *B1* et *B2*, le modèle montre qu'un Port ne peut accueillir au maximum que *2* bateaux,

- Le marquage montre qu'un Bateau à un instant donné ne peut exister que sur un seul Port,
- Le modèle n'est pas *sauf* parce qu'un Port peut accueillir les deux bateaux en même temps ($M0$ *et* $M7$),
- L'évolution du nombre de jetons dans les places est bien contrôlée. Ceci dépend du plan de Migration. Cette couverture montre la stabilité du système,
- Le nombre d'Agents Mobiles est constant donc le modèle est invariant,
- Le modèle est conservatif puisque le nombre des bateaux présenté dans les différents Ports est différent du nul,
- Tout au long de la Migration, le graphe de marquages ne présente jamais un vecteur de pondération nul,
- Chaque transition est quasi vivante, d'où tout le réseau est quasi vivant,
- Le réseau est vivant,
- Le réseau est sans blocage.

5.6. Simulation de la Migration d'un Agent Mobile : cas d'un Système de Gestion du Transport Maritime(SGTM)

5.6.1. Introduction

L'avènement du conteneur a totalement changé le monde du transport maritime. En très peu de temps, les compagnies ont changé de méthode de travail et de matériel pour investir dans de nouveaux navires coûteux et dans des flottes de porte-conteneurs. La tendance actuelle est à la concentration : une concentration financière qui se traduit par des fusions-absorptions ou des prises de contrôle et une concentration technique pour éviter une surcapacité des porte-conteneurs.

Avant d'expliquer comment nous pouvons éviter cette surcapacité à l'aide de notre système de gestion maritime $SGTM$, il faut définir le porte-conteneurs et citer ses

caractéristiques. Un porte-conteneurs est un navire destiné au transport de conteneurs à l'exclusion de tout autre type de marchandises. Apparu dans les années 1970, le porte-conteneurs est maintenant le principal mode de transport maritime dans les ports de commerce.

D'après (Batra, 2012) en juillet 2011, il y avait 12015 navires sur les lignes commerciales représentant une capacité de 42.500.000 EPV (l'unité de mesure de la conteneurisation est devenue l'EVP = Équivalent Vingt Pieds (TEU = Twenty Équivalent Unit). Les conteneurs des différentes dimensions sont ramenés, par équivalence, à des conteneurs de 20'. Ainsi, un conteneur de 20'correspond à 1 EVP. Un conteneur de 40'correspond à 2 EVP. Dans le « jargon » du transport international, un conteneur est souvent nommé une « boîte ». Partout, les réseaux de transport maritime doivent restructurer leurs systèmes de gestion pour faire face aux changements imposés par la globalisation de l'économie afin de pouvoir répondre à la demande des entreprises aux réseaux globaux émanant de la fusion des multinationales. Cette restructuration des $SGTM$ impose une remise en cause de l'approche managériale de la compagnie aux niveaux stratégique, organisationnel et opérationnel.

5.6.2. Contrainte de Mobilité des Agents

Notre recherche se focalise sur l'étude des points suivants :

- *La granularité de la Mobilité* : la migration d'un Agent implique d'abord de savoir quoi déplacer. Il faut connaître les frontières de l'Agent (variables simples, les objets qui ont une relation de composition avec l'objet migré, les liens avec les objets périphériques, etc.).
- *La désignation des objets mobiles* : il est important avant, après et durant la migration des Agents de trouver un mécanisme de désignation global qui permet d'identifier un Agent de façon unique dans un Environnement distribué.

- *La localisation de l'Agent* : la mobilité des Agents nécessite une transparence totale par rapport à la localisation. En d'autres termes, les Agents doivent être capables de contacter un objet sans besoin de connaître sa localité. Elle est traditionnellement assurée par un service de désignation qui maintient une liaison entre un nom symbolique d'un Agent et une ou plusieurs adresses auxquelles l'Agent peut être contacté. Mais, ceci engendre un problème de gestion des groupes d'Agents. Donc, il faut que l'Agent Mobile mémorise les différentes localités des autres Agents.
- *La communication entre les Agents Mobiles* : la migration d'un Agent communicant pose le problème du maintien des canaux de communication avec d'autres Agents. De ce fait, les références des Agents doivent être mises à jour rapidement et la référence que l'Agent possède sur l'Agent migré doit être mise à jour de façon transparente.

5.6.3. Motivations

Tous les problèmes précédents influent directement sur le domaine de conteneurisation. Ces problèmes évoquent une croissance importante de conteneurisation (à peu près de 8.5 % par ans selon les dernières statistiques) (Batra, 2012). Ceci conduit à un problème de décision tel que :

- Gestion de conteneurs (exploitation d'une manière efficace),
- Nombre maximal de conteneurs qui peuvent être chargés sur le navire,
- La durée d'exploitation sur les différents ports.

Donc, il est important de développer des méthodes qui assurent l'équilibrage de charge sur le navire et l'utilisation optimale de ressources disponibles. C'est pour ça nous essayons de trouver des solutions avec notre *SGTM*.

Parmi les motivations de notre travail, nous citons :

- Nous pouvons étudier les comportements globaux d'un Système Multi Agents. Plus précisément, nous pouvons bien observer l'Agent Mobile (le navire dans notre cas) qui s'interagit avec l'Environnement qui le perçoit (le port) et s'interagit avec tous les éléments de *SMA* ce qui nous donne une idée précise sur l'état global de comportements du *SGTM*.
- Grace à ce *SGTM*, nous pouvons ainsi étudier les comportements de chaque navire et les différents ports. Lors du lancement de simulation, nous pouvons étudier les comportements de l'Agent Mobile (navire) lorsqu'il arrive à un nouveau port pour débarquer ou embarquer des conteneurs sur les différentes zones.

En outre, nous pouvons suivre instantanément le processus de migration du navire lorsqu' il migre d'un port à un autre. De plus, l'administrateur du *SGTM* prévu les comportements avant le fonctionnement réel. Ceci lui permet de minimiser les coûts et l'utilisation optimale de ressources disponibles. De même, le *SGTM* donne une vision globale sur le déroulement de processus de transport de conteneurs pour bien choisir le nombre de conteneurs et leurs charges convenables pour éviter la défaillance de la mission du navire. Nous pouvons prévoir les mauvais états dont l'objectif de diminuer les risques probables comme la surcharge du navire. Enfin, l'administrateur de *SGTM* a une idée sur les charges de différents ports de telle façon il connaît les différents types de conteneurs et leurs poids qui se trouvent sur les suivants ports. Ceci qui lui permet de bien choisir le nombre de conteneurs à embarquer ou à débarquer sur les différents ports.

5.6.4. Description du Système

Nous disposons d'un navire ou d'un porte-conteneurs qui fait déplacer les conteneurs d'un port à un autre. Le navire suit toujours la même trajectoire. Cette trajectoire est donnée par le Comité de la Sécurité Maritime afin de localiser toujours tous les navires surtout dans les cas des risques ou dans les cas les bateaux ou les

navires tombent en panne. Pour simplifier, nous considérons que chaque conteneur a un type, une destination et un poids qui le caractérise.

Nous utilisons deux types de conteneurs : « Les conteneurs fragiles » (ce sont des conteneurs qui contiennent des matériaux fragiles.) et « Les conteneurs non fragiles ». Chaque conteneur a une capacité ou un poids de chargement qui varie selon son volume. Nous pouvons, aussi, classer les conteneurs selon leur poids : « Les conteneur lourds » ce sont les conteneurs qui ont un poids supérieur ou égale à trois tonnes et « Les conteneurs non lourds » ce sont les conteneurs qui ont un poids inférieur à trois et demi tonnes (c'est un choix puisque le poids d'un conteneur de vingt pied est égal à 2.2 tonnes). Nous obtenons alors quatre types de conteneurs :

- les conteneurs qui sont « lourds » et « fragiles » (L-F),
- les conteneurs qui sont « non lourds » et « fragiles » (NL-F),
- les conteneurs qui sont « lourds » et « non fragiles » (L-NF),
- les conteneurs qui sont « non lourds » et «non fragiles » (NL-NF).

Dans tous les ports, il y a quatre zones pour débarquer ces différents types de conteneurs déjà cités. Le problème est donc comment nous faisons la gestion de problème de débarquement et d'embarquement de conteneurs sur les ports?

