République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE HADJ LAKHDAR BATNA

Faculté des Sciences

Département d'Informatique

MEMOIRE

Présenté en vue de l'obtention du diplôme de

MAGISTER EN INFORMATIQUE

Spécialité : Ingénierie des Systèmes Informatiques

Thème

Une approche à base d'agents adaptatifs pour la résolution des systèmes complexes

Par

Hichem RAHAB

Directeur de mémoire : Dr. Ramdane MAAMRI

Composition du jury :

Pr. Azzedine BILAMI	Président	(Professeur université de Batna)
Dr. Ramdane MAAMRI	Rapporteur	(Maitre de conférences université de Constantine)
Dr. Brahim BELATTAR	Examinateur	(Maitre de conférences université de Batna)
Dr. Abdelmadjid ZIDANI	Examinateur	(Maitre de conférences université de Batna)
Dr. Farid MOUKHATI	Examinateur	(Maitre de conférences université d'Oum el Bouaghi)

Année universitaire : 2010 - 2011

A mes chers parents : Khadidja et Tayeb,

Pour leur patience, orientation, amour, conseils, éducation et encouragement

durant toutes mes années d'études, et avant et après.

A mes chers frères : Hamza, Ghazali, Abdou, Chemsseddine et ma sœur Ghada

A toute ma famille.

Remerciement

Je tien à remercier au premier le Dieu tout puissant pour la puissance, la volonté, le courage et la patience qu'il nous a donné durant toute les années d'étude.

Je tien aussi à remercier vivement mon encadreur Dr. Ramdane MAAMRI, pour son aide et ces conseils.

Merci …

Je remercie les membres du jury d'avoir accepter de juger mon travail.

A tous qui m'aide de prés ou de loin dans ce travail

Résumé

Les systèmes logiciels actuels sont de plus en plus exposés à de nouvelles situations, qui est difficile de tous prévoir à l'avance et de préparer un traitement convenable, notamment les systèmes multi agents qui sont caractérisés par sont autonomie, puisque ses agents vont déplacés d'un site à un autre où ils trouvent de nouvelles conditions avec lesquelles ils doivent s'adapter.

L'adaptation de comportement des agents avec sont environnement doit être une caractéristique interne, et cela pour minimiser l'intervention du concepteur et préserver l'autonomie de l'agent, mais d'un autre coté elle doit avoir une façon de contrôle pour ne pas arriver à des adaptations aléatoires et des comportements qui peuvent être inadéquats.

Dans ce travail nous avons proposé un modèle d'un agent auto-adaptable à base de composants, et conçu en suivant une approche à deux niveaux. Dans notre modèle on a distingué le comportement normal de l'agent matérialisé par un bloc de fonctionnement qui est une sorte d'agent classique, et le fonctionnement d'adaptation réalisé par un bloc supérieur d'adaptation et de supervision, ce dernier est chargé d'assurer le bon fonctionnement de l'agent en le dotant des bons comportements qui sont destinés aux différentes situations qu'il peut rencontrer.

L'opération d'adaptation a été modélisée par un réseau de Petri permettant d'automatiser le processus de génération de nouvelles configurations à partir des composants disponibles dans la base, et en suivant la description du changement de l'environnement.

Sommaire

Chapitre 4 : Adaptabilité dans les systèmes multi agents 52

Liste des figures

Liste des tableaux

Chapitre 1 : Introduction générale

Les premiers systèmes informatiques sont des entités isolés, dont la communication est limitée à leurs opérateurs humais, bien que les systèmes d'aujourd'hui sont souvent interconnectés et intégrés dans des grands systèmes, qu'on appel systèmes distribués. L'internet est un exemple très dominant de la distribution des systèmes actuels, de telle sorte qu'on a arrivé à un temps où la quasi-totalité des ordinateurs dans les différentes institutions académiques et industrielles font partie de ce gigantesque système distribué de connexion et de partage. Bien que Les systèmes distribués et concurrents sont vues dans ses premiers temps, comme des applications complexes, ambigus, et source de problèmes, ces idées sont changées radicalement avec le développement très reconnu et très rapide de l'Internet. Aujourd'hui, et dans le future, les systèmes distribués et concurrents deviennent la norme dans les systèmes informatiques commercial et industriel, menant chercheurs et praticiens de mettre en cause leurs fondements de raisonnement en informatique, en cherchant des modèles théoriques qui reflètent le mieux la réalité de calcule principalement comme un processus d'interaction.

La tierce tendance est alors de plus vers des systèmes intelligents. C'est que, la complexité des tâches à automatiser et résoudre avec des solutions informatique augmente d'une manière régulière. Et on va progressivement tendre vers une conception des systèmes informatiques résolvant des problèmes que avant, une solution par ordinateur n'est pas du tout envisageable.

La prochaine tendance est vers une augmentation de confiance. Par exemple, on délègue systématiquement aux systèmes informatiques des tâches de plus en plus critiques allant jusqu'au pilotage d'avions. Dans la réalité, on arrive à un point où on peut adopter le jugement d'un ordinateur dans un système avionique, pour un problème donné, sur lequel on fait confiance et le préféré du jugement des pilotes très expérimentés dans le domaine.

La communication homme machine a aussi connu de très importants avancements. Dans les premiers jours de l'informatique, l'utilisateur communique avec l'ordinateur par une manipulation directe des circuits dans le bord de la machine, l'exécution interne de la machine est visible pour l'utilisateur, et pour pouvoir l'utiliser d'une manière efficace, il doit parfaitement comprendre sa structure interne et son fonctionnement.

Cette interface primitive et non productive laisse peu à peu la place à une interface de ligne de commande, où l'interaction avec la machine est significativement avancée, et l'utilisateur sera capable de faire entrer des instructions qui seront ensuite exécutés. A sont tours ces interfaces ont données leurs places aux interfaces graphiques, et à un paradigme de manipulation direct dans lequel l'utilisateur contrôle la machine par la manipulation des icones correspondants aux objets tel que des programmes et des fichiers. Tous ces

développements ont permet au programmeur de concevoir et d'implémenter des logiciels de haut niveau et avec un haut degré d'abstraction et d'interopérabilité.

La tendance d'augmenter le domaine d'application de l'informatique et l'automatisation des tâches implique le besoin de développer des systèmes efficaces qui peuvent agir convenablement pour notre bénéfice. Cela implique deux aptitudes, la première est l'aptitude des systèmes à agir indépendamment de notre intervention directe. La seconde est le besoin des systèmes informatiques d'être capable de représenter parfaitement nos intérêts lors de l'interaction avec nous et/ou avec d'autres membres gens de systèmes.

Ces tendances ont permettaient l'émergence d'une nouvelle filière en informatique : les systèmes multi agents. L'idée de systèmes multi agents est très simple. Un agent est un système informatique qui est capable d'actions autonomes au profit de son propriétaire. En d'autres termes, un agent peut décider de ce qu'il doit faire pour satisfaire ses objectifs de conception, et non seulement de faire ce qui est lui demander à tout moment. Un système multi agents est basé sur un nombre d'agents, qui interagissent les uns avec les autres, typiquement par échange de messages à travers une plate forme sur laquelle les agents en quelque sorte vivent. Dans un système multi agents, et pour la plupart des cas, les agents vont déplacer, communiquer et agir pour le compte d'utilisateurs avec différents buts et différentes motivations. Pour réussir l'interaction, ces agents ont besoin d'une aptitude de coopération, de coordination, et de négociation entre eux, dans le même sens que nous les hommes on coopère, on coordonne et on négocie avec des gens de notre société.

La façon de concevoir les programmes informatiques comme des sortes de "penseurs" repliés sur eux-mêmes se retrouve directement dans la notion de système expert, c'est-à-dire des programmes informatiques capables de remplacer l'être humain dans ses tâches réputées les plus complexes et qui réclament de l'expérience, du savoir-faire et d'une certaine forme de raisonnement.

Dans cet univers, l'environnement des machines serait sans cesse changeant. En effet, comme c'est le cas actuellement avec les terminaux mobiles, une partie des équipements suivrait leurs propriétaires dans leurs déplacements. Les logiciels dans plusieurs domaines actuels doivent ainsi s'adapter automatiquement à ces changements de contexte. La notion d'adaptabilité précise qu'un système qui a cette propriété se comporte de façon adéquate face aux sollicitations de son environnement. Et on entend par système un objet construit rationnellement, dans le but de servir les objectifs de son constructeur en utilisant des méthodes classiques et éprouvées. Il est en effet très simple de constater qu'un système est adaptatif en le regardant se comporter de manières influencées par l'état momentané de son environnement. Et cela conduit à considérer que l'adaptabilité est une propriété locale, parmi d'autres, comme par exemple la capacité à se diriger en optimisant une trajectoire ou bien à émettre des sons harmonieux. Il s'agit donc d'une nouvelle classe de systèmes en

Informatique, qui impose de nouvelles architectures. Il n'y aura pas de notion d'état final atteint, pas de progression vers la solution d'un problème bien posé, mais conformation coordonnée de deux structures en fort couplage, et conduisant à un état stable, l'état adaptatif.

Le développement à base de composants est une notion très populaire de nos jours dans le domaine du génie logiciel. Les technologies supportant les composants telles que EJB, CORBA, COM/DCOM/.NET sont bien établies. Cependant, les méthodes de développement des systèmes à base de composants n'ont pas encore atteint le stade de maturité des technologies actuelles. Réutiliser a de tout temps constitué une action clé de l'ingénierie des systèmes logiciels et des systèmes d'information. Le développement des logiciels à base de composants (Component-Based Software Engineering) est devenu une réalité avec l'évolution des technologies web et java qui ont facilité la distribution, la recherche, et l'interopérabilité des composants sur internet. Le nombre et la diversité de ces propositions constituent un signe d'effervescence des travaux dans le domaine mais aussi de leur disparité.

Malgré un réel avancé, force est de reconnaître que les problèmes liés aux développements et à l'usage des composants ne sont pas encore complètement résolus. La principale raison est que tout produit qui émane d'un processus de développement, qu'il soit conceptuel (spécification) ou plus opérationnel (langage abstrait, code), est naturellement faiblement réutilisable. Rendre réutilisable un produit de développement nécessite de disposer de langages conceptuels et opérationnels permettant l'expression de solutions génériques, adaptables, intégrables, composables, etc. Des formes d'expression des systèmes d'information favorisant la production d'éléments plus modulaires, compréhensibles hors du contexte où ils sont amenés à s'associer à d'autres éléments, dotés de facultés d'intégration/connexion plus explicites ont émergés. La recherche dans le domaine de l'ingénierie des systèmes d'information s'est appliquée à mieux caractériser ces formes, et à en définir leur usage dans des démarches et processus idoines. L'approche à base de composants est aujourd'hui considérée comme un nouveau paradigme de développement des systèmes d'information. Comme avant avec en ingénierie des systèmes d'information, les activités de programmation ont été les premières concernées par ce nouveau paradigme. Des propositions d'approches à base de composants commencent à apparaître pour répondre à des problèmes d'ingénierie de besoins, de spécification et de modélisation de systèmes.

Composants logiciels et systèmes multi-agents ont en commun l'intérêt de permettre la structuration d'un système logiciel dans une organisation d'unités relativement indépendantes et qui interagissent entre eux d'une manière cohérente. L'utilité de ces deux technologies est particulièrement notable à cause des besoins en autonomie et en adaptation.

Le besoin en autonomie se trouve notamment dans le fait que les différents dispositifs matériels impliqués dans une application doivent interagir en minimisant à la fois les interventions humaines (par exemple, répondre automatiquement aux requêtes dans la

fourniture de services web en fonction du profile d'un utilisateur) et la liaison entre ces dispositifs (Par exemple la possibilité de retrait imprévu d'un équipement dans un système distribué). Le besoin en adaptation résulte du fait que les machines impliquées ont des ressources matérielles hétérogènes et que leurs contextes d'exécution évoluent sans cesse. Dans ce contexte, nous sommes confrontés à des problèmes qui nous poussent à utiliser conjointement la programmation par composants et les systèmes multi agents. Nous souhaitons pouvoir adapter dynamiquement et automatiquement les agents à base de composants, en particulier, grâce à des opérations de réassemblage (ou reconfiguration) : l'ajout, la suppression et le remplacement de composants en cours d'exécution.

Les méthodes de modélisation formelles sont de bons candidats pour la spécification des composants logiciels. On entend par méthodes formelles, les langages, techniques, et les outils basés sur la logique mathématique. En effet la base des méthodes formelles est de nous permettre de construire des modèles mathématiques avec une sémantique bien définie du système à analyser. Les réseaux de Petri sont considérés comme des formalismes de spécification dotés d'une définition formelle permettant de construire des modèles exempts d'ambiguïté. Ils possèdent un grand pouvoir d'expression facilitant la description des systèmes complexes, concurrent et distribués.

Dans ce mémoire nous avons abordé le sujet de l'adaptation dans les systèmes multi agents, et on a traité le problème d'un agent auto-adaptable à base de composants, l'approche par composants nous a permettait la réutilisation des composants déjà disponibles, donc un gain très important en termes de temps et d'effort. D'autre part, le modèle par composants facilite l'adaptation par le fait que les relations entre les composants sont des interfaces bien définies, qui facilite l'ajout de nouveaux composants, et/ou le retrait de ceux inutiles. La conception de l'agent se fait en deux niveaux, le niveau de supervision et le niveau de fonctionnement, qui permet de mieux éclairer la structure de l'agent et facilite son adaptation du fait que seul le niveau de fonctionnement va être abordé par le sujet d'adaptation. L'opération d'adaptation dans notre proposition est basée sur la génération de configurations à travers une modélisation par réseau de Petri de la base de composants, et le parcours de ce réseau par un algorithme, en suivant une description de l'état actuel de l'environnement vers une configuration de composants à mettre en marche pour le comportement prochain de l'agent.

Le premier chapitre est une introduction générale à travers laquelle on a expliqué les différentes étapes suivis et les domaines inclus dans notre travail, et cela afin de mettre la liaison entre les approches suivis, et la motivation de notre travail achevé.

Dans le deuxième chapitre, on va faire une vue générale sur les systèmes multi agents, les architectures existantes, les façons de communication et d'interaction entre les agents ainsi que les différentes plateformes et outils utilisées pour les développer.

Le troisième chapitre est l'état de l'art de notre travail sur lequel on a présenté les travaux faits sur l'adaptabilité des systèmes multi agents, et les différentes approches utilisées pour cela.

Dans le quatrième chapitre, on a présenté les réseaux de Petri comme méthode de modélisation, nous avons commencé par un ensemble de définitions de différentes notions liés aux réseaux de Petri, ensuite on a donné les types de réseaux de Petri, et quelques exemples de problèmes à modéliser avec.

Dans le cinquième chapitre on a proposé notre modèle d'un agent auto adaptable à base de composants.

Le sixième chapitre est une étude de cas à travers laquelle on a validé notre approche proposée par exemple d'un agent de fourniture de services web.

On termine avec une conclusion de ce qui est fait, et des perspectives pour de nouveaux travaux dans le même axe.

Chapitre 2 : Agent et Systèmes multi agents

2.1 Introduction

Les concepteurs de systèmes experts industriels sont arrivés à des conclusions semblables à partir des difficultés soulevées par le développement de bases de connaissance. La plupart d'entre elles proviennent de ce que, dans une application complexe, l'expertise que l'on cherche à intégrer dans la base de connaissance, c'est-à-dire le savoir-faire, les compétences et les connaissances diverses, est détenue par des individus différents qui, au sein d'un groupe, communiquent, échangent leurs connaissances et collaborent à la réalisation d'une tâche commune. Dans ce cas, la connaissance du groupe n'est pas égale à la somme des connaissances des individus: chaque savoir-faire est lié à un point de vue particulier et ces différentes visions, parfois contradictoires, ne sont pas nécessairement cohérentes entre elles. La réalisation d'une tâche commune nécessitera alors des discussions, des mises au point, voire même des négociations pour résoudre les conflits éventuels. Une résolution efficace de ces problèmes doit s'intéresser à mettre la solution dans des entités autonomes reflétant la réalité distribué des systèmes naturelles.

Les systèmes multi agents constituent une extension de l'intelligence artificielle distribuée (IAD), qui consiste à mettre l'intelligence, précédemment fixé sur une seule entité, sur un ensemble de sous systèmes et d'avoir un comportement globale intelligent qui émerge de l'interaction de ces parties même si aucun d'entre eux ne possède un comportement intelligent. Ainsi, au lieu de se trouver en présence d'une "machine" - une entité bien localisée par sa structure et son architecture - on se trouve, comme le développement des langages objets l'a montré, devant un ensemble d'entités en interactions, chaque entité étant définie de manière locale sans vision globale détaillée de toutes les actions du système. Cette façon d'envisager des programmes introduisit de nouvelles méthodes de conception en génie logiciel et un changement de perspective, on passa de la notion de programme à celle d'organisation.

La résolution de problèmes par systèmes multi agents consiste à pouvoir étudier, concevoir et réaliser des univers ou des organisations d'agents artificiels capables d'agir, de collaborer à des tâches communes, de communiquer, de s'adapter, de se reproduire, de se représenter l'environnement dans lequel ils évoluent et de planifier leurs actions, pour répondre soit à des objectifs définis extrinsèquement (par un programmeur humain par exemple), soit intrinsèquement à partir d'un objectif général de survie. On dira aussi que la solution procède par la construction de systèmes multi-agents, c'est-à-dire par la réalisation de modèles composés d'entités artificielles qui communiquent entre elles et agissent dans un environnement.[1]

[1] Voir J. Ferber : Les systèmes multi-agents - Vers une intelligence collective. InterEditions, 1995.

2.2 Agents et système multi agents

2.2.1 Que ce qu'un agent ?

Une définition d'un agent donné par J Ferber est :

"On appelle agent une entité physique ou virtuelle

a. *qui est capable d'agir dans un environnement,*
b. *qui peut communiquer directement avec d'autres agents,*
c. *qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),*
d. *qui possède des ressources propres,*
e. *qui est capable de percevoir (mais de manière limitée) son environnement,*
f. *qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),*
g. *qui possède des compétences et offre des services,*
h. *qui peut éventuellement se reproduire,*
i. *dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit*"[2].

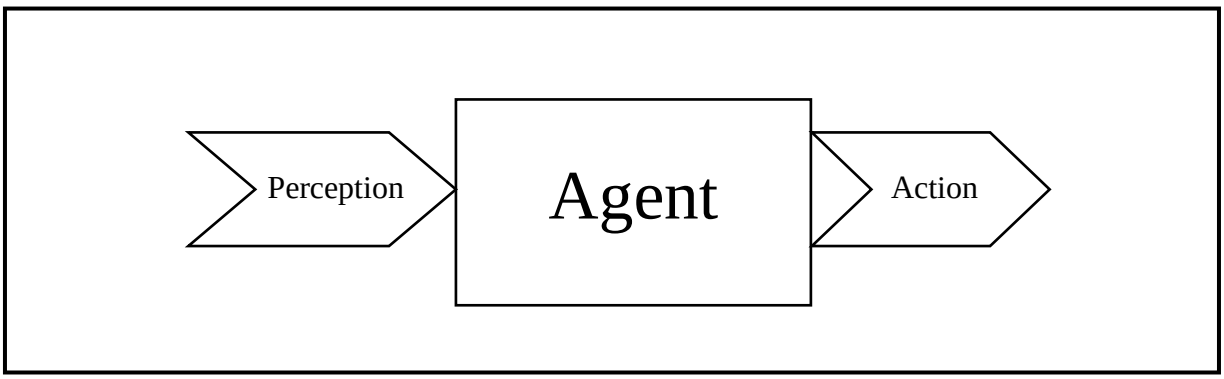

Figure 1: Représentation d'un agent

Donc un agent est une entité **physique** agit dans le monde réel. Un robot, un avion une voiture sont des exemples d'entités physiques. En revanche, un composant logiciel, un module informatique sont des entités **virtuelles**, car elles n'existent pas physiquement.

[2] J. Ferber : Les systèmes multi-agents - Vers une intelligence collective. InterEditions, 1995. P27

A la différence des systèmes experts classiques qui donnent seulement des conseils (feedback) les agents sont capables d'agir. L'action, qui est un concept fondamental pour les systèmes multi-agents, repose sur le fait que les agents accomplissent des actions qui vont modifier leur environnement et donc leurs futures prises de décision. Ils peuvent aussi communiquer entre eux.

Les agents ont une autonomie sur les actions qu'ils sont chargés d'accomplir. Cela signifie qu'ils ne sont pas dirigés par des commandes venant de l'utilisateur ou d'un autre agent (comme c'est le cas des objets par exemple dans la programmation orienté objets), mais par un ensemble de tendances qui peuvent prendre la forme de buts individuels à satisfaire ou de fonctions de satisfaction ou de survie que l'agent cherche à optimiser. On pourrait dire ainsi que le moteur d'un agent, c'est lui-même, C'est lui qui est actif. Il a la possibilité de répondre par l'accord ou par le refus à des requêtes provenant des autres agents. Il dispose donc d'une certaine liberté de manœuvre, ce qui le différencie de tous les concepts semblables, qu'ils s'appellent "objets", "modules logiciels" ou "processus". Mais l'autonomie n'est pas liée seulement au comportement, elle porte aussi sur les ressources.

Les agents ont une représentation de leur environnement, mais cette représentation ne peut en aucun cas être complète, elle est plus au mois idéale selon la nature de l'agent et son rôle dans l'environnement. C'est-à-dire qu'ils n'ont pas de vision globale de tout ce qui se passe. On peut comparer les systèmes multi agents à un grande usine où personne n'a une connaissance détaillée de ce que faire les autres mais ils peuvent tous travailler en parfaite collaboration.

On peut conclure qu'un agent est une entité logiciel, physique ou même un être humain, qui est chargée d'accomplir seule une certaine tâche ou objectif. L'autonomie est une caractéristique essentielle d'un agent, elle signifier qu'il est laissé seul dans son environnement et il doit décider – en fonction de ce qui il a comme compétences- comment doit-il se comporter.

2.2.2 Un système multi agents

"On appelle système multi-agent (ou SMA), un système composé des éléments suivants:

1. *Un environnement E, c'est-à-dire un espace disposant généralement d'une métrique.*
2. *Un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.*
3. *Un ensemble A d'agents, qui sont des objets particuliers (A inclut O), lesquels représentent les entités actives du système.*
4. *Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.*

5. *Un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O.*
6. *Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers. "* [3]

Donc un système multi agents (SMA) est une sorte de société d'un ensemble d'agents qui coopèrent pour un but commun, qu'un agent ne peut pas le faire seul.

Il y a plusieurs tentatives pour la définition des agents et des systèmes multi-agents. La situation est comparable en quelque sort avec celle rencontrée quand les scientifiques ont essayé de définir la notion d'intelligence artificielle. Pourquoi a-t-il été si difficile de définir l'intelligence artificielle (et nous nous doutons même maintenant d'avoir réussi à donner une définition exacte) et pourquoi est-il si difficile de définir les systèmes d'agents, quand d'autres concepts de l'informatique, comme ceux d'objet et orienté-objet, calcul distribué, etc., n'ont pas rencontré une si grande résistance à être définis.

Une réponse possible est que la notion d'agent, ainsi que celle d'intelligence artificielle, sont émergées des humains et de la société humaine. Il est évidement difficile de modéliser ou de simuler le comportement spécifique inspiré de la société humaine dans des programmes informatiques. Des recherches anciennes tentent de développer des programmes informatique (ou plus précisément d'intelligence artificielle) pour émuler le comportement d'un être humain intelligent, dont le but était de créer un système artificiel ayant les mêmes capacités qu'une personne intelligente. Bien que les recherches actuelles aient été orientées vers la simulation du comportement collectif des sociétés humaines dans leur environnement, comment ils réagissent, comment ils résolvent des problèmes de plus en plus complexes par la distribution des tâches ou comment ils élargissent leurs performances par la coopération ou la compétition.

La résolution coopérative de problèmes prend une place prépondérante dans les recherches en intelligence artificielle distribuée (IAD). Le domaine des systèmes multi-agents (SMA) est un domaine de recherche relativement complexe, dérivé de l'IAD. La thématique SMA se focalise sur l'étude des comportements collectifs et sur la répartition de l'intelligence sur des agents plus ou moins autonomes, capables de s'organiser et d'interagir pour résoudre des problèmes complexes.

A la différence de l'Intelligence Artificielle (IA) qui modélise le comportement intelligent d'un seul agent, l'intelligence artificielle distribuée (IAD) s'intéresse à des comportements intelligents qui résultent de l'activité coopérative de plusieurs agents. Suite à la distribution de l'expertise sur un ensemble de composants qui communiquent pour atteindre

[3] J. Ferber : Les systèmes multi-agents - Vers une intelligence collective. InterEditions, 1995. P28

un objectif global ou résoudre un problème, il est nécessaire de diviser le problème en sous-problèmes. Cette division n'est pas toujours aisée car beaucoup de problèmes ne peuvent pas être divisés. Ainsi une extension des systèmes d'IAD est proposée : les composants doivent être capables de raisonner sur les connaissances et les capacités des autres dans le but d'une coopération effective. Pour ce faire, ils doivent être dotés de capacités de perception et d'action sur l'environnement et doivent posséder une certaine autonomie de comportement, on parle alors d'agents et par conséquent de système multi-agents

2.3 Les SMAs et les autres paradigmes de la programmation

2.3.1 Les SMAs et l'intelligence artificielle

Les SMA sont nés dans le milieu de l'intelligence artificiel distribué (IAD), te au besoin des entités intelligentes qui font des tâches autonomes et peuvent être éparpillés de partout où on a besoin.

Malgré les points communs qui existent entre les systèmes multi agents et les systèmes experts, surtout ce qui concerne la tentative de résoudre des problèmes qui n'ont pas de solution avec d'autres méthodes traditionnelles. Il y a certaines différences entre les deux notions :

- un système expert ne perçoit pas son environnement mais plutôt acquiert les informations de l'environnement à travers une tiers personne, bien qu'un agent a des capteurs à travers lesquelles il perçoit son environnement ;

- Aussi un système expert ne réagit pas à l'environnement mais donne seulement des conseils (ou des feeds back) qui aident un expert, dans le temps où un agent à des actionneurs à travers lesquelles il modifier lui-même son environnement.

2.3.2 Les SMAs et l'approche Orienté Objet

Un paradigme très proche des systèmes multi agents est l'approche Orienté Objet, et il n'y a pas des frontières claires entre un agent et un objet, et la preuve est que certains agents sont programmés en utilisant des langages orientés objet. Cependant on peut trouver des différences entre les deux entités objet et agent tel que :

- L'autonomie, un agent peut décider seul dans des situations de prendre l'un ou l'autre des chemins possibles, bien que l'objet fasse exactement ce qui est prévu précédemment,

- Un objet ne peut pas refuser d'exécuter une méthode une fois invoquer par d'autres objets donc il n'a pas de contrôle sur son comportement à l'inverse un agent négocie les actions qui lui sont demandés et exécute seulement ceux que lui conviennent.

2.4 L'agent et son environnement

L'environnement est un aspect très important dans l'étude des systèmes multi agents, puisque dans cet environnement ils vivent, ils le perçoivent à travers ses capteurs et agit sur lui à travers ses effecteurs.

Donc un agent doit avoir une bonne modélisation de son environnement assez complète pour pouvoir agir efficacement. Mais elle ne peut aucunement être complète due à la nature des environnements et des problèmes à résoudre avec tels systèmes.

Bien que les environnements sont différents, et il y a beaucoup de critères qui entre dans leurs propriétés, il est difficile d'avoir une liste exhaustive de leurs caractéristiques. Une classification des environnements est celle de Russell et Norvig[4].

- **Accessible versus inaccessible.** Un environnement est accessible si l'agent peut avoir une vue complète, exacte et actualisée sur l'état de l'environnement. Et la plupart des environnements réels ne sont pas accessible dans ce sens.

- **Déterministe versus non déterministe.** Un environnement est déterministe si toute action sur cet environnement a un seule effet, c à d qu'il n y a pas une ambiguïté sur l'état qui résulte après l'achèvement d'une action.

- **Statique versus dynamique.** Un environnement est statique s'il est supposé inchangé sauf par les actions de l'agent. Par contre un environnement dynamique peut avoir des changements au-delà de la portée de l'agent.

- **Discret versus continu.** Un environnement discret est un environnement où les perceptions de l'agent et ses actions sont au nombre limité à l'inverse d'un environnement continu.

[4] M. Wooldridge : Introduction to MultiAgent Systems. John Wiley and Sons, 2002. p. 18

A travers cette classification on peut considérer un environnement idéale comme étant accessible, déterministe, statique et discret, mais en réalité que les environnements existant possèdent toute ces caractéristiques d'une manière plus au moins différente.