Nous avons respecté le modèle donné par le standard $MASIF$ d'OMG. Le navire est l'Agent Mobile, il peut se déplacer vers des différents Environnements. Le patch du monde *Nelogo*[2] représente la place où l'Agent perçoit et interagit avec son Environnement pour exécuter une action par exemple. En outre, il y a quatre agences dans notre $SGTM$: ce sont les quatre ports représentés par quatre rectangles avec des couleurs différentes. Toutes ces agences appartiennent à une même région qui est le monde *Netlogo*.

[2] http://ccl.northwestern.edu/netlogo/

NetLogo est un environnement de simulation des SMA

Nous disposons alors d'un navire qui contient des types différents de conteneurs et qui se dirige vers un port. Lorsque le navire arrive à un port, il débarque tous les conteneurs qui ont le numéro de ce port comme un port de destination et il pointe instantanément (grâce à un pointeur qui se trouve à chaque port) tous les conteneurs qui sont débarqués sur ce port. Ensuite, le navire change sa direction et il se dirige vers le suivant port en embarquant tous les conteneurs qui se trouvent dans la zone d'embarquement. Sachant que le navire a une capacité de conteneurs qu'il ne faut pas la dépasser. Lorsque le navire débarque tous les conteneurs sur les différents ports, il s'arrête sur le prochain port après un certain temps. A la fin de cette description il faut dire que tous les paramètres de cette simulation sont modifiables par l'administrateur de *SGTM* et ils sont enregistrés avec les résultats obtenus pour une future utilisation. Par exemple, pour la comparaison avec d'autres résultats de simulations afin de prendre des bonnes décisions. Pour réaliser notre système, nous suivons les étapes du processus de simulation donné par (Drogoul, 1995). En effet, nous avons modélisé ce problème en utilisant un Système Multi Agents : le navire est l'Agent Mobile qui perçoit son Environnement et exerce une action lors de son interaction avec les autres Agents.

5.6.5. Expérimentations

Dans cette section, nous effectuons une série de tests en enregistrant des mesures spécifiques afin de visualiser et d'analyser l'effet de la variation de certains paramètres dont on pourrait citer :

- Nombre de « conteneurs fragiles » sur un port,
- Nombre de « conteneurs fragiles » sur le navire,
- Nombre de « conteneurs fragiles » sur le navire,
- Nombre de « conteneurs non fragiles » sur un port,
- Capacité du navire.

Initialement, nous supposons que tous les paramètres d'entrée ont des valeurs nulles. Notons qu'il y a toujours trois conteneurs sur le navire qui ont comme destination les ports suivants : le port numéro *1*, le port numéro *3* et le port numéro *4*. Ce sont des conteneurs utilisés généralement pour garantir que le navire débarque au moins un conteneur sur le port qui le visite et pour bien étudier le processus de migration d'Agent Mobile. De plus, il faut noter que la simulation débute lorsque le navire est sur la mer et il se dirige vers le deuxième port.

Cas 1 : Blocage du processus de migration

Les paramètres de simulation

Pour le premier cas, nous allons seulement régler les paramètres d'entrée ci-dessous pour initialiser la simulation.

Nb_Max_Cont = 100	*Cont_Non_Fragiles_Port2 = 15*	*Dest_B_F = 4*
ConteneursF_bat = 25	*Cont_Fragile_Port3 = 50*	*Dest_B_NF =4*
ConteneursNF_bat = 22	*Cont_Non_Fragiles_Port3 = 40*	*Port_dest_3F = 1*
Cont_Fragile_Port2 = 10	*Port_dest_3NF = 4*	*Port_dest_2NF = 4*
Port_dest_2F = 4		

Les résultats obtenus sont :

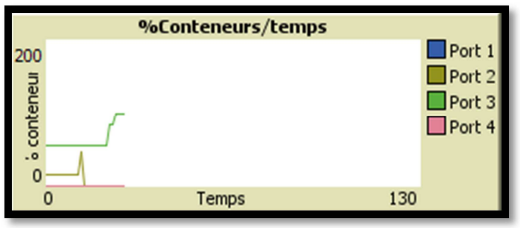


Figure 45. Les Conteneurs qui Passent par les Ports

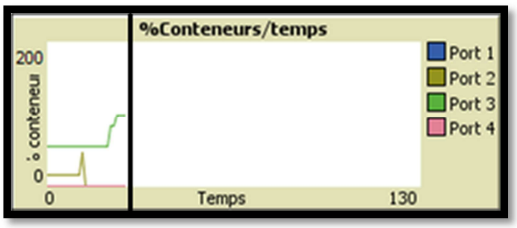


Figure 46. Arrêt de Processus de Migration

Figure 47. Localisation du Navire

Figure 48. Arrêt de Processus de Migration sur le Schéma de Localisation du Navire

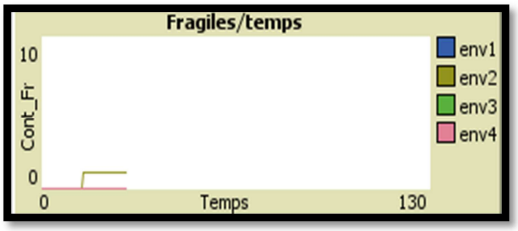


Figure 49. Les Conteneurs "Fragiles" qui sont Débarqués sur les Ports

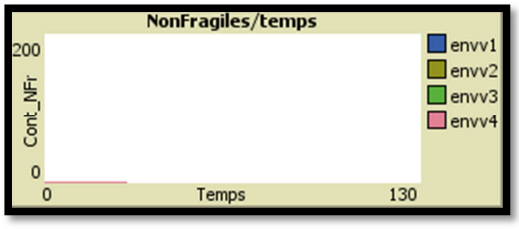


Figure 50. Les Conteneurs "Non Fragiles" qui sont Débarqués sur les Ports

Figure 51. Les Conteneurs qui ne sont pas encore Débarqués

Figure 52. Arrêt de Processus de Simulation

Analyse des résultats

A partir de la **Figure 45**, nous remarquons une faible augmentation de la courbe qui représente le nombre de conteneurs qui passent par le deuxième port (conteneurs sur le navire avec les conteneurs non embarqués sur ce port). Ensuite, nous observons une concentration importante de conteneurs qui passent par le troisième port: à peu-prés 97% de conteneurs qui ne sont pas encore débarqués se trouvent sur ce port. Par contre ce n'est pas le cas pour le quatrième port puisque il n y a aucun conteneur qui passe par ce dernier.

Il faut mentionner que lorsque la valeur sur l'axe de temps est égale à *30 ticks* (le tick c'est l'unité de mesure de temps dans le simulateur Netlogo), le processus de migration s'arrête : c'est le cas de la **Figure 46** (nous supposons qu'un tick est égal à trois jours par exemple). Nous pouvons voir sur les **Figures 47** et **48** que la simulation s'arrête lorsque la distance entre le navire et le troisième port (avec la couleur vert) égal à *0* c'est-à-dire le navire est sur le troisième port.

Pour le schéma de conteneurs « fragiles » (**Figure 49**), il y a un seule conteneur débarqué sur le deuxième port: l'un des trois conteneurs que nous avons mis par défaut sur le navire est débarqué sur ce port. Pour la **Figure 50** de conteneurs « non fragiles » nous n'avons pas choisi des conteneurs « non fragiles » sur les ports ou sur le navire c'est pour cela il n'y a aucune courbe dans cette figure. Les **Figures 51** et **52** montrent qu'il y a *163* conteneurs qui ne sont pas encore débarqués avant que le processus de migration s'arrête.

Nous constatons d'après les résultats obtenus que le nombre de conteneurs sur le navire dépasse sa capacité (*Nb_Max_Cont)* qui égale à *100* conteneurs. Il y a *163* conteneurs c'est-à-dire il y a une surcharge de *63* conteneurs sur le navire. C'est pour cela que le processus de migration du navire s'arrête et le programme affiche un message d'avertissement qui indique le nombre probable de conteneurs qui sont en

surcharge (**Figure 53**) puisque il s'agit d'un état de risque. Il faut alors changer soit la capacité du navire soit diminuer le nombre de conteneurs qui ont le quatrième port comme un port de destination.