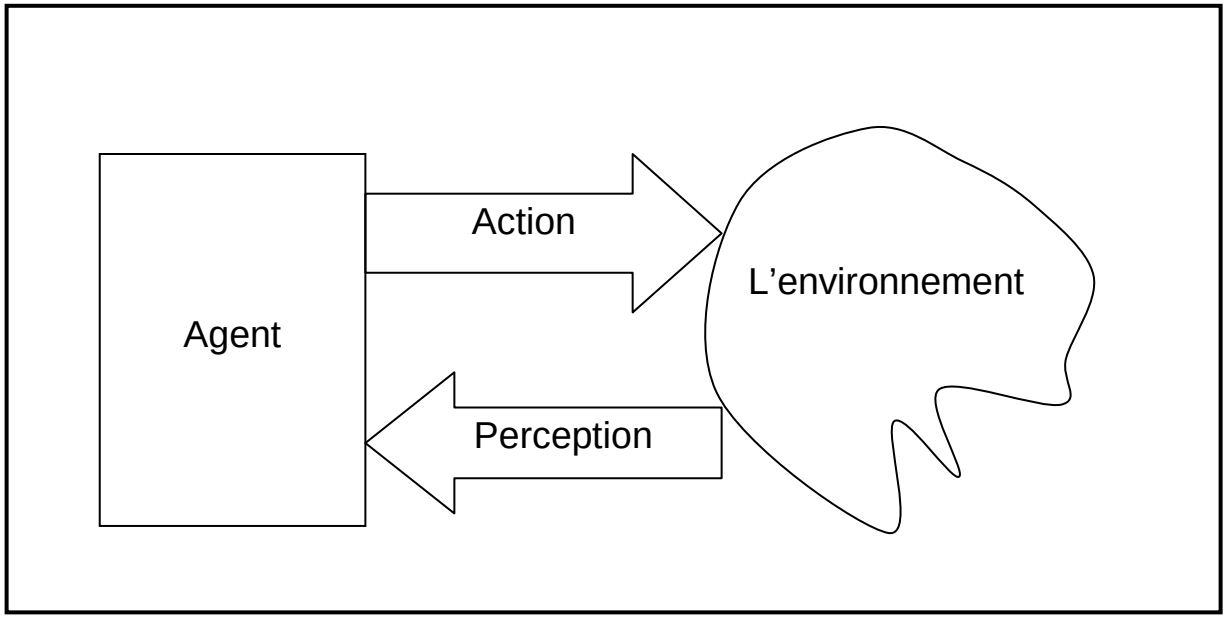

Figure 2: L'agent et son environnement

On peut classer les agents selon leurs interactions avec sont environnement entre des agents ***purement communicants*** et des agents ***purement situés***,

Un agent *purement communicant* est un agent qui n'a pas une perception de sont environnement, et il n'agit pas directement sur cet environnement, comme par exemple il s'agit des agents mobiles.

Un agent *purement situé* est quant à lui diffère des autres agents par le fait qu'il ne possède pas une perception des autres agents, ces agents ne communiquent pas directement avec les autres agents mais indirectement à travers ses perceptions et actions sur l'environnement[5].

2.5 Architectures d'agents

Un agent se caractérise essentiellement par la manière dont il est conçu et par ses actions, en d'autres termes par son architecture et par son comportement. L'architecture correspond à un point de vue de concepteur, qui peut se résumer par la façon d'assembler les différentes parties d'un agent de manière qu'il accomplisse les actions que l'on attend de lui?

[5] J. Ferber : Les systèmes multi-agents - Vers une intelligence collective. InterEditions, 1995.

Suivant la façon dont un agent prend les décisions de ses réactions on peut distinguer plusieurs types d'agents.

L'architecture d'un agent caractérise ainsi sa structure interne, c'est-`a-dire le principe d'organisation qui sous-tend l'agencement de ses différents composants.

2.5.1 Agent basé sur la logique

Cette architecture est basée sur une représentation symbolique de l'environnement, ainsi que des actions que l'agent doit entreprendre. Puis une suite de déductions logiques qui permettent de faire la liaison entre les perceptions (qui sont déjà représenter symboliquement), et les actions qui conviennent.

2.5.2 Agent réactif

Un agent réactif est un agent qui réagit aux changements de l'environnement en suivant des règles de correspondance directes entre un ensemble de perceptions et un ensemble d'actions.

On peut trouver deux modèles de conception d'agents réactifs :

2.5.2.1 Agents à reflex simple

Ce type d'agent réagit seulement en fonction de ses perceptions actuelles, il utilise des règles de la forme : **Si** condition **Alors** action.

2.5.2.2 Agents conservant une trace du monde

A la différence de type précédent ce type d'agent considère en plus de la perception de ses capteurs, des informations sur comment le monde évolue, et l'impact des actions de l'agent sur son environnement sont aussi prises.

Un tel agent doit avoir un mécanisme qui (en fonction de l'historique de son environnement) permet de distinguer deux situations ayants des perceptions identiques, mais qui sont en réalité différentes

Par exemple, si un radar détecte un missile et que l'instant d'après, il le perd de vue, dû à un obstacle, cela ne signifie nullement qu'il n'y a plus de missile. Dès lors, l'agent en charge du contrôle doit tenir compte de ce missile dans le choix de ses actions et ce, même si

le radar ne détecte plus le missile. Le problème que nous venons de mentionner survient parce que les capteurs de l'agent ne fournissent pas une vue complète du monde. Pour régler ce problème, l'agent doit maintenir des informations précédentes sur l'état du monde, et utilise ses informations lors de la prise des prochaines actions.

2.5.3 Agent délibératif

Un agent délibératif est un agent qui fait une suite de délibération pour choisir ses actions, soit en se basant sur les buts, soit en utilisant une fonction d'utilité.

2.5.4 Agent BDI (Belief-Desire-Intention)

BDI est l'acronyme de Belief-Desire-Intention qui signifier en français de croyances, désires, intention. Donc le comportement d'un agent BDI est basé sur ces trois concepts.

Un agent BDI est composé de :

1. **Un ensemble de croyances courantes :** représentent les informations que l'agent a maintenant sur son environnement.

2. **Une fonction de révision de croyances :** permet de mettre à jour les croyances de l'agent en fonction de ses croyances précédentes et de l'état actuel de l'environnement.

3. **Une fonction de génération des options :** elle détermine les options disponibles actuellement à l'agent (ses désires, ou encore les choix des actions à entreprendre).

4. **Un ensemble d'options courantes :** qui sont les désires ou les alternatives possibles pour l'agent.

5. **Une fonction de filtre :** c'est le processus de délibération de l'agent qui en fonction de croyances, désires, et intentions actuels de l'agent génère les nouveaux intentions.

6. **Un ensemble d'intentions courantes :** C'est le centre d'attention actuel de l'agent, c'est les objectifs qu'il essaye d'atteindre.

7. **Une fonction de sélection des actions :** cette fonction détermine les actions à effectuer en se basant sur les intentions actuelles de l'agent.

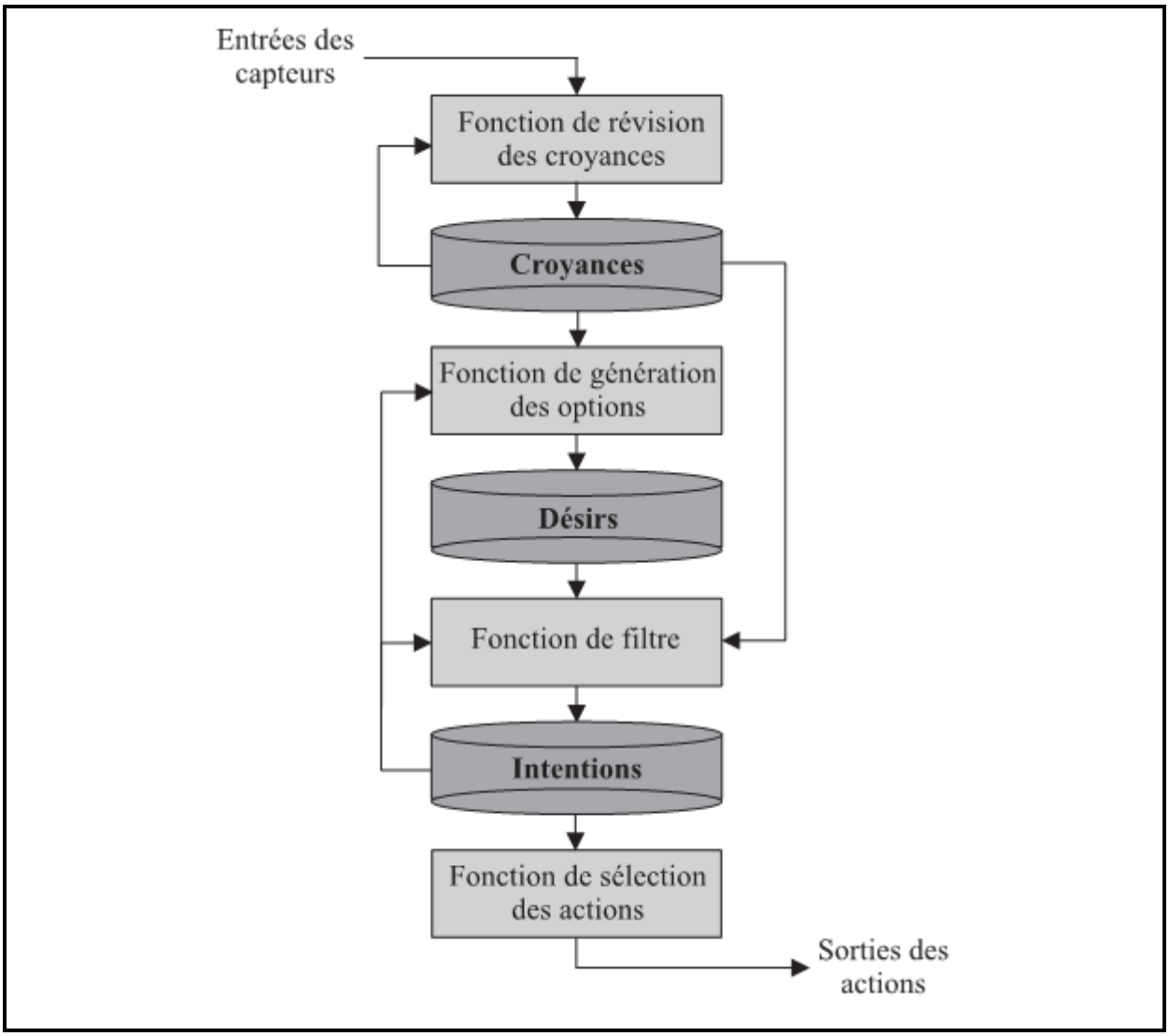

Figure 3: L'architecture d'un agent BDI

2.5.5 Architecture hybride ou architecture couches

Dans la majorité des cas un comportement purement réactif ou purement délibératif ne convient pas puisque on peut se retrouver dans des situations qui nécessitent une réponse et avec le même agent on peut se confronter à des situations que l'important n'est pas le temps mais la qualité de la réponse.

Dans cette architecture appelée architecture hybride ou en couches, le comportement de l'agent est réparti sur plusieurs couches.

2.5.5.1 Architecture en couches horizontales

Dans ce type d'architecture chaque couche se comporte comme un agent de fait quelle est liée aux entrés des capteurs et aux sorties des actionneurs et elle produit des propositions sur les actions à exécuter.

Le problème avec cette architecture réside du fait que lorsque chaque couche propose une action il faut choisir lequel doit être exécuter.

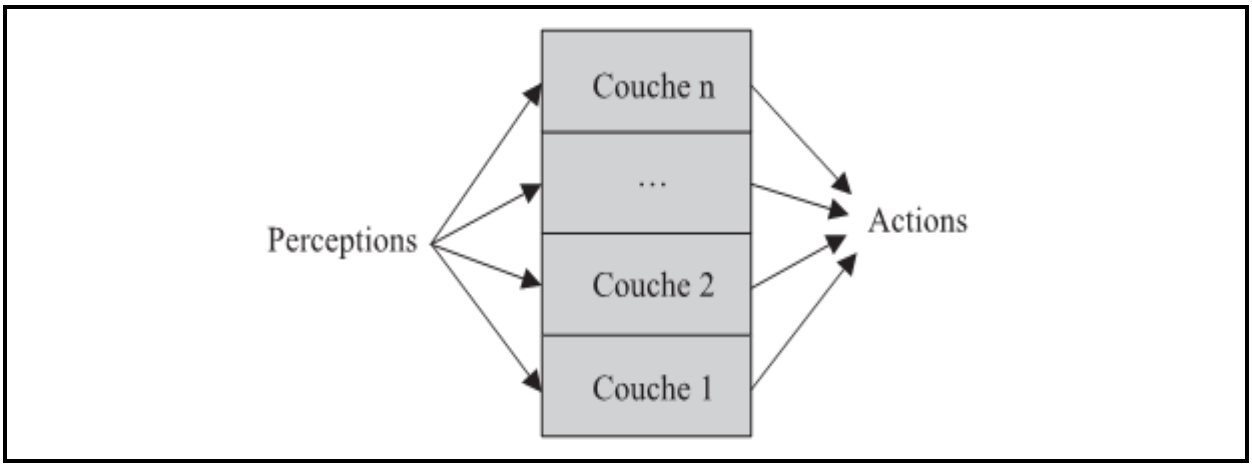

Figure 4: Architecture en couches horizontale

2.5.5.2 **Architecture** en couches verticales

Dans cette architecture seulement une couche est liée au entrées des capteurs et une autre seule couche est liées aux sortie des actionneurs, donc le problème de conflit entre les couches lors du choix des comportements ne se pose pas. On trouve deux types d'architectures en couches verticales: architecture en couches à un seul passe et architecture en couches à deux passes.

Dans une architecture en couches à **deux passes** les informations circulent dans un sens et le flux de contrôle dans le sens inverse.

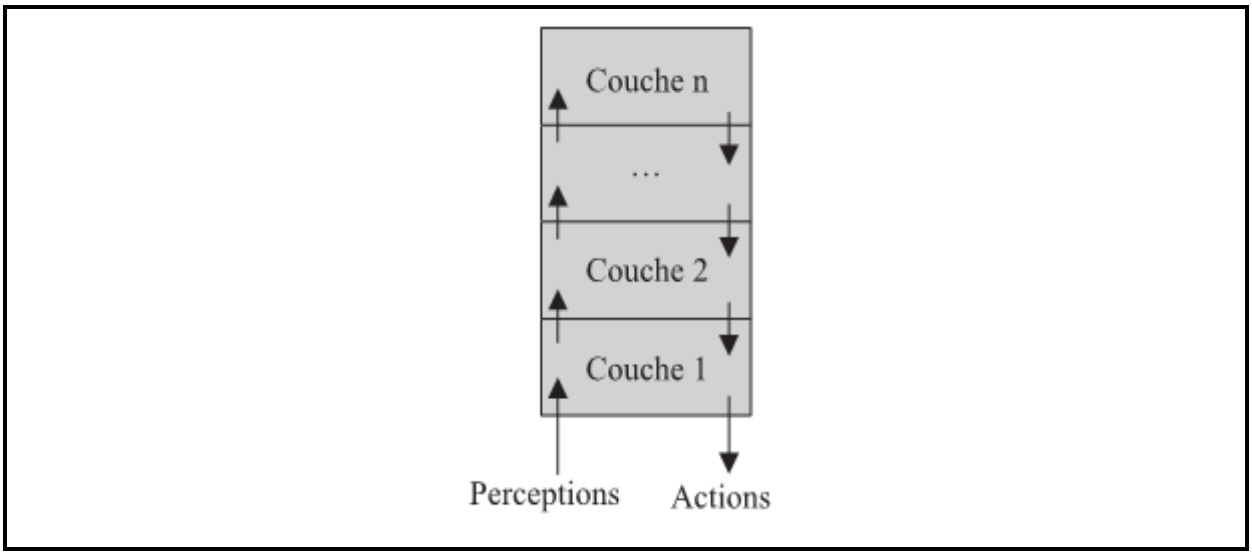

Figure 5: Architecture en couches verticale à deux passes

Alors que dans l'architecture à **une seule passe** le flux de contrôle passe de couche à l'autre jusqu'à ce que la couche finale génère l'action à exécuter.

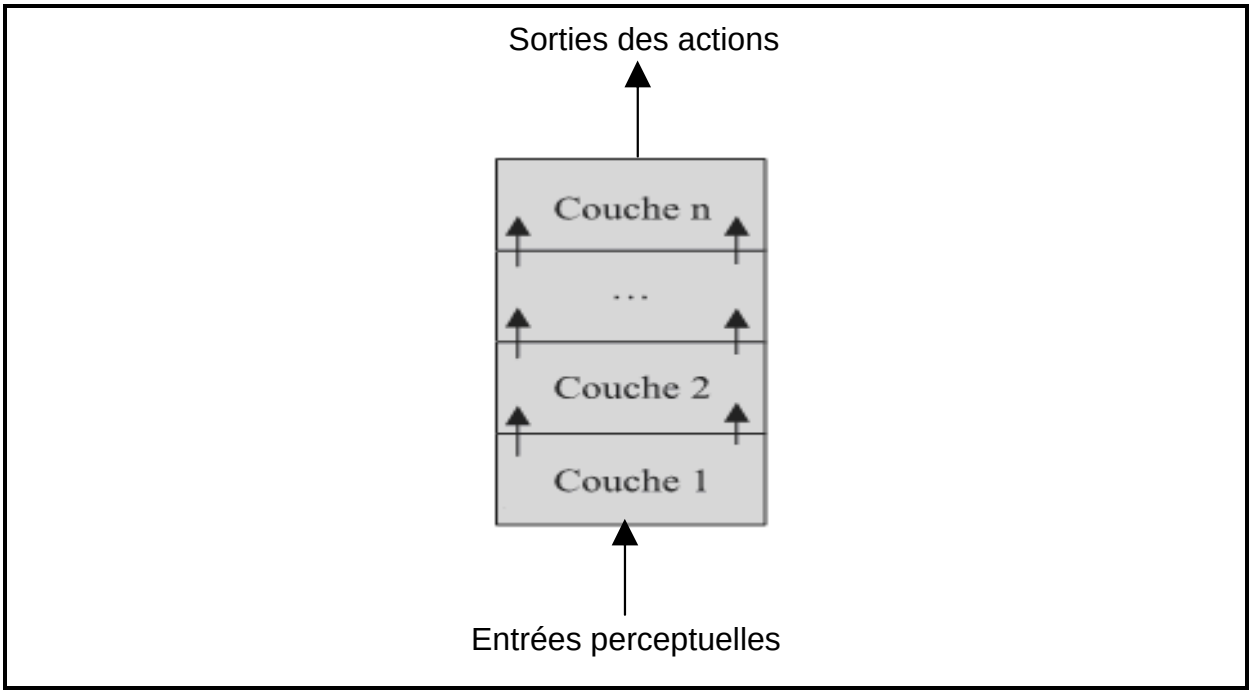

Figure 6: Architecture en couches verticale à une seule passe

2.6 Communication et interaction en SMA

2.6.1 Interaction

Une des principales propriétés de l'agent dans un SMA est celle d'interagir avec les autres agents. Ces interactions sont généralement définies comme toute forme d'action exécutée au sein du système d'agents et qui a pour effet de modifier le comportement d'un autre agent. Elles permettent aux agents de participer à la satisfaction d'un but global. Cette participation permet au système d'évoluer vers un de ses objectifs et d'avoir un comportement intelligent indépendamment du degré de complexité des agents qui le composent.

En général, les interactions sont mises en œuvre par un transfert d'informations entre agents ou entre les agents et leur environnement, soit par perception, ou bien par communication. Par la perception, les agents détiennent la connaissance de changement de comportement d'un tiers au travers du milieu. Par la communication, un agent fait un acte délibéré de transfert d'informations vers un ou plusieurs autres agents.

Le degré de complexité des connaissances nécessaires pour traiter les interactions dépend des capacités cognitives (de raisonnement) de l'agent et du fait que l'agent a ou non une connaissance de l'objectif du système global. En effet, un agent qui poursuit un objectif individuel au sein du système comme c'est le cas pour les agents dits réactifs ne focalise pas son énergie pour interagir avec les autres même s'il y est amené. Par contre, un agent qui participe à la satisfaction du but global du système tout en poursuivant un objectif individuel, va passer une partie de son temps à coopérer ou à se coordonner avec les autres agents. Pour cela, il doit posséder des connaissances sociales qui modélisent ses croyances sur les autres agents.

Situations d'interaction

Dans les systèmes multi agents, la communication est l'un des moyens utilisés pour échanger des informations entre agents (e.g. plans, résultats partiels, buts, etc.). La capacité de communiquer et, par conséquent, de coopérer des agents s'appuient sur un fond commun d'aptitudes aussi complexes que la perception, l'apprentissage, la planification et le raisonnement. On retrouve ainsi dans la résolution de problèmes des situations d'interaction et des stratégies coopératives non déterministes, difficiles à interpréter et parfois non complètement reproductibles. L'équilibre entre l'autonomie des agents, dotés d'un comportement plus ou moins intelligent, et la convergence vers un objectif global caractérise l'interaction.

L'interaction constitue donc à un niveau d'abstraction supérieur à la notion de communication et d'action. Le problème est alors de savoir combiner ces éléments (communications et actions) afin de coordonner et de contrôler les échanges entre plusieurs agents pour avoir un comportement collectif cohérent du système.

Plusieurs types d'interaction ont été définis et analysés à travers divers composantes (Ferber, 1995). J Ferber donne une classification des situations d'interaction abordées selon le point de vue d'un observateur extérieur. Cette classification présente les différentes situations d'interaction que l'on retrouve en fonction des objectifs des agents (compatibles ou incompatibles), des ressources dont ils disposent (suffisantes ou insuffisantes) et de leurs compétences pour la résolution d'un problème.

Tableau 1: Classification des situations d'interaction

Buts	Ressources	Compétences	Situation
Compatibles	Suffisantes	Suffisantes	Indépendance
Compatibles	Suffisantes	Insuffisantes	Collaboration simple
Compatibles	Insuffisantes	Suffisantes	Encombrement
Compatibles	Insuffisantes	Insuffisantes	Collaboration coordonnée
Incompatibles	Suffisantes	Suffisantes	Compétition individuelle pure
Incompatibles	Suffisantes	Insuffisantes	Compétition collective pure
Incompatibles	Insuffisantes	Suffisantes	Conflits individuels pour des ressources
Incompatibles	Insuffisantes	Insuffisantes	Conflits collectifs pour des ressources

Il aboutit ainsi à une première typologie des situations d'interactions : indépendance, collaboration simple, encombrement, collaboration coordonnée, compétition individuelle pure, compétition collective pure, conflit individuel, conflits collectifs. Le tableau 1 met en évidence l'influence de trois critères (objectifs, compétences, ressources) sur la situation d'interaction induite. En l'occurrence, la compatibilité des objectifs induira des situations de coopération dès que les ressources ou les compétences sont insuffisantes.

2.6.2 Communication et actes de langage

2.6.2.1 Communication

Les communications, dans les systèmes multi agents comme chez les êtres humains, sont à la base des interactions et de l'organisation. Une communication peut être définie comme une forme d'action locale d'un agent vers d'autres agents. Les questions abordées par un modèle de communication peuvent être résumées par l'interrogation suivante : qui communique quoi, à qui, quand, pourquoi, et comment ?

- ***Pourquoi les agents communiquent-ils*** ? La communication doit permettre la mise en œuvre de l'interaction et par conséquent la coopération et la coordination d'actions.

- ***Quand les agents communiquent-ils*** ? Les agents sont souvent confrontés à des situations où ils ont besoin d'interagir avec d'autres agents pour atteindre leurs buts locaux ou globaux. La difficulté réside dans l'identification de ces situations. Par exemple, une communication peut être sollicitée suite à une demande explicite par un autre agent.

- ***Avec qui les agents communiquent-ils*** ? les communications peuvent être sélectives sur un nombre restreint d'agents ou diffusées à l'ensemble des agents. Le choix de l'interlocuteur dépend essentiellement des accointances de l'agent (connaissances qu'a l'agent sur les autres agents).

- ***Comment les agents communiquent-ils*** ? La mise en œuvre de la communication nécessite un langage de communication compréhensible et commun à tous les agents. Il faut identifier les différents types de communication et définir les moyens permettant non seulement l'envoi et la réception de données mais aussi le transfert de connaissances avec une sémantique appropriée à chaque type de message.

2.6.2.2 Actes de langage

La théorie des actes de langage est une théorie de la communication fondée dans les années 1960. Le postulat à la base de la théorie affirme que la production d'un énoncé s'assimile à l'accomplissement d'actions : si agir c'est transformer l'état des choses, parler c'est, également, transformer l'état mental des interlocuteurs. Une conséquence de ce postulat est qu'il doit être possible d'identifier les constituants élémentaires de la communication. Ceux-ci sont appelés *actes de langage.*

J.L. Austin s'est intéressé aux différents types d'actes de langage que constituent les actions intentionnelles effectuées au cours d'une communication. Il distingue trois types d'actes de langage : actes locutoires, actes illocutoires et actes perlocutoires[6].

- Les *actes locutoires* se rapportent à la formulation d'un énoncé. Ces actes sont satisfaits lorsque l'énoncé est correctement formulé.

- Les *actes illocutoires* désignent l'action effectuée sur l'auditeur lors de la formulation de l'énoncé (e.g., donner un ordre, poser une question). Ils sont accomplis avec succès lorsque l'effet attendu sur auditeur est obtenu (e.g., un ordre exécuté, réponse à une question).

- Les *actes perlocutoires* rapportent aux conséquences indirectes visées par les actes locutoires et illocutoires. Ils ne sont pas codés dans l'énoncé et dépendent du contexte dans lequel s'effectue la communication. Ils sont satisfaits lorsqu'ils sont reconnus par l'auditeur (e.g. convaincre, faire croire, etc.).

En tant qu'objet d'étude, la théorie des actes de langage se présente en trois points:

- Répertorier et proposer une taxonomie des différentes formes d'actes,

- Définir un formalisme d'expression et de manipulation des actes (langage de communication),

- Etudier les liens entre les actes et leurs sémantiques en terme de modification d'état mental.

2.7 Les langages de communication dans les SMAs

Pour coopérer, les agents ont besoin de la communication entre eux. Comme pour les êtres humains les agents ont besoin d'un langage commun pour rendre une conversation possible.

L'utilisation d'un langage commun implique que tous les agents comprennent son vocabulaire sous tous ses aspects concernant :

- **La syntaxe**, qui précise le mode de structuration des symboles;

[6] J. Ferber : Les systèmes multi-agents - Vers une intelligence collective. InterEditions, 1995. P320

- **La pragmatique** pour pouvoir interpréter les symboles;

- **L'ontologie** pour pouvoir utiliser les mêmes mots d'un vocabulaire commun.

Il existe un certain nombre de langages qui permettent la communication entre les agents dont on va parler sur deux qui sont le KQML et FIPA ACL.

2.7.1 Le langage KQML

"Knowledge Query and Manipulation Language" (KQML) est un langage "extérieur" de haut niveau pour les agents, il est orienté vers l'échange des messages, indépendant de la syntaxe et de l'ontologie du contenu des messages.

Le langage KQML est indépendant aussi du mécanisme de transport (TCP/IP, e-mail, objets CORBA etc.) et du langage utilisé pour coder le contenu des messages (e.g. Prolog, STEP, SQL, KIF etc.).

2.7.1.1 Les performative de KQML

Exemple type d'un message KQML :

```
(<performative>
         : <attribut_1> <valeur_1>
         : <attribut_2> <valeur_2>
         . . .
         : <attribut_n> <valeur_n>
         )
```

Exemple d'un message :

On peut prendre comme exemple de message KQML, le message d'un agent E (émetteur) qui demande aux autres agents récepteurs (R) de lui fournir le prix d'une imprimante à jet d'encre (la question est placée dans le contenu du message)

```
 (ask-all
         :content (PRIX JET ?prix)
         :sender Ag1
         :reply-with prixJet
         :ontology imprimantes
         :language LPROLOG
         )
```

ask-all est dite *'performative'* dans KQML elle signifie que l'agent E désire que tous les agents R répondent à sa question. .

Les attributs de cette performative sont :

- content : le contenu du message (l'information transportée par la performative).
- sender : c'est l'agent émetteur du message.
- reply-with : identificateur unique du message, en vue d'une référence ultérieur.
- ontology : précise le nom de l'ontologie utilisé dans content.
- language : le nom du langage utilisé dans le contenu du message (content).

Les performatifs de KQML peuvent être classifiés en trois grandes catégories :

1. Discourt : ils permettent l'échange de l'information et de connaissance entre les agents.
2. Interconnexion : permet l'obtention de l'information et de connaissances entre les agents.
3. Exception : sont les performatives de modification de flux de connaissances.

Tableau 2: Les performatives de KQML

Performative	**Signification**
achieve	E veut que R rende quelque chose **vrai** dans leur environnement
advertise	E fait connaître aux autres agents les capacités disponibles dans le traitement d'une performative
ask-about	E veut connaître toutes les propositions pertinentes dans les BVC de R
ask-if	E veut savoir si la réponse à la question précisée en C se trouve dans la BVC de R
ask-one	E veut que seulement R réponde à sa question C
break	E veut que R interrompe le flux établi par pipe
broadcast	**E veut que R transmette à son tour la performative á tous ses connexions**

broker-all	E veut que R collecte toutes les réponses á la performative
broker-one	E veut que R l'aide à répondre à une performative
deny	la performative ne peut pas être appliquée à E
delete-all	E veut que R efface toutes les propositions qui s'apparie avec C de sa BVC
delete-one	E veut que R efface une proposition qui s'apparient avec C de sa BVC
discard	E ne veut pas le reste des réponses de R à une interrogation
eos	termine un flux de réponses (en anglais stream) à une interrogation (en anglais eos vient de 'end of stream')
error	E considère le message précédent de R comme mal formé
evaluate	E veut que R évalue (simplifie) C
forward	E veut que R transmette le message vers un autre agent
generator	la même signification que standby de stream-all
insert	E demande ã R d'ajouter C à sa BVC
monitor	E veut que R actualise sa réponse à un stream-all
next	E veut la réponse suivante de R à un flux d'une performative
pipe	E veut que R transmette toutes les performatives futures à un autre agent
ready	E est prêt pour répondre à la performative mentionnée précédemment par R
recommend-all	E veut tous les noms des agents qui peuvent répondre à C
recommend-one	E veut le nom d'un agent qui peut répondre à C
recruit-all	E veut que R 'recrute' tous les agents capables de lui répondre à C
recruit-one	E veut que R 'recrute' un agent capable de lui répondre à C
register	E peut transmettre des performatives à un certain agent
reply	communique une réponse attendue
rest	E veut le reste des réponses de R à une interrogation nommée précédemment
sorry	R ne peut pas fournir plus d'information
standby	E veut que R soit prêt pour répondre à une performative
stream-about	une version réponse multiple de ask-about

stream-all	une version réponse multiple de ask-all
subscribe	E veut que R actualise par la suite sa réponse
tell	E affirme au R que C est dans la BVC de E
transport-address	E associe un nom symbolique à une adresse de transport
unadvertise	l'action contraire de advertise
unregister	l'action contraire de register
untell	E affirme à R que C n'est pas dans la BVC de E

2.7.1.2 Facilitateurs KQML

Les facilitateurs représentent une classe spécifique d'agents qui distribuent des meta-informations sur les autres agents et offrent des services de communication tels que les suivants :

- Retransmission et distribution des messages (en anglais : message forwarding and broadcasting).
- Découverte des ressources.
- routage basé sur le contenu du message.
- Appariement.