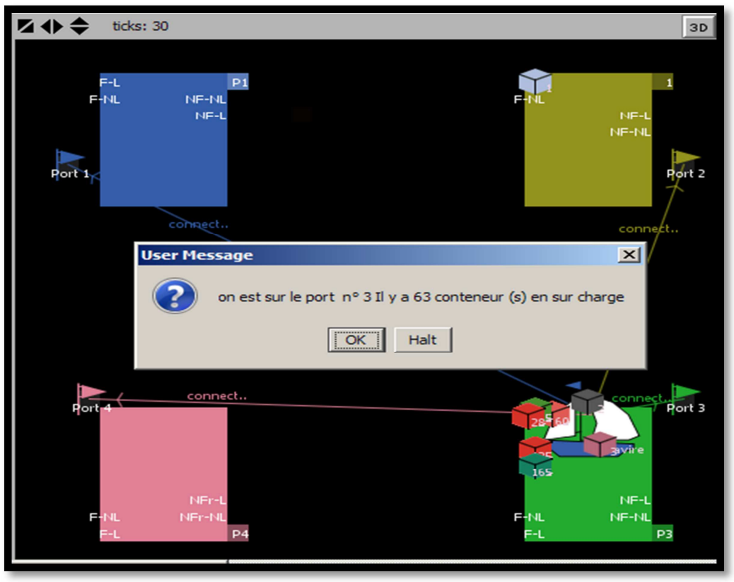


Figure 53. Résultat de la Simulation : Cas de Blocage

Cas 2 : Déroulement Normal du Processus de Migration

Les paramètres de simulation

Dans ce cas, nous gardons les mêmes paramètres d'entrés mais nous modifions la capacité du navire pour qu'il soit capable d'embarquer tous les conteneurs sur le troisième port. Nous changeons les poids de quelques conteneurs. Ceci est pour donner une idée sur les zones de débarquements de conteneurs et mettre l'accent sur un déroulement normal de la simulation et sur le processus de migration du navire en particulier. Les paramètres de *SGTM* sont changés comme suit :

Nb_Max_Cont = 200 *Poid_2F = 1.1* *Poid_3F = 0.5*
Poid_2NF = 3.3 *Poid_3NF = 4.7*

Les résultats obtenus sont :

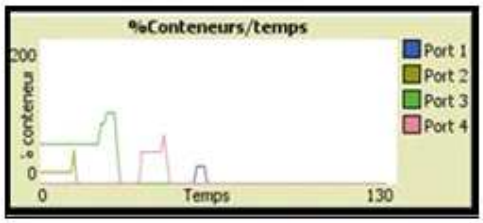


Figure 54. Les conteneurs qui Passent par les Ports

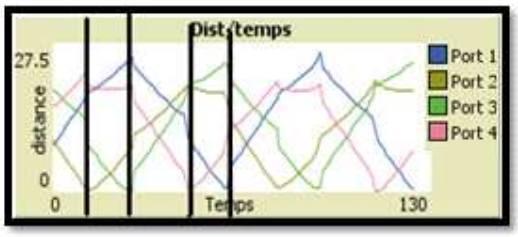


Figure 55. Arrêt de Processus de Migration

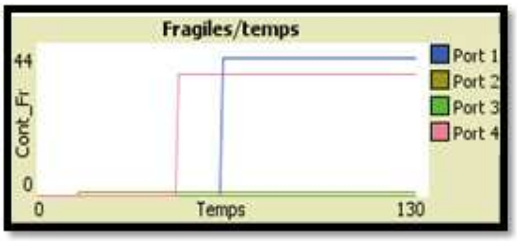


Figure 56. Les Conteneurs "Fragiles" qui sont Débarqués sur les Ports

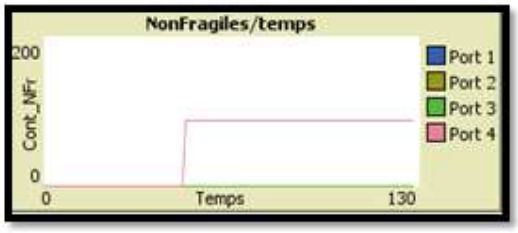


Figure 57. Les Conteneurs "Non Fragiles" qui sont Débarqués sur les Ports

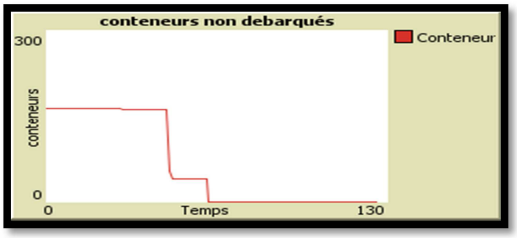


Figure 58. Les Conteneurs qui ne sont pas Débarqués sur les Ports

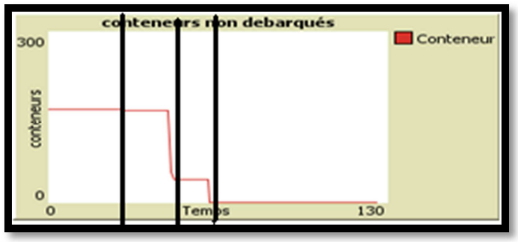


Figure 59. Les Conteneurs qui ne sont pas Débarqués sur les Ports

Analyse des résultats

La **Figure 54** présente le pourcentage ou la concentration de conteneurs qui passent par les différents ports au cours de la simulation : nous remarquons que les conteneurs s'accumulent sur le troisième port car la pluparts des conteneurs ont le quatrième port comme un port de destination et il faut qu'ils passent par ce dernier. Puis lorsque le navire sort du troisième port (*tick = 30)* et arrive au quatrième port, nous observons une chute de pourcentage de conteneurs qui passent par le deuxième port. Ce phénomène dû au nombre de conteneurs qui sont déjà débarqués sur le troisième port.

La **Figure 55** explique convenablement comment nous localisons le navire dans notre *SGTM* : lorsque l'une des courbes qui représente un port (selon sa couleur) a la distance égale à zéro. Ceci veut dire que le navire est sur ce port. La **Figure 56** montre que lorsque le navire est sur le quatrième port il y a une forte augmentation du nombre de conteneurs fragiles débarqués sur ce port. De plus, nous remarquons que lorsque le navire arrive au *Port1* il y a une forte augmentation de courbe associée au nombre de conteneurs « fragiles » débarqués sur ce port. Pour la **Figure 55**, nous remarquons que lorsque le navire est sur le quatrième port il y a une forte augmentation de nombre de conteneurs « non fragiles » débarqués sur ce port. C'est tout à fait évident puisque la plupart de conteneurs «non fragiles » ont le quatrième port comme un port de destination (88 conteneurs). Au début de simulation, nous avons 163 conteneurs qui ne sont pas encore débarqués (**Figures 58** et **59**). Lorsque le navire arrive au deuxième port nous pouvons remarquer une faible diminution de nombre de conteneurs non embarqués. Ceci est dû à un débarquement d'un seul conteneur sur ce port c'est le même cas pour les autres ports jusqu'à ce que tous les conteneurs sont débarqués.

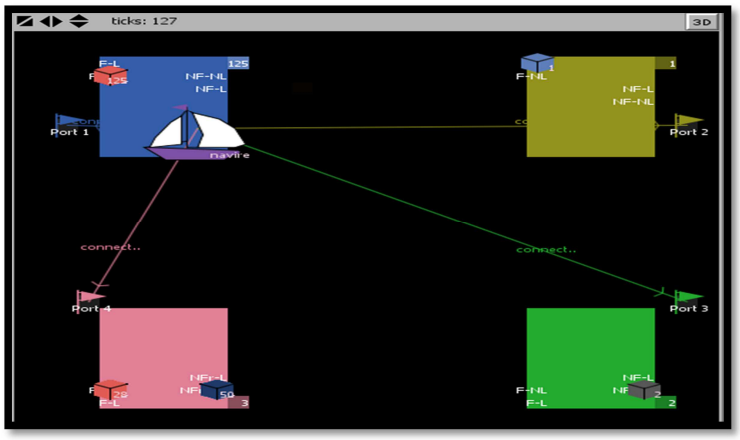


Figure 60. Résultat de la Simulation du cas : Déroulement Normale

La **Figure 60** montre que le processus de migration du navire se déroule convenablement car nous augmentons la capacité du navire pour qu'il soit capable de transporter le nombre de conteneurs sur les ports.

Cas 3. Exemple complexe de simulation

Les paramètres de la simulation

Dans ce cas, nous essayons de compliquer la simulation en testant notre *SGTM* avec un nombre très élevé de conteneurs de plusieurs types et avec différents ports de destination. Nous allons régler les paramètres d'entrée comme suit :

Nb_Max_Cont = 400	*Poid_2NF = 4.1*	*Port_dest_1F = 2*
cont_Fragile_Port1 = 35	*Poid_3NF = 2.6*	*Port_dest_2F = 1*
cont_Fragile_Port2 = 24	*Poid_4NF = 4.7*	*Port_dest_3F = 4*
cont_Fragile_Port3 = 39	*Poid_1F = 1.1*	*Port_dest_4F = 3*
cont_Fragile_Port4 = 50	*Poid_2F = 5.0*	*Port_dest_1NF = 4*
Cont_Non_Fragiles_Port1 = 50	*Poid_3F = 3.4*	*Port_dest_2NF = 3*
Cont_Non_Fragiles_Port2 = 24	*Poid_4F = 3.6*	*Port_dest_3NF = 2*
Cont_Non_Fragiles_Port3 = 50	*ConteneursF_bat = 50*	*Port_dest_4NF = 1*
Cont_Non_Fragiles_Port4 = 25	*ConteneursNF_bat = 17*	*Poid_1NF = 3.3*
Poid_B_NF = 1.8	*Dest_B_F = 2*	
Poid_B_F = 1.8	*Dest_B_NF = 1*	

Les résultats obtenus sont :

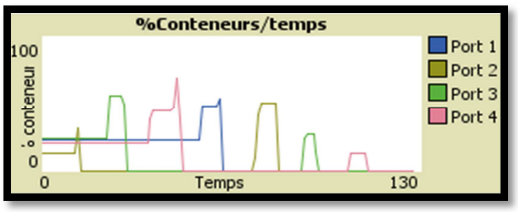


Figure 61. Les Conteneurs qui Passent par les Ports

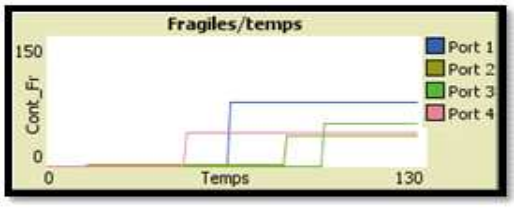


Figure 62. Les conteneurs "Fragiles" qui sont Débarqués sur les Ports

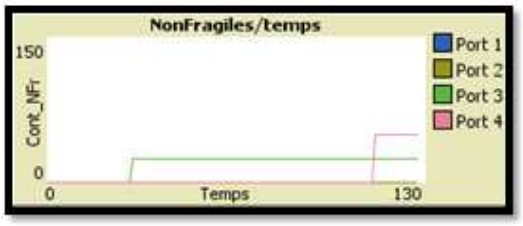


Figure 63. Les Conteneurs "Non Fragiles" qui sont Débarqués sur les Ports

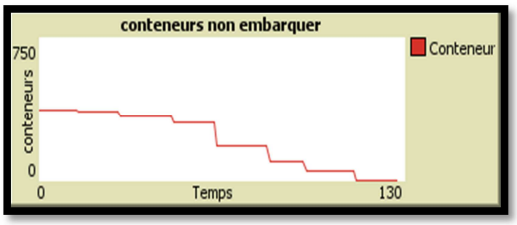


Figure 64. Les Conteneurs qui ne sont pas Débarqués sur les Ports

Analyse des résultats

La **Figure 61** représente les pourcentages des conteneurs qui ne sont pas encore embarqués et qui sont sur la zone de l'un de quatre ports. Nous remarquons qu'à chaque migration du navire à un port, il y a une augmentation de taux de conteneurs sur ce port par contre il y a une chute de courbe qui représente le nombre de conteneurs lors de départ. La **Figure 62** montre que la plupart de conteneurs utilisés sont de conteneurs « fragiles » parce que nous observons que les conteneurs « Fragiles » sur les quatre ports ont des valeurs importantes. Par contre, ce n'est pas le cas pour les conteneurs «Non Fragiles» qui sont représentés dans la **Figure 63** puisque il n'y a que quelque conteneurs sont débarqués sur le Port3 et le Port4. La **Figure 64** représente le nombre de conteneurs qui ne sont pas encore débarqués : Chaque décroissement de courbe correspond à une migration du navire d'un port à un autre.

Malgré l'augmentation de nombre de conteneurs et l'utilisation de tous les types de conteneurs avec des ports de destination variés, le *SGTM* a réussi à donner des résultats pour aider les décideurs à propos de ce problème (**Figure 65**).

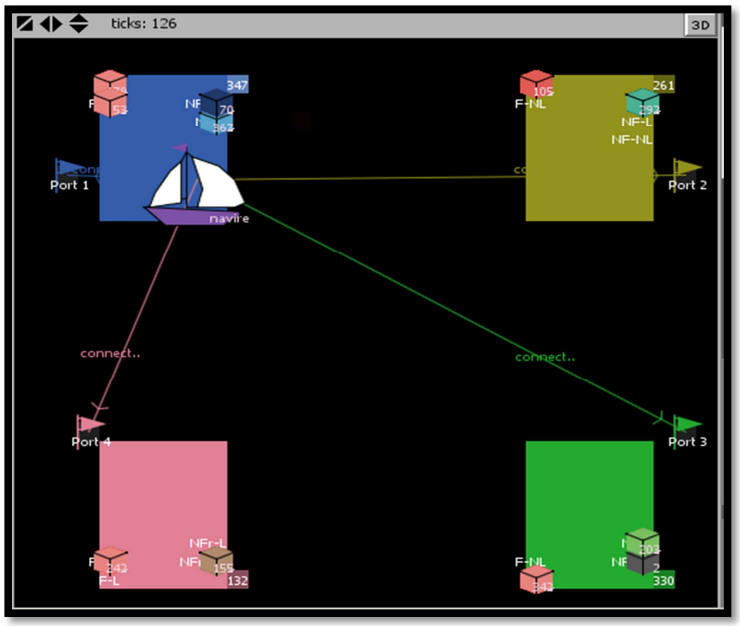


Figure 65. Résultat de la Simulation du Cas 3

Nous constatons aussi que le processus de migration du navire vers les différents ports se déroule convenablement pendant toute la simulation malgré que nous utilisions tous les types disponibles de conteneurs embarqués et débarqués sur les différents ports.

5.6.6. Validation de SGTM

Après la phase d'expérimentation, nous pouvons conclure que notre *SGTM* a atteint plusieurs objectifs de la simulation à base d'Agent. Parmi les objectifs nous citons :

- *La validation* : notre système est capable de valider un déroulement normal de processus de chargement et de déchargement de conteneurs sur les ports après l'exécution de la simulation.
- *L'évaluation* : le *SGTM* est capable d'évaluer la simulation et de prédire les mauvaises situations.
- *La communication* : le navire se communique avec les différents éléments de *SGTM*.

- La visualisation : le *SGTM* visualise toutes les étapes de migration de navire ainsi son comportement lors de sa migration d'un Environnement à un autre.
- Le contrôle: grâce au *SGTM* l'administrateur est capable de contrôler le système de gestion de conteneurs.
- La prévision et la prédiction : ce sont des éléments primordiales dans tout les *SGTMs* et c'est le même cas pour notre propre système puisque l'administrateur prévoit le ca des risque de surcharge de navire par exemple dès qu'il lance la simulation.
- Le pilotage : le superviseur peut contrôler l'état interne de l'Agent Mobile (le nombre de conteneurs sur le navire ou la direction de ce dernier).

5.6.6.1. SGTM et Mobilité

Suite à la simulation effectuée pour le *SGTM*, nous pouvons recenser les principales propriétés qui caractérisent un Agent Mobile(Navire) :

- Le navire est un Agent réactif. En effet, il reçoit les ordres du superviseur pour embarquer les conteneurs d'un port à un autre : ces actions doivent être assez précises.
- Le problème de localisation forme un contrôle continu sur les ports donc l'Agent doit être capable de percevoir la totalité des Environnements. La perception d'Environnement est un caractère principal pour que les Agents réagissent convenablement avec l'Environnement visité.
- Vue la sensibilité des tâches du navire, il doit se comporter d'une manière prudente vis-à-vis les ordres de superviseur.
- Au cours de la mobilité, le navire doit être capable de contrôler son état interne (sa charge). Donc il peut avertir le superviseur concernant des situations non prédictibles. Ceci peut augmenter la stabilité et la sureté du système.