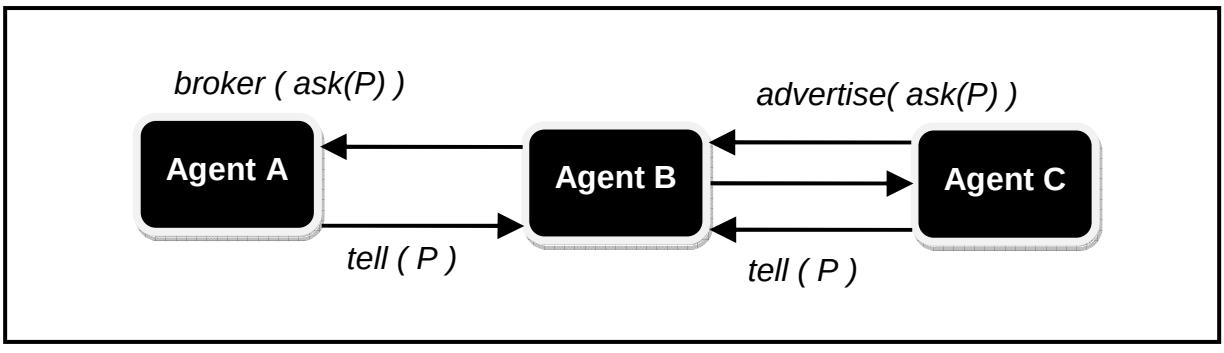

Figure 7 : Schéma des messages échangés entre les agents A et C en utilisant les facilitateurs

2.7.2 Le langage FIPA ACL

FIPA (*Foundation for Intelligent Physical Agents*) organisation qui a été crée en 1996. Parmi ses préoccupations, une place importante concerne l'élaboration des spécifications du langage de communication entre agents FIPA-ACL.

Ayant une syntaxe similaire à KQML le langage s'appuie sur la définition de deux ensembles:

- Un ensemble d'actes de communication primitifs, auquel s'ajoutent les autres actes de communication pouvant être obtenus par la composition des ces actes de base.

- Un ensemble de messages prédéfinis que tous les agents peuvent comprendre.

FIPA-ACL possède 21 actes communicatifs, exprimés par des performative, qui peuvent être groupés selon leur fonctionnalité de la façon suivante :

1. Passage d'information : inform*, inform-if (macro act), inform-ref (macro act), confirm*, disconfirm*

2. Réquisition d'information : query-if, query-ref, subscribe.

3. Négociation : accept-proposal, cfp, propose, reject-proposal.

4. Distribution de tâches (ou exécution d'une action) : request*, request-when, request-whenever, agree, cancel, refuse.

5. Manipulation des erreurs : failure, not-understood.

En FIPA-ACL il n'existe pas de primitives de gestion ni de facilitation.

Les actes communicatifs peuvent être primitifs ou composés. Les actes communicatifs primitifs sont définis de façon atomique, c'est-à-dire qu'ils ne sont pas définis à partir d'autres actes (dans la classification ci-dessus ils sont suivis d'une étoile "*"). En revanche, les actes communicatifs composés sont définis à partir d'autres actes par l'une des opérations suivantes :

- Un acte fait partie du contenu d'un autre acte ; à travers l'opérateur de composition " ; " pour indiquer une séquence d'actions.

- A travers l'opérateur de composition " | " pour indiquer un choix non déterministe de l'action.

Un message comprend plusieurs éléments qui sont présentés dans ce tableau.

Tableau 3 : Structure d'un message FIPA-ACL

Elément	**Signification**
performative	type de l'acte communicatif
sender	l'émetteur du message
receiver	le destinataire du message
reply-to	participant à l'acte de communication
content	le contenu du message (l'information transportée par la performative)
language	le langage dans lequel le contenu est représenté
encoding	décrit le mode d'encodage du contenu du message
ontology	le nom de l'ontologie utilisé pour donner un sens aux termes utilisés dans le *content*
protocol	contrôle la conversation
conversation-id	identificateur de la conversation
reply-with	identificateur unique du message, en vue d'une référence ultérieure
in-reply-to	référence à un message auquel l'agent est entrain de répondre (précisé par l'attribut reply-with de l'émetteur)
reply-by	impose un délai pour la réponse

On peut observer que le contenu des deux premières catégories des performatives, passage et réquisition d'information exprime en fait une proposition, le contenu de la catégorie3, une négociation, une action et une proposition, ainsi que la catégorie4, une distribution de tâches, et certainement une action.

Exemple

```
(inform
         :sender A
         :receiver B
         :reply-with laptop
         :language KIF
         :ontology ordinateurs
         :content (=(prix HP-Jet) (scalar 1500 USD))
         :reply-by 10
         :conversation-id conv01
         )
```

2.8 Plateformes multi agents

Il y a deux grands groupes de méthodologies pour le développement orienté agent. Le premier groupe étend ou adapte les méthodologies orientées objet pour mettre en œuvre les caractéristiques des agents. Beaucoup de méthodologies orientées objet sont utilisées pour développer des systèmes multi agents dans le domaine industriel avec succès, Cependant les extensions et les modifications des méthodologies orientées objet doivent prendre en compte les différences qui existent entre les objets et les agents. Pour le deuxième groupe, il part des méthodologies qui ont été dédiées à la modélisation des systèmes à bases de connaissances.

On peut aussi trouver un certain nombre de plateformes dédiés au développement de systèmes multi agents, dont certain logiciels libres, la plus connue est JADE. On va présenter quelque unes entre eux.

2.8.1 La plate-forme JADE

JADE (Java Agent Development Framework - Bellifemine, Poggi, Rimassa, 1999) est une plate-forme multi-agents développée en Java par CSELT (Groupe de recherche de Gruppo Telecom, Italie) qui a comme but la construction des systèmes multi-agents et la réalisation d'applications conformes à la norme FIPA (FIPA, 1997). JADE comprend deux composantes de base : une plate-forme agents compatible FIPA et un paquet logiciel pour le développement des agents Java.

Le but de JADE est de simplifier le développement des systèmes multi-agents en conformité avec la norme FIPA pour réaliser des systèmes multi-agents interopérables. Pour atteindre ce but, JADE offre la liste suivante de **caractéristiques** au programmeur d'agents :

- **La plate-forme multi-agents compatible FIPA**, qui inclut le Système de Gestion d'Agents (AMS), le Facilitateur d'Annuaire (DF), et le Canal de Communication entre Agents (ACC). Ces trois agents sont automatiquement créés et activés quand la plate-forme est activée.

- **La plate-forme d'agents distribuée**. La plate-forme d'agents peut être distribuée sur plusieurs hôtes, à condition qu'il n'y ait pas de pare-feu entre ces hôtes. Une seule application Java, et donc une seule Machine Virtuelle Java, est exécutée sur chaque hôte. Les agents sont implémentés comme des threads d'exécution Java et les événements Java sont utilisés pour la communication efficace et légère entre agents sur un même hôte. Un agent peut exécuter des tâches parallèles et JADE planifie ces tâches d'une manière plus efficace (et même plus simple pour le programmeur) que la planification faite par la Machine Virtuelle Java pour les threads d'exécution.

- **Un certain nombre de DF** (Facilitateurs d'Annuaire) compatibles FIPA qui peuvent être activés quand on lance la plate-forme pour exécuter les applications multi-domaines, où la notion de domaine est la notion logique décrite par le document FIPA97 dans sa Partie 1.

- **Une interface de programmation** pour simplifier l'enregistrement de services d'agents avec un ou plusieurs domaines de type DF.

- **Le mécanisme de transport et l'interface** pour l'envoi et la réception des messages.

- **Le protocole IIOP** compatible avec le document FIPA97 pour connecter des plates-formes multi-agents différentes.

- **Le transport léger de messages ACL** sur la même plate-forme d'agents. Dans le but de simplifier la transmission, les messages internes (sur la même plate-forme) sont transférés codés comme des objets Java et non comme des chaînes de caractères. Quand l'expéditeur ou le récepteur n'appartient pas à la même plate-forme, le message est automatiquement converti à/du format de chaîne de caractères spécifiés par la FIPA. De cette façon, la conversion est cachée au programmeur d'agents, qui a seulement besoin de traiter la classe d'objets Java.

- **Une bibliothèque de protocoles** d'interaction compatibles FIPA.

- **L'enregistrement automatique d'agents** dans le Système de Gestion d'Agents (AMS).

- **Un service d'attribution de noms compatible FIPA** ; quand on lance la plate-forme, un agent obtient un identificateur unique (Globally Unique Identifier - GUID).

- **Une interface graphique utilisateur** pour gérer plusieurs agents et plates-formes multi-agents en partant d'un agent unique. L'activité de chaque plate-forme peut être supervisée et enregistrée.

2.8.2 La plate-forme Mace

MACE (Gasser e.a., 1987) est le premier environnement de conception et d'expérimentation de différentes architectures d'agents dans divers domaines d'application. Dans MACE, un agent est un objet actif qui communique par envoi de messages. Les agents existent dans un environnement qui regroupe tous les autres agents et toutes les autres entités du système. Un agent peut effectuer trois types d'actions : changer son état interne, envoyer des messages aux autres agents et envoyer des requêtes au noyau MACE pour contrôler les événements internes. Chaque agent est doté d'un moteur qui représente la partie active de l'agent. Ce moteur détermine l'activité de l'agent et la façon dont les messages sont interprétés. MACE a été utilisé pour développer des simulations d'applications distribuées.

2.8.3 La plate-forme ZEUS

ZEUS (Nwama e.a., 1999) est une plate-forme multi-agents conçue et réalisée par British Telecom (Agent Research Programme of BT Intelligent Research Laboratory) pour développer des applications collaboratives. ZEUS est écrit dans le langage Java et il est fondé sur les travaux de la FIPA. L'architecture des agents regroupe principalement les composantes suivantes :

- Une boîte aux lettres et un gestionnaire de messages qui analyse les messages de la boîte aux lettres et les transmet aux composantes appropriées ;

- Un moteur de coordination ;

- Un planificateur qui planifie les tâches de l'agent en fonction des décisions du moteur de coordination, des ressources disponibles et des spécifications des tâches ;

- Plusieurs bases de données représentant les plans connus par l'agent, les ressources et l'ontologie utilisée ;

- Un contrôleur d'exécution qui gère l'horloge locale de l'agent et les tâches actives.

L'environnement comporte trois bibliothèques : une avec des agents utilitaires, une avec des outils pour la construction des agents, et une autre avec des composants agents.

ZEUS met un fort accent sur la méthodologie de développement, fondée sur la notion de rôle. ZEUS a été utilisé pour développer plusieurs applications réelles comme les ventes aux enchères et la simulation de la fabrication des ordinateurs. Les caractéristiques des domaines d'applications de ZEUS ont été définies par les concepteurs; parmi ces caractéristiques, on peut mentionner :

- Chaque agent crée un plan qui nécessite un raisonnement explicite pour atteindre son but ;

- La résolution de problèmes nécessite une coopération entre agents ;

- Le rôle de chaque agent consiste à contrôler un système externe qui réalise une tâche du domaine d'application, la résolution de problème est ainsi effectuée par ce système externe et contrôlée par les agents.

2.8.4 La plate-forme MADKIT

MADKIT (Madkit, 2003) est une plate-forme développée par le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) de l'Université Montpellier II. MADKIT est libre pour l'utilisation dans l'éducation. Il est écrit en Java et est fondé sur le modèle organisationnel Alaadin. Il utilise un moteur d'exécution où chaque agent est construit en partant d'un micro-noyau. Chaque agent a un rôle et peut appartenir à un groupe. Il y a un environnement de développement graphique qui permet facilement la construction des applications.

2.8.5 La plate-forme SWARM

SWARM (Minar e.a., 1996) est une plate-forme multi-agents avec agents réactifs. L'inspiration du modèle d'agent utilisé vient de la vie artificielle. SWARM est l'outil privilégié de la communauté américaine et des chercheurs en vie artificielle. L'environnement

offre un ensemble de bibliothèques qui permettent l'implémentation des systèmes multi-agents avec un grand nombre d'agents simples qui interagissent dans le même environnement. De nombreuses applications ont été développées à partir de SWARM qui existe aujourd'hui implémenté en plusieurs langages (Java, Objective-C).

2.9 Conclusion

Au lieu d'adapter l'esprit humain à la pensée de la machine en exprimant nos idées d'une façon séquentiel, les visions actuelles et future dans les systèmes informatiques vont changer de méthode en rendant les machines se comporter selon ce que nous pensons. Une vision future de J Ferber est : « En l'an 2045, tout ce qui constitue notre culture informatique actuelle sera dépassé. L'intelligence artificielle n'existera peut-être plus en tant que discipline à part, si ce n'est comme étude de la "psycho-socio-biologie des artefacts". Elle sera totalement fusionnée et amalgamée avec tous les secteurs de l'informatique et en collaboration avec les sciences physiques, biologiques, sociales et psychologiques. La structure des ordinateurs aura changé. La notion même de processeur central aura totalement disparu. Chaque ordinateur sera constitué de plusieurs milliers de petits processeurs travaillant ensemble, constituant ainsi le "milieu" dans lequel les petites entités des programmes évolueront. ».

La résolution de problèmes ne va plus donc se reposer sur des visions classiques globales et centralisées, mais plutôt sur une vision locale, basée sur interactions entre entités autonome, résolvant le problème global par émergence de fonctionnalités simples. Cette conversion va changer toute notre façon de concevoir et même de penser pour la résolution des actuels problèmes.

Cette vision du futur suppose que les logiciels puissent coopérer harmonieusement à l'instar des humains, qui apprennent par "osmose", c'est-â-dire par confrontation avec la réalité, en imitant, en essayant et en modifiant des comportements "pour voir ce que ça fait", les logiciels de l'avenir devront être en contact avec nous, en regardant et en apprenant à partir des flux d'informations qui circulent normalement entre les humains. [7]

[7] J. Ferber : Les systèmes multi-agents - Vers une intelligence collective. InterEditions, 1995. P467

Chapitre 3 : Les réseaux de Petri

3.1 Introduction

Historiquement, le concept des réseaux de Petri a ses origines dans les travaux de Carl Adam Petri, soumis en 1962 dans la faculté des mathématiques et physique, dans l'université technique de Darmstadt, dans l'ouest d'Allemagne. La thèse a été préparée lorsque C. A. Petri travail comme chercheur à l'université de Bonn.de 1970 à 1975, le Computation Structure Groupe en MIT a activé pour conduire des travaux en relation avec les réseaux de Petri, et produit plusieurs rapports et thèses en relation avec les réseaux de Petri.