- Durant le processus de migration, les actions de l'Agent navire sont déjà planifiées. Ceci augmente le bon déroulement de ces taches à réaliser : le processus de chargement et de déchargement des conteneurs sur les ports de *SGTM*.
- L'Agent doit déterminer attentivement sa route vers le prochain site car toutes fautes de la direction de l'Agent engendrent des perturbations du système.

5.6.6.2. Interface de Gestion de Résultats de SGTM

Nous remarquons que les schémas générés par le simulateur *Netlogo* ne donnent pas assez de détails surtout lorsque nous les utilisons en dehors du simulateur. Ainsi nous ne pouvons pas enregistrer ni les paramètres ni les schémas pour une future utilisation. *Netlogo* ne permet pas de comparer les résultats de deux simulations sur la même interface. Considérons par exemple le cas de deux ou trois résultats qui sont générés avec des différents paramètres comme nous avons dans la précédente section de ce chapitre. C'est pour cela, nous implémentons une application qui a comme rôles :

- Enregistrer tous les paramètres et les schémas de simulations que le superviseur souhaite l'utiliser ultérieurement,
- Produire des résultats et des schémas plus clairs que ceux donnés par *Netlogo*,
- Importer des anciens résultats de simulation pour les comparer avec des autres qui sont plus récents.

Pour implémenter cette application, nous utilisons les fichiers d'extension « *.CSV* »générés par le simulateur *NetLogo*. C'est un format de fichier (Comma Separated Values) dont les informations sont séparées par une virgule.

Nous avons réussi à récupérer tout le contenu utile de ce fichier qui nous aide à récupérer les paramètres d'entrée de simulation ainsi une description des schémas. Ensuite nous présentons ces données dans une simple *IHM* qui est utilisé par

l'administrateur de *SGTM* qui contient tous les paramètres d'entrée utilisés et tous les résultats obtenus. Cette interface (**Figure 66**) permet de bien comparer les différents résultats obtenus de simulation ainsi les paramètres d'entrée. L'administrateur de *SGTM* est capable aussi d'enregistrer ou d'imprimer les différents paramètres et les résultats obtenus pour chaque simulation. Finalement, cette interface donne des schémas plus clairs et plus significatifs que ceux donnés par *Netlogo*.

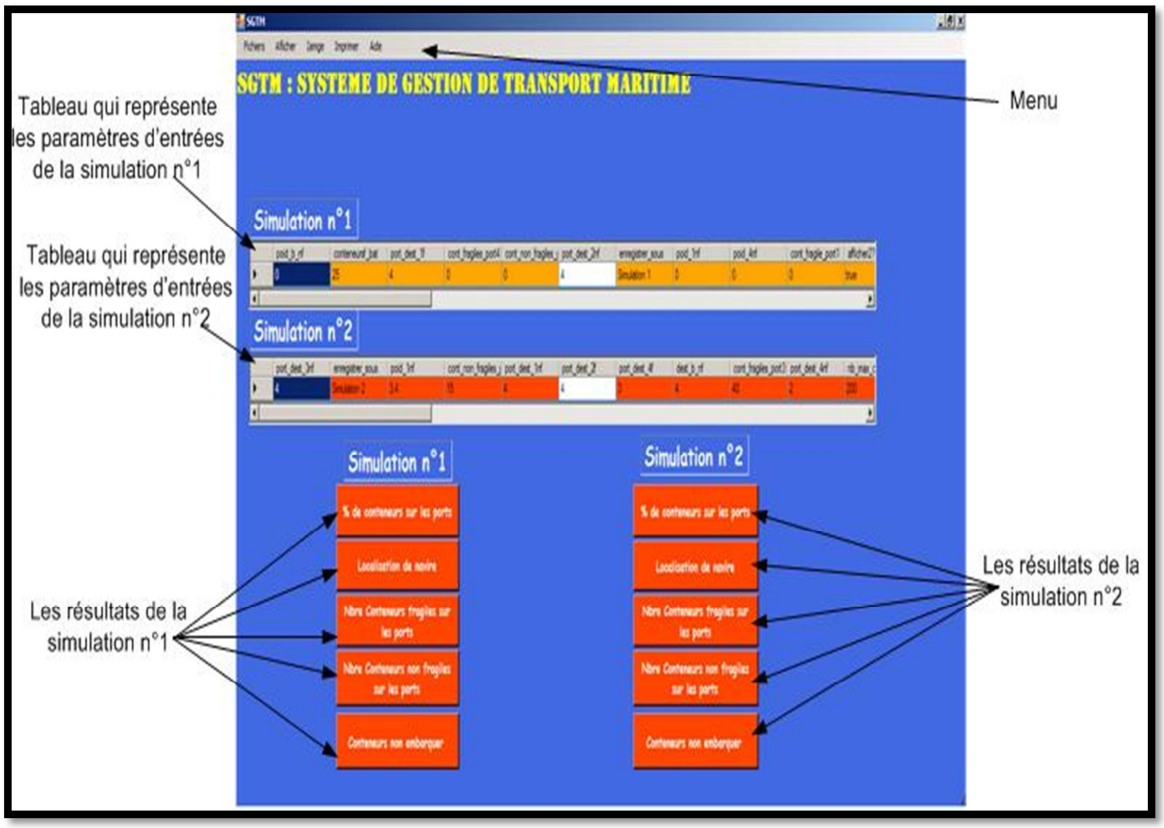


Figure 66. Interface de Gestion de SGTM

5.7. Conclusion

Le modèle que nous avons conçu offre la possibilité de modéliser, de vérifier et de valider d'une manière rigoureuse et efficace la migration des Agents d'un Environnement à un autre. En effet, deux niveaux de vérification sont présentés, le

premier est comportemental que nous avons appliqué à travers le modèle Influence / Réaction. Ceci garantie une fiabilité pour la réalisation des actions des Agents. Tandis que le deuxième niveau est structurel que nous avons assuré à travers la vérification de principales propriétés du modèle. Nous avons montré que le modèle de migration doit être toujours borné, non sauf, vivant et sans blocage. D'autres propriétés peuvent être duites suivant le cas d'étude tel que la présence d'état d'accueil et des séquences répétitives. Ce modèle peut servir pour créer une nouvelle propriété des *SMA* : des Environnements communicants à travers une interface standard.

Notre démarche s'appuie sur une validation par simulation. Cette technique permet d'approcher les conditions de l'expérimentation. Une fois le modèle est programmé, nous pouvons le faire tourner autant de fois que le nous voulons en faisant varier les paramètres. Concrètement, nous modifions les paramètres dont l'impact nous intéresse, on en observe l'effet sur le résultat final et on recommence l'opération. Aussi, cette technique permet de tester rapidement le changement de certaines hypothèses. En effet, nous avons simulé, à l'aide de *Netlogo*, un *SGTM*. Grace à ce simulateur nous avons réalisé plusieurs expérimentations qui nous aident à étudier les comportements du navire lors de sa migration d'un port à un autre.

Conclusion Générale et Perspectives

Le concept des Systèmes Multi Agents devient un domaine clé pour toute nouvelle technologie applicative. Le déroulement dynamique au sein de ce type de systèmes est basé sur l'Interaction et la Mobilité des entités qui le composent. Ce domaine a été porté par l'élan de l'évolution technologique constante que connait le monde de l'informatique actuellement, notamment, celle des systèmes distribués au cours des dix dernières années. Pour cela, définir un Système Multi Agents revenir, en fait, à la définition de l'entité base qui est l'Agent. Notre étude montre, clairement, que ce type des systèmes évolue avec l'évolution des Agents dans leur Environnement d'accueil. L'autonomie, la sociabilité, la réactivité et les compétences d'un Agent déterminent directement un Système Multi Agents. Certes, l'Agent doit coopérer et doit interagir avec d'autres Agents pour atteindre soit un objectif individuel, soit dans la plupart des cas un objectif commun. La nécessité des systèmes assez intelligents engendre, sans doute, la mise en place des techniques de Modélisation et de Vérification indispensables.

Face à ce problème, nous devons appliquer une démarche assez formelle. En fait, plusieurs techniques de modélisation et de vérification formelles ont été appliquées pour réduire l'ambigüité engendrée par la complexité croissante des *SMA*. Cependant, jusqu'à présent, il n'existe aucun formalisme qui satisfait à l'exigence complète de

Modélisation et de Vérification des *SMA*. Nous affirmons, sans doute, que les travaux de recherche antérieurs ont seulement couvert un aspect particulier des *SMA* et non la totalité de ses propriétés.