Les réseaux de Petri présentent un outil graphique et mathématique de modélisation. Ils sont utilisés pour la description et l'étude des systèmes en cours d'exécution qui sont concurrents, asynchrones, distribués, parallèles, non déterministes, et/ou stochastiques.

Comme un outil graphique, les réseaux de Petri constituent un aide pour la communication visuelle. En plus les jetons sont utilisés dans ces réseaux pour simuler les activités dynamiques et concurrentes des systèmes. Comme un outil mathématique il est possible de faire des équations d'état, des équations algébriques, et d'autres modèles mathématiques pour gérer le comportement de système. Les réseaux de Petri peuvent être utilisés indifféremment par les théoriciens ou par les praticiens, car ils offrent un médium puissant de communication entre eux.

3.2 Définitions

3.2.1 Un réseau de Petri

Un réseau de Petri est un graphe biparti caractérisé par deux types de nœuds : des places (représentés par des cercles) et transitions représentés (représentés par des barres). Ces nœuds sont connectés par des arcs (représentés par des flaches) allant des places vers les transitions et l'inverse.

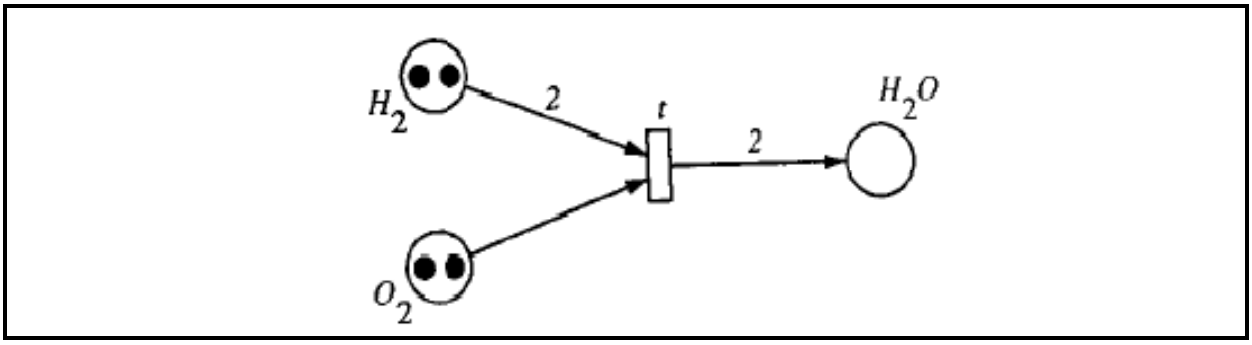

Figure 8: Un réseau de Petri représentant une réaction chimique de la composition de l'eau

Les arcs sont libellés par leurs poids (entier positive), tel que le poids *n* d'un arc représente *n* arcs parallèles. Pour les places ils portent un ensemble de jetons qui représentent le marquage de la place. Le marquage de réseau est représenté par un vecteur de taille égale au nombre de places dans le réseau tel que : *M(P)* représente le nombre de jetons dans la place *P*.

Dans la modélisation de systèmes, on utilise le concept de condition et le concept d'évènement. Tel que les places représentent les conditions (par les jetons qu'ils contiennent) et les transitions représentent les évènements. Une transition (un évènement) à un ensemble des places d'entrée et un ensemble de places de sorties, qui représentent ses pré et post condition d'exécution. Une autre interprétation possible est celle qui considère le nombre de jetons comme le nombre d'unités de ressources disponibles.

Et on peut donner une définition formelle d'un réseau de petri comme suit :

Un réseau de Petri est un 5-tuple, PN=(P,T,F,W,M_0) où :

- P=$\{p_1,p_2, \dots ,p_m\}$ un ensemble fini de places ;
- T=$\{t_1,t_2, \dots ,t_n\}$ un ensemble fini de transitions ;
- $F \subseteq (P \times T) \cup (T \times P)$ ensemble d'arcs ;
- $W: F \rightarrow \{1,2,3, \dots\}$ une fonction de poids.
- $M_0: P \rightarrow \{0,1,2,3, \dots\}$ le marquage initial ;

  $P \cap T = \emptyset \; et \; P \cup T \neq \emptyset$.

Un réseau de Petri (*P, T, F, W*) sans aucun marquage initiale spécifique est noté N.

Un réseau de Petri avec un marquage initial donné M_0 est noté par (N, M_0).

3.2.2 Transition source et transition puits

Une transition sans aucune place d'entrée est appelée **transition source**. Une transition source est toujours franchissable. Une **transition puits** est une transition sans place de sortie, une transition puits consomme des jetons et ne rien produire.

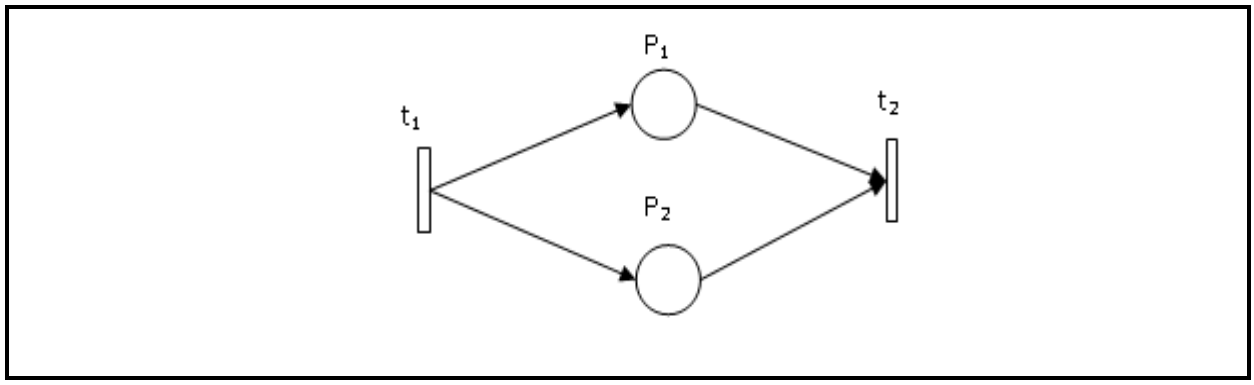

Figure 9 : La transition t1 est une transition source et t2 est une transition puits

3.2.3 Boucle

Une paire de place p et de transition t est appelé boucle si p est la place d'entrée et de sortie de t. un réseau de Petri est pure lorsque il n'a pas de boucle.

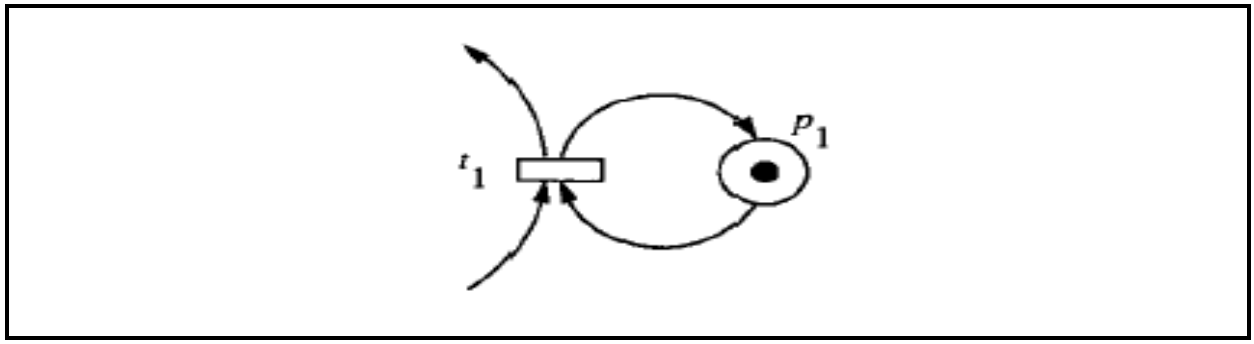

Figure 10: Une boucle

3.3 Propriétés des réseaux de Petri

Après la modélisation des systèmes par réseaux de Petri une question qui se pose « que ce qu'on peut faire avec les modèles », un point très fort des réseaux de Petri est sont support de l'analyse de plusieurs propriétés et problèmes associés aux systèmes concurrents. Deux types de propriétés peuvent êtres étudies par modèle de réseau de Petri : ceux qui sont dépendent du marquage initiale ou les propriétés comportementales, et ceux qui sont indépendant du marquage initiale, qui sont les propriétés structurelles.

3.3.1 Règles de franchissement

Le marquage d'un réseau de Petri change suivant les règles suivantes :

1) Une transition *t* est franchissable si toutes les places d'entrée *p* de *t* sont marqués au moins par w(p,t) jetons, où w(p,t) est le poids de l'arc allant de *p* vers *t*.

2) Une transition franchissable peut comme elle peut ne pas être franchie (sa dépend de l'arrivée ou non de l'évènement correspondent).

3) Le franchissement d'une transition franchissable *t* élimine w(p,t) jetons de chaque place *p* d'entrée de *t*, et ajoute w(t,q) jetons à chaque place de sortie q de *t*, tel que w(t,q) est le poids de l'arc allant de *t* vers q.

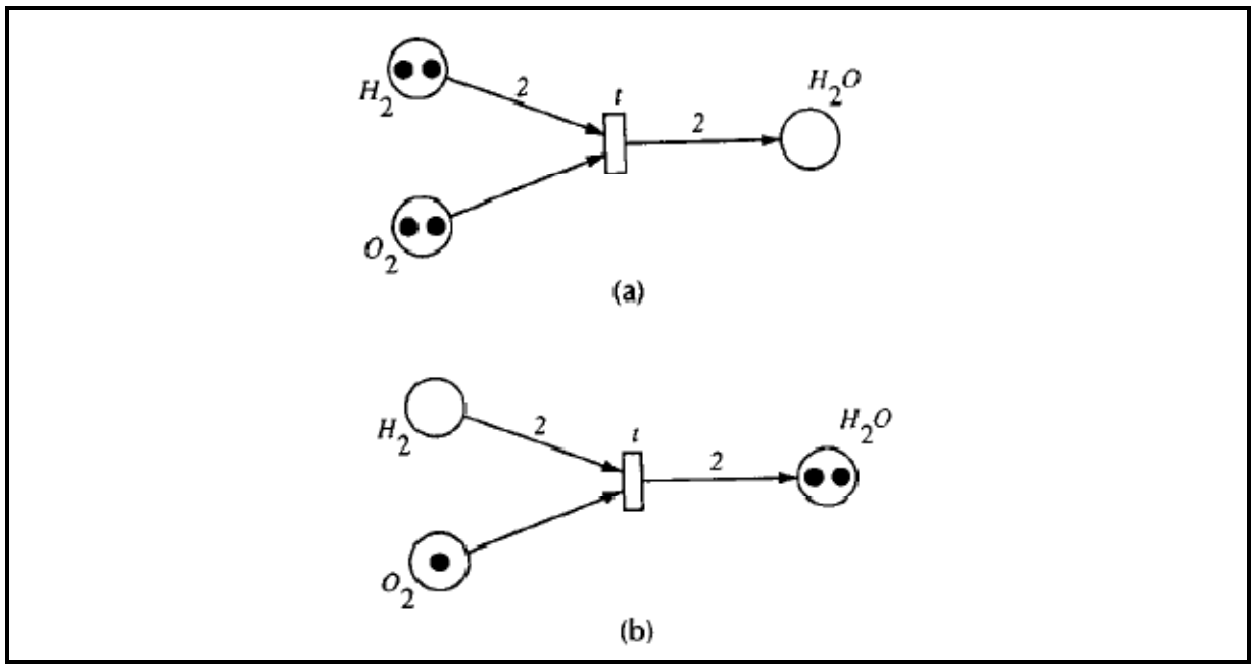

Figure 11: Franchissement d'une transition

3.3.2 Accessibilité (Reachability)

L'accessibilité est une base fondamentale pour l'étude des propriétés dynamiques de tout un système. Le franchissement d'une transition va changer la distribution des jetons (le marquage) dans le réseau suivant les règles de franchissement expliqués précédemment. Une suite de marquages résulte d'une séquence de franchissements. Un marquage M_n est accessible depuis le marquage M_0, s'il y a une séquence de franchissement qui transforme M_0 vers M_n. Une séquence de franchissement est notée $\partial = M_0\ t_1\ M_1\ t_2\ M_2\ \dots\ t_n\ M_n$, ou simplement $\partial = M_0\ M_1\ M_2\ \dots\ M_n$. Dans ce cas M_n est accessible depuis M_0 par ∂ et on écrit $M_0\ [\ \partial > M_n$. l'ensemble de tous les marquages accessible depuis M_0 dans un réseau (N, M_0) est noté par $R(N, M_0)$ ou simplement $R(M_0)$. L'ensemble de tous les séquences de franchissement dans un réseau (N, M_0) est noté par $L(N, M_0)$ ou simplement $L(M_0)$.

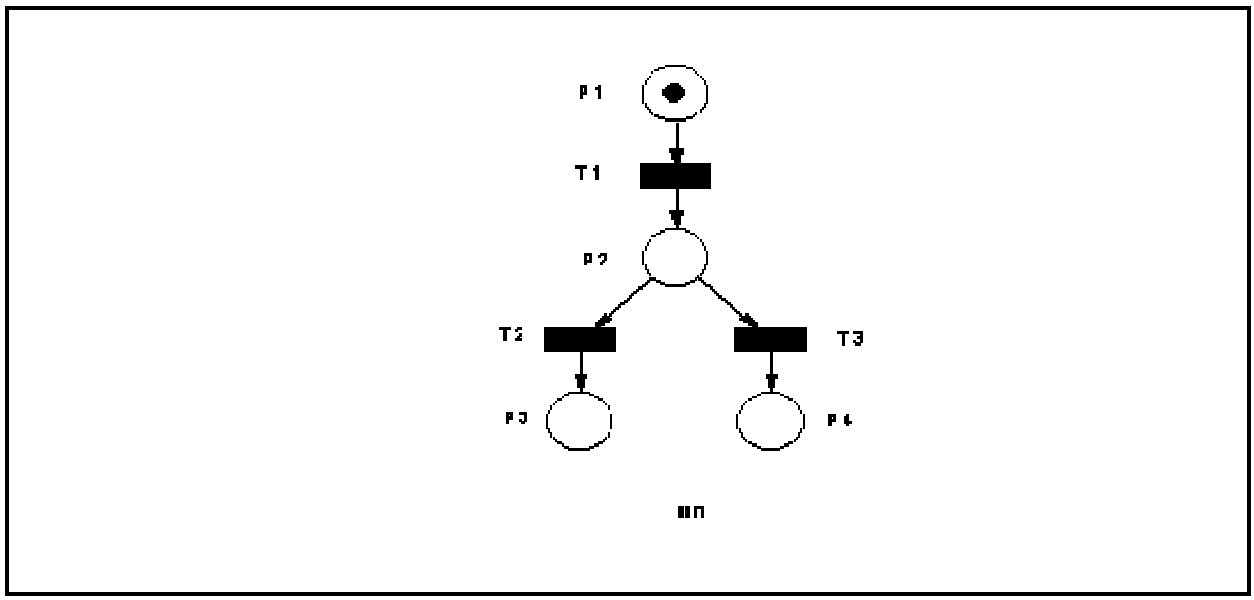

Figure 12: Marquages accessibles

Dans la figure ci-dessus l'ensemble de marquages accessibles depuis le marquage initial M_0 [1,0,0,0] est $R(M_0)=\{M_1, M_2, M_3\}$, tel que :

- M_1=[0,1,0,0].
- M_2=[0,0,1,0].
- M_3=[0,0,0,1].

3.3.3 Bornitude

Un réseau de Petri (N, M_0) est k-borné ou simplement borné si le nombre de jetons dans chaque place ne peut pas dépasser un nombre fini K, pour tout marquage accessible depuis le marquage initiale M_0 : $\forall\, p \in P\ et\ \forall\, M \in R(M_0)/M(p) \leq k$. Un réseau de Petri est sain si il est 1-borné.

Les places sont utilisées souvent dans les réseaux de Petri pour représenter des buffers et registres de stockage de données intermédiaires. La vérification que le réseau de Petri est borné ou sain, est une garantie qu'il n'y aura pas un débordement dans les buffers et les registres, quelque soit la séquence de franchissement.

3.3.4 Vivacité

Le concept de vivacité est étroitement lié à l'absence complète d'impasse dans les systèmes d'exploitation. Un réseau de Petri (N, M_0) est vivant (ou d'une manière équivalent M_0 est un marquage vivant de N) si, quelque soit le marque obtenu du marquage initiale M_0, il est possible de franchir n'import quelle transition dans le réseau par progression à travers une séquence de franchissement. Cela signifie qu'un réseau de Petri vivant garantie l'absence d'impasse, quelque soit la séquence de franchissement choisie. Un exemple de réseau de Petri vivant est illustré dans la figure ci-dessous.

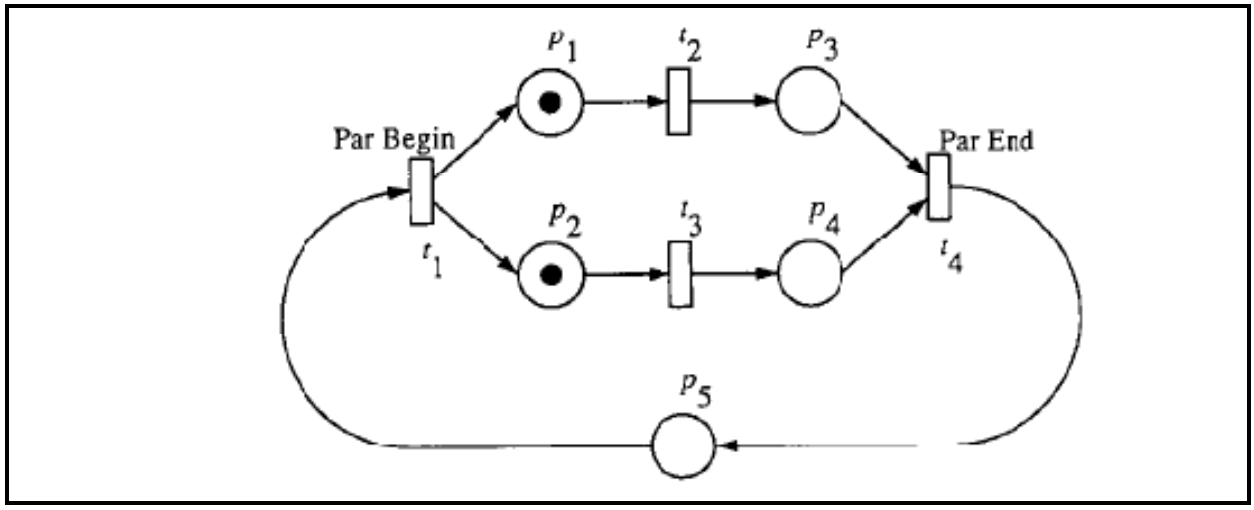

Figure 13: Réseau de Petri vivant

Et un autre exemple de réseau de Petri non vivant et celui de la figure suivante, dans cet exemple aucune transition ne sera franchissable si t_1 est franchie en premier.

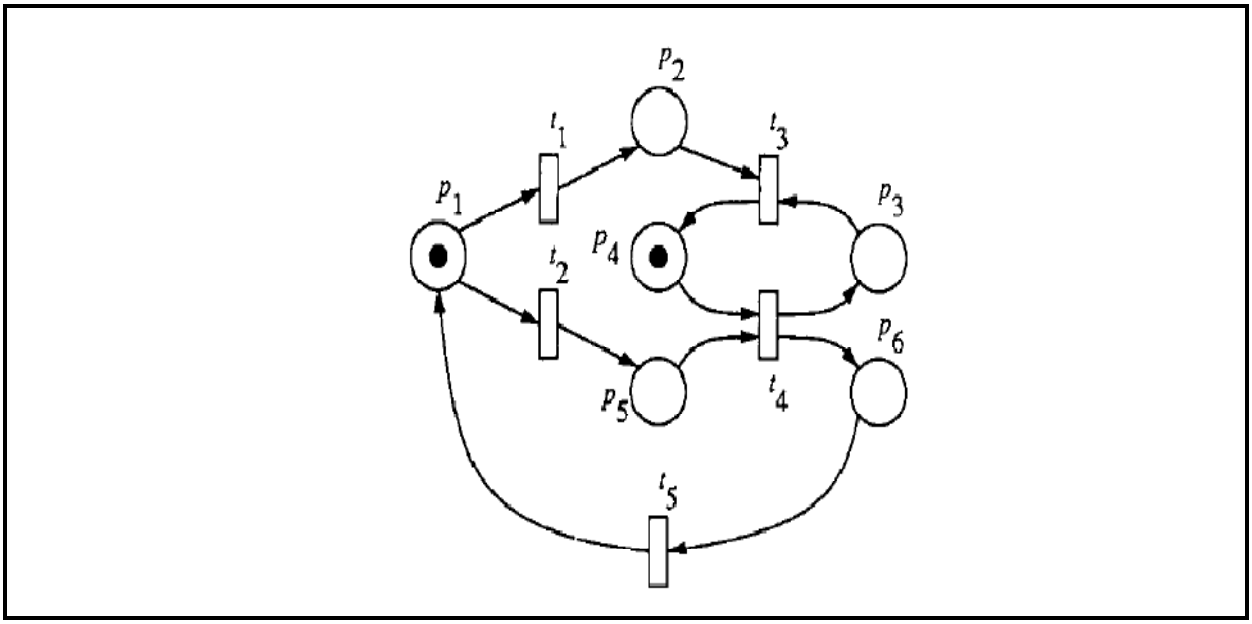

Figure 14: Réseau de Petri non vivant

La vivacité est une propriété idéale pour plusieurs systèmes. Cependant, il est non praticable, et trop couteux de vérifier cette propriété puissante pour certains système tel que les systèmes d'exploitation. Et pour cela on définie plusieurs niveaux de vivacité. Une transition t dans un réseau de Petri (N, M_0) est :

0. Morte (L0-vivante) si *t* ne peut jamais être franchie, dans aucune séquence de franchissement dans L(M_0).

1. L1-vivante, si elle peut être franchie au moins une fois dans quelques séquences de franchissement dans L(M_0).

2. L2-vivante si, étant donné un entier positive k, t peut être au mois franchie k fois dans quelques séquences de franchissement dans L(M_0)..

3. L3-vivante si t apparait infiniment, dans quelques séquences de franchissement dans L(M_0).

4. L4-vivante ou vivante si t est L1-vivante pour tout marquage M dans R(M_0).

Un réseau de Petri (N, M_0) est Lk-vivant si chacune de ses transitions est Lk-vivante, k=0,1,2,3,4.

L4-vivant est le plus fort, et donc : L4-vivant⇒ L3-vivant⇒ L2-vivant⇒ L1-vivant. Une transition est strictement Lk-vivante, si elle Lk-vivant et ne pas L(k+1)-vivante, k=1,2,3.

Dans la figure suivante les transitions sont comme suit :

- t_0 : L_0-live.
- t_1 : L_1-live.
- t_2 : L_2-live.
- t_3 : L_3-live.

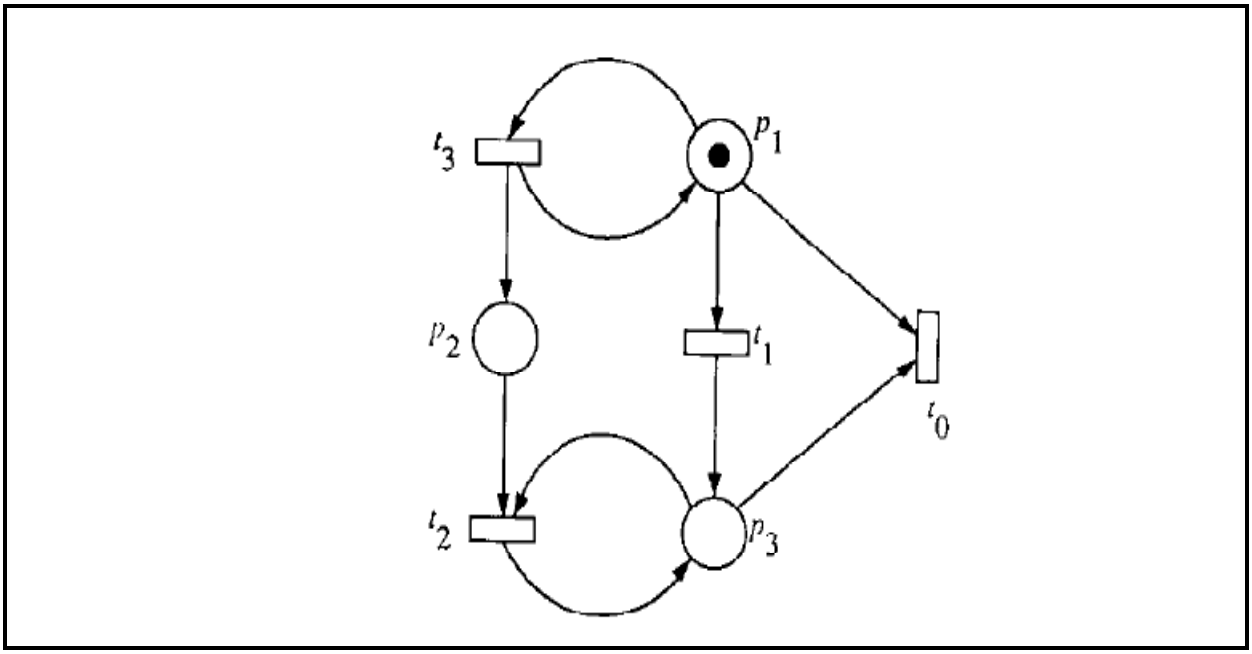

Figure 15: Réseau de Petri avec des transitions de différents niveaux de vivacité

3.3.5 Réversibilité et état d'accueil

Un réseau de Petri (N, M_0) est réversible si, pour tout marquage M dans $R(M_0)$, M_0 est accessible depuis M. Dans un réseau de Petri réversible il est toujours possible de retourner en arrière vers l'état initiale. On défini un état d'accueil par : un marquage M' est un état d'accueil si pour tout marquage M dans $R(M_0)$, M' est accessible depuis M.

3.3.6 Couverture

Un marquage M dans un réseau de Petri (N, M_0) est couverable, s'il existe un marquage M' dans $R(M_0)$ tel que $M'(p) \geq M(p)$ pour chaque p dans le réseau.

3.3.7 Persistance

Un réseau de Petri (N, M_0) est persistant si, pour toutes deux transitions franchissables, le franchissement d'une transition ne va pas désactiver l'autre. Une transition dans un réseau de Petri persistant, une fois être franchissable, reste franchissable jusqu'à son franchissement. La notion de persistance est utile dans le contexte des systèmes parallèles. Le réseau de Petri de la figure suivante est persistant.

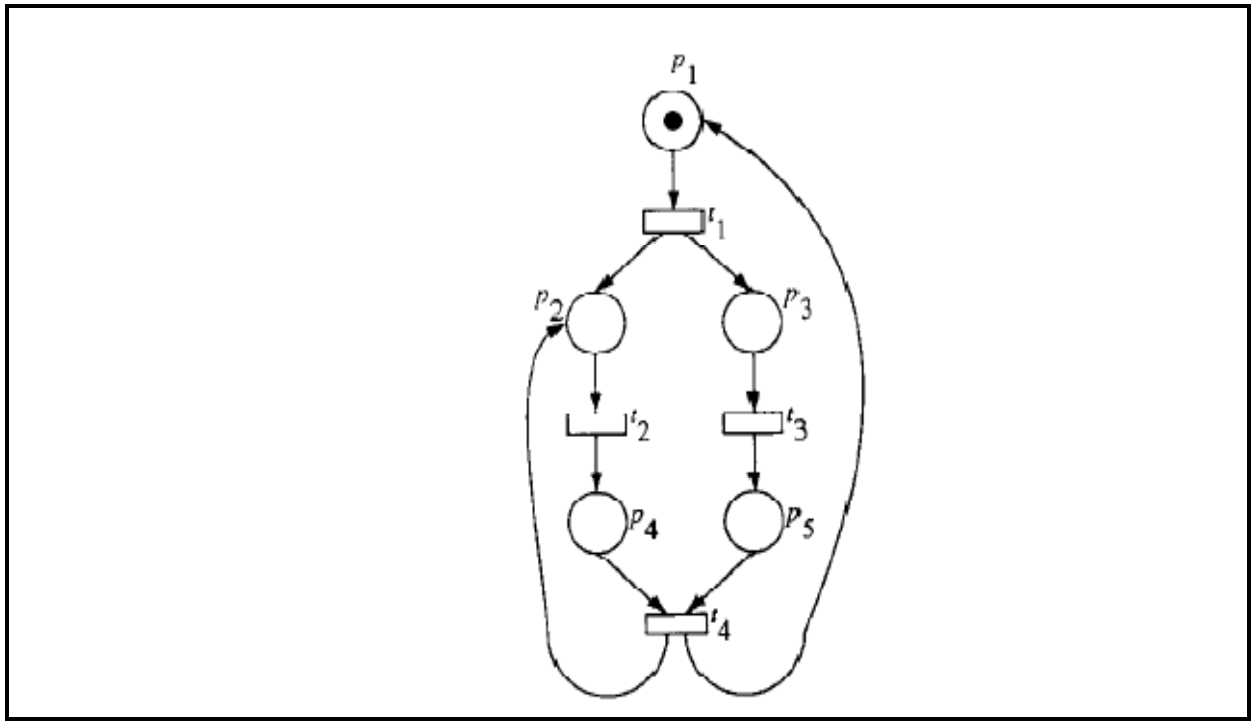

Figure 16: Réseau de Petri persistant

3.4 Extensions des réseaux de Petri

3.4.1 Les réseaux de Petri généralisés

Jusqu'à présent le franchissement de transition ne consomme qu'un jeton par place. Il est possible de donner une valeur aux arcs du réseau en indiquant un poids. De tels réseaux sont dits généralisés. Le franchissement d'une transition peut alors consommer plusieurs marques (selon le poids de l'arc) dans chaque place en entrée et en produire plusieurs dans chaque place en sortie.