Pour remédier à cette insuffisance méthodologique, nous avons proposé un nouveau formalisme des Réseaux de Petri. Cette extension est à base d'Agents. Elle profite des caractéristiques des Agents et des Systèmes Multi Agents. En effet, chaque jeton d'une place représente un Agent et la transition est dotée d'un ensemble de fonctions qui décrit, en particulier, la condition de son franchissement et les relations entre les Agents. Nous avons appelé ce formalisme : Réseau de Petri à Agent(*RdPA*).

A travers nos travaux de recherche dans cette thèse, nous avons proposé, aussi, un modèle de spécification des protocoles de communication dans un Système Multi Agents. Notre objectif était d'appréhender les interactions entre Agents en proposant un modèle rendant leur modélisation plus simple. Il est incontestable que l'utilisation des protocoles d'Interaction pour les conversations facilite considérablement le développement des systèmes à base d'Agents communicants. Nous estimons que les limitations inhérentes à d'autres formalismes décrits dans la première partie du manuscrit rendent nécessaire l'utilisation d'un formalisme supportant la concurrence et la factorisation pour modéliser des Interactions aussi complexes et concurrentes.

Une telle spécification autorise la validation des protocoles d'Interaction par analyse de leurs propriétés, même si malheureusement, le pouvoir d'expression d'une notation s'accroît en général au détriment des possibilités d'analyse formelle des modèles.

Il est bien connu, par exemple, que certaines propriétés deviennent indécidables dès qu'un formalisme atteint un degré supérieur dans son pouvoir d'expression. Ainsi, l'ajout aux *RdPA* d'arcs inhibiteurs (qui permettent de tester l'absence de jeton dans une place) accroît sensiblement le confort de modélisation, mais réduit les possibilités d'analyse des réseaux. Ce compromis entre pouvoir d'expression et pouvoir d'analyse se retrouve d'ailleurs dans toute autre notation à caractère formel ou semi-formel.

Nous avons également proposé un modèle pour assurer la Migration d'Agent d'un Environnement à un autre. Nous nous sommes intéressés à la vérification du modèle Migration dans un *SMA*. En fait, nous avons réalisé une décomposition structurelle du modèle *RdPA*. L'idée été de construire une Interface dit « Interface de Migration », qui favorise une description par modèle de la relation entre un Agent mobile, son Environnement de départ et son Environnement d'arrivé. Pour le contrôle du comportement de chaque Agent, chaque Environnement, et donc la totalité du système, nous avons appliqué la théorie de modèle Influence/Réaction. A travers ce modèle, nous avons choisi la simulation de comportement des Agents Mobiles via un exemple concret : SGTM. Cette simulation construit une méthode non formelle pour validation de notre nouveau modèle.

Les perspectives des travaux présentés dans cette thèse suivent différents axes de réflexion et se situent dans différents contextes de recherche. Certains points restent ouverts à de futurs développements, tels que la paramétrisation des protocoles ; on peut avoir à modéliser des contraintes temporelles s'appliquant aux règles des protocoles. Par exemple, au cours d'une enchère, pendant combien de temps un manager est-il autorisé à attendre avant de réaliser une intervention ? Plus généralement, les modérateurs ont besoin de dates limites, possiblement floues, pour décider à quel moment mettre fin à une conversation inachevée. Autrement, on peut étendre notre modèle d'Interaction en intégrant un mécanisme de temporisation et une gestion d'exceptions pour éviter le blocage durant les conversations.

Partant de *RdPA* et étant donné son aspect purement formel, une deuxième perspective consiste à décrire ces différentes étapes en se servant de l'approche orientée modèle. Ainsi, une décomposition du fonctionnement de *SMA* en des sous-fonctions réutilisables exige la définition d'un formalisme permettant le passage à l'échelle systématiquement.

Finalement, nous chercherons à résoudre les problèmes liés aux contraintes temporelles. Nous enrichirons notre modèle par l'intégration de *RdP P*-Temporisation et de *RdP T*-temporisation. Cependant, nous devrons toujours vérifier si le modèle a concevoir est décidable ou non.

Références Bibliographiques

Abrial, J. (1996). *The B-Book, Assigning Programs to Meanings.* New York, USA: Cambridge University Press.

Adam, E. (2008). *Systèmes Multi Agents: Eléments Introductifs.* Université de Valenciennes et du Hainaut-Cambrésis, France.

Augeraud, M., Collé, F., & Sarramia, D. (2006). Conception Centre Interaction: Application à la Conception de la Simulation Interactive. *Actes du 6ème Congrès Informatique des Organisations et Systèmes d'Information et de Décision, INFORSID'06*, (pp. 86-97). Hammamet, Tunisie.

Barreteau, O. (1998). *Un Système Multi Agents pour Exploiter la Viabilité des Systèmes Irrigués : Dynamique des Interactions et Modes d'Organisation.* France: Ecole Nationale du Génie Rural des Eaux et des Forêts.

Barthélemy, F. (2005). *Les Systèmes Multi Agents Basés sur des Réseaux de Petri Colorés : Technique de Pointe ou Technologie Dépassée.* Namur, Belgique: Institut d'informatique des Facultés Notre-Dame de la Paix (FUNDP).

Batra, J. (2012). *Liner Shipping Strategy Review.* www.drewry.co.uk: The Independent Maritime Adviser.

Bernich, M., & Creteil , M. (2007). Software Management Based on Mobile Agents. *Second International Conference on Systems and Networks Communications* (pp. 64-76). Cap Estere, France: IEEE.

Bouali, M. (2009). *Contributions to the Formal Analysis and Diagnosis Using Colored Petri Nets With Rear Access.* France: Compiegne University of Technologies.

Bouchoul, F., & Mostefai, M. (2012). Formal verification of Mobile Agent Based Worflows. *International Conference on Information Technology and e-Services (ICITeS).* Souisse, Tunisia: IEEE.

Boussard, M., Bouzid, M., & Mouaddib, A. (2007). La Décision Multi Critère pour la Coordination Locale dans les Systèmes Multi Agents. *Modèle formels de l'interaction (MFI'07)*, 277-282.

Bouzid, M., Chevrier, V., & Vialle, s. (2003). Parallel Simulation of Stochastic Agent/Environnemnt Interaction Model. *Integrated Computer Aided Engineering, Vol. 3, N. 3*, 189-203.

Bratman, M., & Pollack, E. (1988). Plans and Resource Bounded Practical ReaSoning. *Computational Intelligence, Vol. 4*, 349-355.

Brooks, R. (1991). Intelligence Without Reason. *IJCAI'91 Proceedings of the 12th International joint conference on Artificial intelligence - Vol. 1* (pp. 569-595). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Burkhard, H. (1993). Liveness and Fairness Properties in Multi Agent Systems. *International Joint Conference on Artificial Intelligence IJCAI'93* (pp. 63-78). Chmbery, France: Distributed Al .

Cao, J., Feng, X., Lu, J., & Sajal, K. (2002). Design of Adaptive and Reliable Mobile Agent Communication Protocols. *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02).*

Caouiya, j. (2007). Petri Net Modeling of Biological Networks. *Brief.Bioinform V.8, N.4*, 210-219.

Celaya, J., Desrochers, A., & Robert, J. (2009). Modeling and Analysis of Multi Agent Systems Using Petri Nets. *Systems, Man and Cybernetics* (pp. 1439 - 1444). Montreal, Canada: IEEE.

Chaib-draa, B., Jarras, I., & Moulin, B. (2001). *Systèmes Multi Agents : Principes Généraux et Applications.* Paris, France: Hermès.

Chapurlat, V. (2008). *Vérification et Validation de Modèles de Systèmes Complexes: Application à la Modélisation d'Entreprise.* France: Université Montpellier II - Sciences et Techniques du Languedoc.

Charif, Y., & Sabouret , N. (2006). Protocole d'Interaction pour la Composition de Services dans l'Intelligence Ambiante. *14 th Journées Francophones sur les Systèmes Multi-Agents, JFSMA'06* (pp. 253-266). Annecy- France: Hermès.

Chen, B., David, D., Linz, D., & Cheng, ,. H. (2008). XML-based Agent Communication, Migration and Computation. *Journal of System and Software, V.81; N.8*, 1364-1376.

Devi, P., & Dutta, A. (2012). A Conceptual Graph Petri Net Model based Multi Agent System. *International Journal of Computer Applications, V.45- No.12*, 71-83.

Drogoul, A. (1995). When Ants Play Chess (or Can Strategies Emerge from Tactical Behaviors). *In Proceedings of Fifth European Workshop on Modelling Autonomous Agents in a Multi Agent World, MAAMAW '93* (pp. 13-27). Lausanne: Springer.

Durfee, E., & Montogomery, A. (1990). A Hierarchal protocol for Coordinating Multi Agent Behaviours. *Proc. of American Association for Artificial Intelligence(AAAI)* (pp. 86-93). Boaston, USA: AAAI Press.

EI Fallahi, A., Haddad, H., & Mazouzi, H. (2001). A Formal Study of Interactions in Multi Agent Systems. *International Journal of Computers and their Applications Vol. 8- N. 1*, 23-32.

El jed, M. (2006). *Interactions Sociales en Univers Virtuel: Modèles pour une Interaction Située.* France: Université Toulouse III : Paul Sabatier.

Esterline, A., & Rorie, T. (2007). Using the π-Calculus to Model Multiagent Systems. *Journal of Shanghai University, Vol.11, N. 1*, 58-63.

Ezzedine, H., & Kolski, C. (2008). Petri Net,Theory and Applications. In K. Vedran, *in, Use of Petr Nets for Modeling an Agent-Based Interactive System: Basic Principles and Case Study* (pp. 45-48). Vienna, Austria: I-Tech Education and Publishing.

Fabien , M. (2007). Le modèle IRM4S :utilisation du principe Influence/Réaction pour la simulation de systèmes multi Agents. *Journal of Artificial Intelligent, V.21*, 56-64.

Ferber, J. (1995). *Les systèmes Multi Agents:Vers une Intelligence Collective.* Paris, France: InterEdition.

Ferber, J. (1997). *Technologie Multiagents, Technique et Science Informatiques.* Paris, France: Vol. 16, No. 8, Hermès.

Ferber, J., & Müller, J. (1996). Influences and Reactions: a Model of Situated Multi Agents Systems. *Second International Conference on Multi-Agent Systems (ICMAS-96)* (pp. 72-79). Kyoto, Japan: AAAI Press.

Ferber, J., & Perl, C. (1991). Actors and Agents as Reflective Concurrent Objects: a Mering IV Perspective. *IEEE Transactions on Systems, Man and Cybernetics, Vol. 21, No. 6.*

Fougères, A. (2000). Une Architecture Cognitive d'Agents Communicants dans des Systèmes d'Information Complexes. *Modèles formels de l'interaction, Actes des Secondes Journées Francophones, MFI'03* (pp. 255--260). Lille, France: Cépaduès.

Fujita, M. (2009). Intelligence Dynamics: a Concept and Preliminary Experiments for Open-Ended Learning Agents. *Autonomous Agents and Multi Agent Systems, V.19, N. 3*, 248-271.

Gervais, F. (2004). *EB4 : Vers une Méthode Combinée de Spécification Formelle des Systèmes d'Information.* Québec, Canada: Université de Sherbrooke.

Giardini, F., & Amblard, F. (2013). Multi Agent Based Simulation XIII. *International Workshop MABS, Lecture Notes in Artificial Intelligence, Springer-Verlag, V.7838*, 91-115.

Gómez, J., Pavón, J., & Pavón, J. (2004). Methodologies for Developing Multi Agent Systems. *Journal of Universal Computer Science, V.10, No.6*, 359-374.

Gouardères, E. (2009). *Conception d'Applications à Base d'Agents.* Paris, France: Hérmes.

Guyet, T. (2007). Knowledge Construction FromTime Series Data Using a Collaborative Exploration Approach. *Journal of Biomedical Informatics, V. 40, No.8*, 672–687.

Hernandez, I. (2004). *Modélisation, Spécification Formelle et Vérification de Protocoles d'Interaction: une Approche Basée sur les Rôles.* France: Institut national polytechnique de Grenoble.

Hilaire, V. (2008). *Du Semi-formel au Formel : une Approche Organisationnelle pour l'Ingénierie de Systèmes Multi Agents.* France: Université de Franche-Comté.

Huget, M. (2001). *Une Ingénierie des Protocoles d'Interaction pour les Systèmes Multi Agents.* France: Université Paris IX-Dauphine.

Idani, A. (2006). *B/UML : Mise en Relation de Spécifications B et de Descriptions UML pour l'aide à la Validation Externe de Validation de Développement Formels en B.* France: Université de Grenoble 1.

Jaillet, C., & Krajecki , M. (2007). Parallel Programming With OpenMP: a New Memory Allocation Model Avoiding Cache Faults. *International Workshop on OpenMP, IWOMP 2007* (pp. 148-152). Tsinghua University, Beijing, China: LNCS, Springer.

Jean, M. (2002). *Des Systèmes Autonomes aux Systèmes Multi Agents: Interaction, émergence et systèmes complexes. .* France: Université de Montpellier II.

Jennings, N.-R. (2000). On Agent-Based Software Engineering. *Artificial intelligence, V. 117, N. 2*, 277-296.

Jiannong, C., Feng, X., Jian, L., & Sajal, K. (2002). Design of Adaptive and Reliable Mobile Agent Communication Protocols. *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)* (pp. 471-472). IEEE.

Jones, C. (1978). The Vienna Development Method: The Meta-Language. *Lecture Notes in Computer Science, Springer Berlin Heidelberg, V. 61, No.7*, 218-277.

Kahloul, L., Barkaoui, K., & Sahnoun , Z. (2005). Using AUML to Derive Formal Modelling Agents Interactions. *3rd ACS/IEEE Int. Conference on Computer Systems and Applications,AICCSA'05,* (pp. 109-116). Cairo, Egypt: IEEE.

Kawabe, Y., Mano, K., & Kogure, K. (2001). The Nepi2 Programming System: A π-Calculus-Based Approach to Agent-Based Programming. *In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems* (pp. 81-96). Greenbelt: Springer.

Kolski, C., & Mathieu, P. (2004). Conception des systèmes Multi Agents : Pistes de Réflexion en vue de Futures Coopérations entre Ergonomes et Informaticiens. *Actes de la conférence Ergo-IA* (pp. 147-154). France: Université René Descartes.

Koning, J., & Hernández, V. (2003). Generating Machine Processable Representations of Textual Representations of AUML. *Third International Workshop, AOSE'02, LN CS, V. 2585* (pp. 112-123). Bologne, Italy: Springer.

Labidi, S., & Lejouad, W. (1993). *De l'intelligence Artificielle Distribuée aux Systèmes Multi Agents.* France: Institut national de recherche en informatique et en automatique.

Lamy, P., & Blaise, J.-C. (2005). *Utilisation de Méthodes Formelles.* France: J'automatise N°42.

Le Bars, M. (2003). *Un Simulateur Multi Agent pour l'aide à la Décision d'un Collectif : Application à la Gestion d'une Ressource Limitée Agro-environnementale.* France: Université Paris IX-Dauphine.

Lehmann , K., & Moldt, D. (2004). Modeling and Analysis of Agent Protocols With Petri Nets. *LNCS, Multi Agent System Technologies, V. 31, Springer*, 85-98.

Lehmann , K., & Moldt, D. (2004). Modeling and Analysis of Agent Protocols With Petri Nets. *LNCS, Multi Agent System Technologies, Springer, V. 31, No.5*, 85-98.

Leinberger, W., Karypis, G., Kumar, V., & Biswas, R. (2000). Load Balancing Across Near-homogeneous Multi Resource Servers. *Washington, IEEE Computer Society*, 60–71.

Leitao, P., Colomb, W., & Restivo, F. (2004). An Approch to the Formal Specification of Folonic Control Systems. *Halonic and Multi Agents systems for Manufacturing, Lecture Notes in Computer Science Vol.2744* (pp. 59-70). Springer-Verlag.

Luck, M., & Inverno, M. (1995). Structuring a Z Specification to Provide a Formal Framework for Autonomous Agent Systems. *Lecture Notes in Computer Science Volume 967*, 46-62.

Mancini, S., & Petit, B. (2010). *Répartition de Charge Dynamique dans un Système Distribuée.* Grenoble: INP – Ensimag,.

Manson, S. (2003). Validation and Verification of Multi Agent Models for Ecosystem Management. *In Complexity and Ecosystem Management: The Theory and Practice of Multi Agent Approaches*, 63-74.

Martial, V. (1992). Coordinating Plans of Autonomous Agents. *In Lecture Notes in Artificial Intelligence, Springer-Verlag, V. 610* , 251-263.

Mathieu, P., Jean-C, R., & Secq, Y. (2001). Dynamic Skills Learning : a Support to Agent Evolution. *Symposium on Adaptive Agents and Multi-Agent Systems AISB'01*, (pp. 25-32). University of York, United Kingdom.

Mathieu, P., Routier, J.-C., & Yann , S. (2003). RIO: Rôles, Interactions et Organisations. *Actes des Secondes Journées Francophones sur les Modèles Formels de l'Interaction (MFI'03)*, 179-188.

Michel, F., Gouaich, A., & Ferber, J. (2003). Weak Interaction and Strong Interaction in Agent Based Simulations. *Proceedings of MABS 2003, Fourth International Worshop in Multi-Agent-Based Simulation III* (pp. 43-56). Lecture Note in Artificial Intelligence, Springer-Verlag.

Milner, R., Parrow, J., & Walker, D. (1992.). A Calculus of Mobile Processes. *Journal of Information and Computation*, 77-89.

Mokhati, F., Mourad, B., & Badri, L. (2006). A Formal Framework Supporting the Specification of the Interactions between Agents. *Informatica*, 97–110.

Moldt, D., & Weinberg, F. (1997). Multi Agent-Systems Based on Coloured Petri Nets. *18th ICATPN, Lecture Notes in Computer Science, V. 1248: 18th ICATPN*, 82-101.

Murata, T., Nelson, P., & Yim, J. (1991). A Predicate-Transition Net Model for Multiple Agent Planning. *Information Sciences, Elseiver Science*, 361-384.

Nicolas, J., Sycara, K., & Wool, M. (1998). A Roadmap of Agent Research and Development. *Autonomous Agents and Multi Agent Systems, V. 1, N.1*, 7-38.

Noél, B. (2009). *Ingénierie des Systèmes Multi Agents Adaptatifs: Vers un Guide pour la Conception.* Toulouse, France: Université Paul Sabatier.

Odell, J., Parunak, H., & Bauer, B. (2000). Representing Agent Interaction Protocols in UML. *the 1 st Internationnal Workshop on Agent Software Engineering (AOSE)* (pp. 121-140). Limerick, Ireland: Springer.

Petri, C. A. (1962). *Communication par les automates, Thèse de doctorat.* Université de Bonn, Allemagne,.

Pinney, J., Westhead, D., & DcConkey, G. (2003). Petri Net Representations in Systems Biology. *Biochem.Soci. Transaction, V.31*, 1513-1515.

Pouyan, A., & Reeves, S. (2004). Behavioral Modeling for Mobile Agent Systems Using Petri nets. *Systems, Man and Cybernetics* (pp. 135-147). IEEE International Conference.

Quan, B., Minjie, Z., & Haijun, Z. (2005). A Coloured Petri Net Based Strategy for Multi-agent Scheduling. *In Rational, Robust, and Secure Negotiation Mechanisms in Multi-Agent Systems, (RRS"05)* (pp. 3-10). IEEE.

Quani, B., Minjie, Z., & Khin, T. (2004). A Color Petri Net Based Approach for Multi-agent Interactions. *2nd International Conference on Autonomous Robots and Agents* (pp. 152-157). Palmerston North, New Zealand.: Massey University - Institute of Information Sciences & Technology.

Scott, R., Chen, Y., Finin, T., & Labrou, Y. (1999). Modeling Agent Conversations with Colored Petri Nets. *in Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, (pp. 59-66). Seattle, Washingto.

Sibertin-Blanc, C., Cardoso, C., & Hanachi, J. (2001). Spécification of interaction protocole with Petri Net. *Actes des 11èmes Journées Francophones sur les Systèmes Multi Agents, JFIADSMA'01* (pp. 121-147). Montréal, Canada: Hermes.

Steels, L. (1994). The Artificial Life Roots of Artificial Intelligence. *Artificial Life Journal, Vol. 1, MIT Press Editor*, 75-110.

Tari, Z., & Arora, P. (2007). A Communication Protocol Validation Approach based on Partial Exploration of Complex State Machines. *Distributed Computing and Internet Technology, 4th International Conference, ICDCIT* (pp. 121-135). Bangalore, India: Lecture Notes in Computer Science, Springer.

Thomas, V. (2005). *Proposition d'un Formalisme pour la Construction Automatique d'Interactions dans les Systèmes Multi Agents Réactifs.* France: Université Henri Poincaré, Nancy 1.

Tiberiu S, F. J. (2009). MASQ: Towards an Integral Approach to Interaction. *in the Eighth International Conference on Autonomous Agents and Multiagent Systems: AAMAS*, (pp. 813-820). Budapest, Hungary.

Treuil, J.-P., Drogoul, A., & Zu, J.-D. (2008). *Modélisation et Simulation à Base d'Agents.* Collection: Sciences Sup, Dunod/IRD.

Vally, J.-D., & Courdier, R. (2002). Hybrid Model to Design Proactivity and Multi Agent Systems. *International Conference on Evolutionary Computations.* 2002 World Scientific and Engineering Society (WSES).

Weyns, D., & Holvoet , T. (2002). A Colored Petri Net for a Multi Agent Application. *Components, Objects and Agents, MOCA'02*, (pp. 121-140). Aarhus, Denmark.

Wooldridge, M., Jennings, N., & Kinny , D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design. *In Journal of Autonomous Agents and Multi Agent Systems, V. 3, No. 3*, 285-312.

Xu, D., Volz, R., Loeger, T., & Yen, J. (2002). Modeling and Verifyning Multi Agent Behaviors UnsingPredicat/Transition Nets. *14th international conference on Software engineering and knowledge engineering, SEKE* (pp. 193-200). ACM.

Yeddes, M., Feng, L., & Ben Hadj-Alouane, N. (2009). Modifying Security Policies for the Satisfaction of Intransitive Non-Interference, (pp. 161-169), *IEEE Transactions on Automatic Control*.

Borhen MARZOUGUI

Contribution à la Modélisation et à la Vérification des Systèmes Multi Agents

le cnam

Résumé :

Les Réseaux de Petri (RdP) sont actuellement les approches les plus prometteuses pour modéliser et vérifier les systèmes complexes tels que les Systèmes Multi Agents (SMA). De nombreuses solutions ont été proposées pour remédier aux problèmes de communication, de coordination et d'interaction entre les Agents. Cependant, il n'existe aucune en mesure de traiter, à la fois les aspects structurels et comportementaux, du moins à notre connaissance. La thèse s'intéresse à la problématique de modélisation formelle et de vérification automatique et semi-automatique de propriétés pour les Systèmes Multi Agents. Plus précisément, l'objectif consiste à proposer un nouveau modèle formel original basé sur les réseaux de Petri, les Réseaux de Petri à Agents (RdPA), qui permettent d'exprimer de manière consistante et plus précise les systèmes Multi Agents. Il s'intéresse de plus à l'extension de ce modèle aux fins de modéliser la migration des Agents dans le cadre des systèmes à Agents mobiles. Cette classe de modèle permet de s'intéresser à la vérification formelle de propriétés classiques comme notamment la vivacité ou l'absence d'interblocage dans le cadre des Systèmes Multi-Agent.

Mots clés :

Réseaux de Petri (RdP) ; Systèmes Multi Agents (SMA) ; Interaction ; Approche formelle ; Réseau de Petri à Agent (*RdPA*).

Abstract:

Petri nets (PN) are currently the most promising approaches to model and to verify complex systems such as Multi Agent Systems (MAS). Several solutions have been proposed to solve the problems of communication, coordination and interaction among Agents. However, to best of our knowledge, none of this solution has able to handle both aspects: structural and behavioral. The thesis focuses on the problem of formal modeling and automatic and semi-automatic verification of properties in Multi Agent Systems. More specifically, the objective is to propose a new original formal model based on Petri nets, Agents Petri nets (APN), which express consistently more accurate a Multi Agent Systems. There is growing interest in the extension of this model for modeling the migration of Agents within the mobile Agent systems. This class of model allows focusing on the formal verification of classical properties such as alertness or absence of deadlock in the context of Multi Agent Systems.

Keywords:

Petri Nets (PN); Multi Agents System (MAS); Interaction; Formal approach; Agent Petri Nets (APN).