3.4.2 Les réseaux de Petri colorés

La modélisation d'un système réel peut mener à des réseaux de Petri de taille trop importante rendant leur manipulation et/ou leur analyse difficile. La question est alors de modifier (étendre) la modélisation par Réseau de Petri de façon à obtenir des modèles Réseau de Petri de plus petite taille.

C'est pour cette raison qu'une autre classe de Réseau de Petri a été introduite : les Réseaux de Petri colorés. Les réseaux de Petri colorés sont importants pour la modélisation de systèmes de production (au sens large). Ils ont été motivés par le fait suivant.

Une taille trop importante peut découler du fait que l'on ne peut pas distinguer entre elles les différentes marques d'une place. Plusieurs marques dans une place peuvent modéliser un certain nombre de pièces identiques dans un stock. Si le stock contient plusieurs types de pièces, des places supplémentaires doivent être introduites pour sa modélisation.

Dans la Figure suivante Le nombre de marques dans la place P3 correspond au nombre de pièces de type 1 en stock. Le nombre de marques dans la place P4 correspond au nombre de pièces de type 2 en stock. Le nombre de marques dans la place P7 indique le nombre d'emplacements libres dans le stock. Si un stock est susceptible de contenir deux (respectivement n) types de pièces, il doit donc être modélisé par au moins 3 (resp. n + 1) places. Peut-on imaginer une modification du modèle Réseau de Petri qui permette de la modélisé par un nombre de places indépendant du nombre de types de pièces stockées.

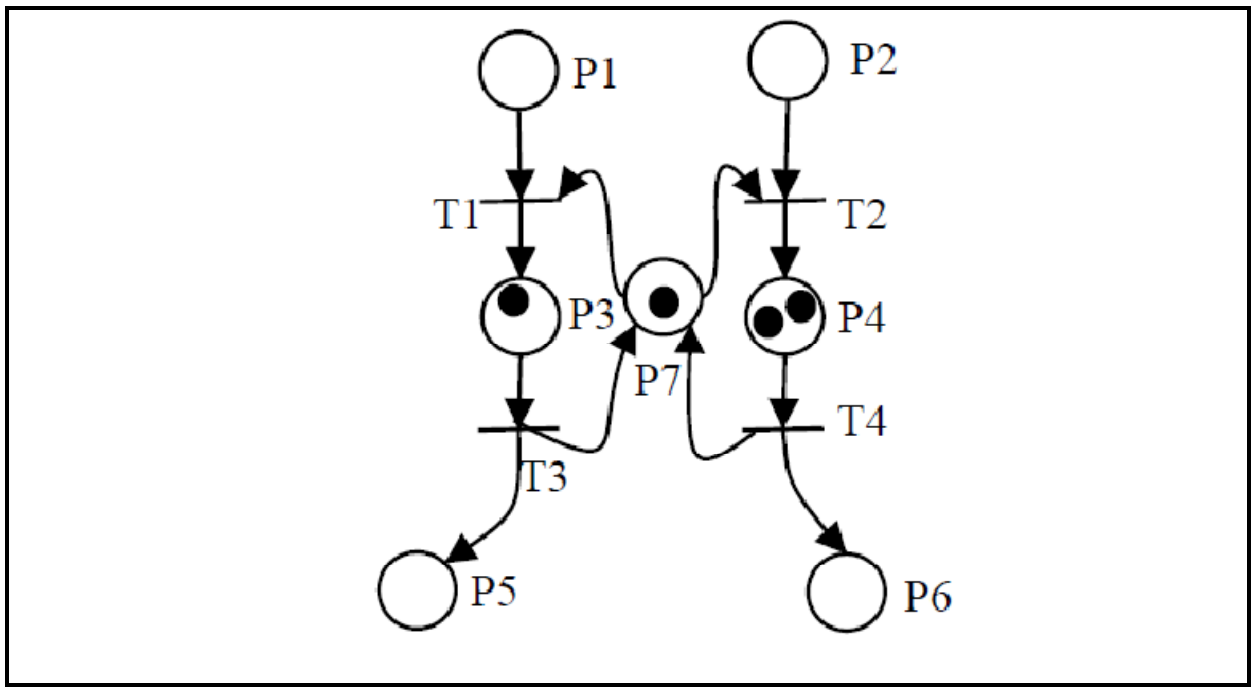

Figure 17: Modélisation d'un stock

3.4.3 Les réseaux de Petri temporisés

Le comportement de ces modèles dépend du temps. Ils permettent d'évaluer les performances temporelles d'un système. Par exemple, il peut s'agir d'évaluer le temps d'exécution d'une série d'opérations (par exemple dans un procédé de fabrication industriel) en fonction de la capacité des machines et/ou des stocks, etc.…

3.4.4 Les réseaux de Petri avec arc inhibiteur

Une autre extension des réseaux ordinaires consiste à permettre de tester l'absence de marques dans une place, alors que lors d'un franchissement classique, on vérifie au contraire la présence d'une marque qui est consommée. Lorsqu'une place en entrée est reliée à une transition par un arc inhibiteur, cette transition n'est franchissable que si la place est vide (à ceci peut s'ajouter les conditions sur les autres places naturellement). Lors du franchissement la place en question reste vide.

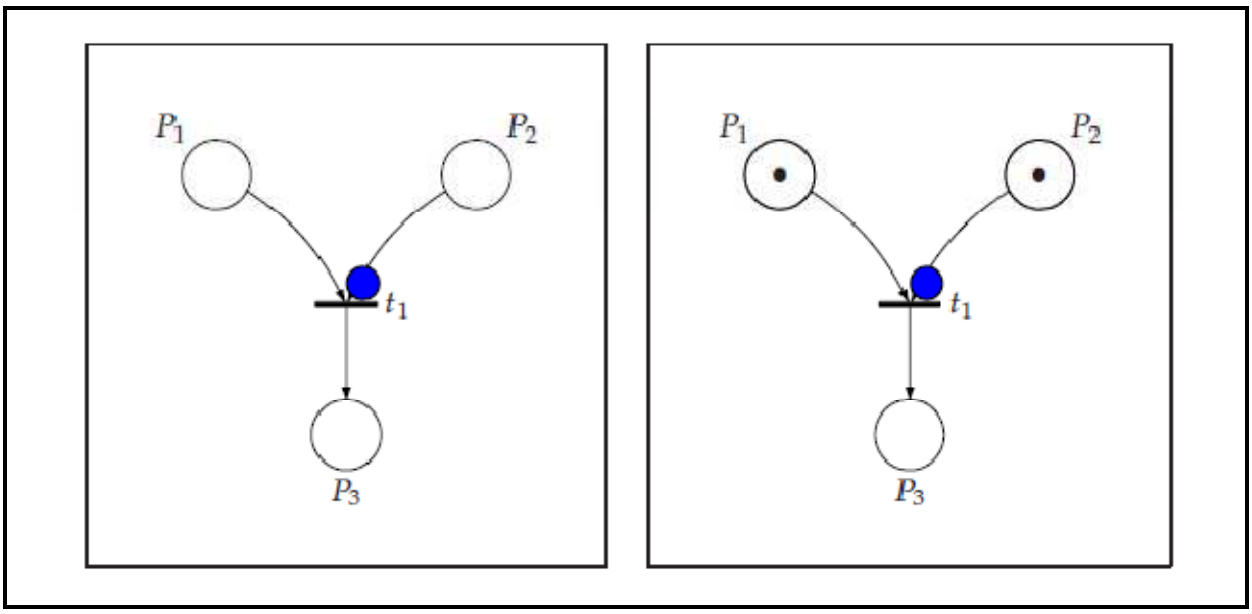

Figure 18: Deux réseaux de Petri à arc inhibiteur non franchissables

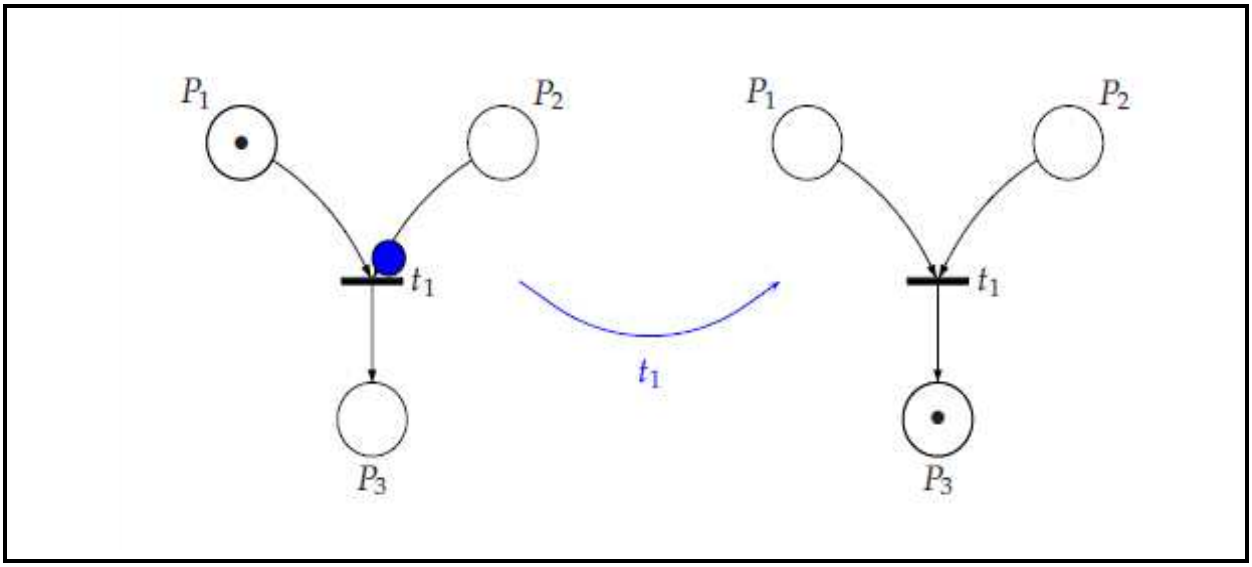

Figure 19: Un réseau d Petri à arc inhibiteur franchissable avant et après franchissement

3.5 Exemples de modélisation par des réseaux de Petri

3.5.1 Machine à états finis

Les machines à état finis peuvent être représentées par un réseau de Petri. Comme exemple de machine à états finis, on considère une machine de vente quelle accepte des pièces de monnaie et vend des bonbons de 15D et de 20D. Le diagramme d'état de la machine

à état finis peut être représenté par le réseau de Petri illustré à la figure ci-dessous, où les 5 états sont représentés par les 5 places étiquetées 0D, 5D, 10D, 15D, et 20D. Et les transformations d'un état à un autre sont illustrés par des étiquètes tel que « déposer 5D ». L'état initiale est indiqué par la mise d'un jeton dans la place p_1, avec l'étiquète 0D dans cet exemple. Notons que chaque transition dans ce réseau a un seul arc d'entrée et un seul arc de sortie. Les réseaux de Petri avec cette propriété sont connus comme des machines à états. Toute machine à états fini ou (son diagramme d'état) peuvent être modélisé avec une machine à états.

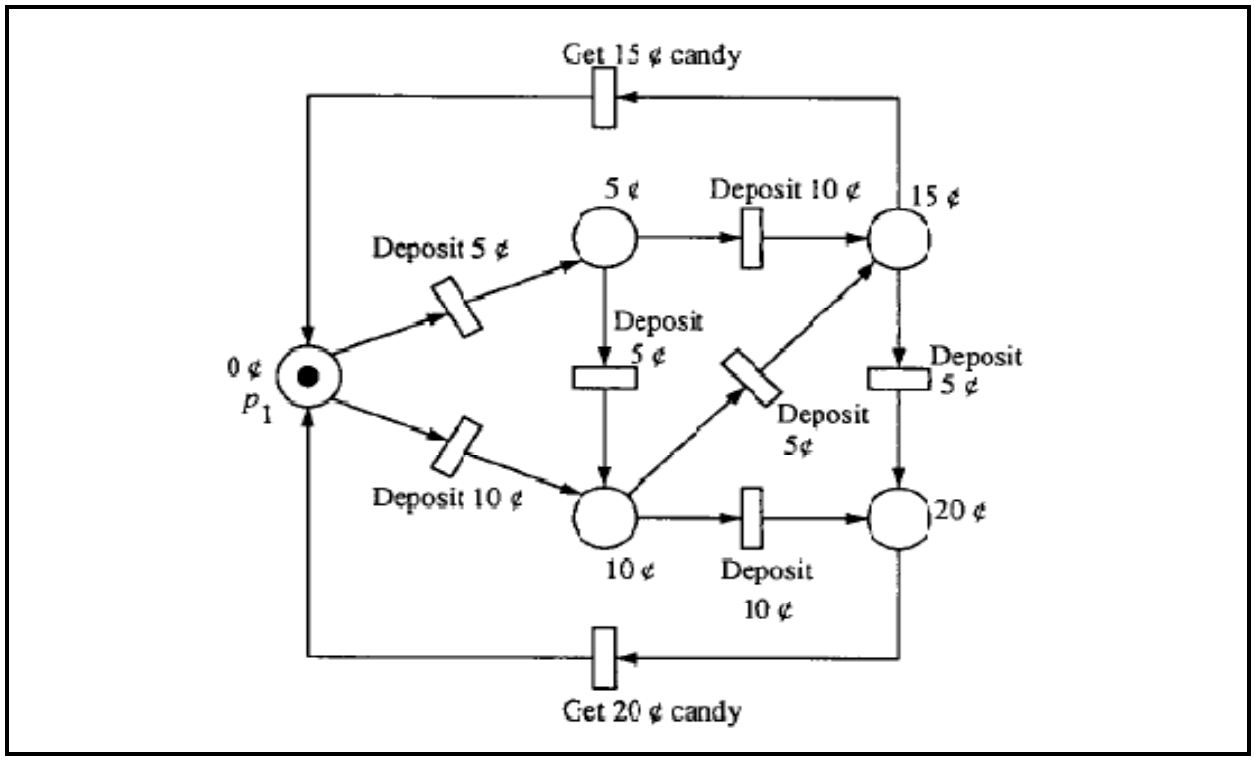

Figure 20: Modélisation d'une machine à états finis

3.5.2 Activités parallèles

Les activités parallèles ou la concurrence peuvent être facilement exprimé par des réseaux de Petri. Par exemple, dans le réseau de Petri montré dans la figure 21. Les activités parallèles ou concurrents représentés par les transitions t2 et t3 commence dans le franchissement de la transition t_1 et termine avec le franchissement de la transition t_4. En générale, deux transitions sont dites concurrentes s'ils sont causalement indépendantes, i.e., une des transitions peut être franchie avant ou après ou en parallèle avec l'autre, comme était le cas de t_2 et t_3 dans la figue 6.

Notons que chaque place dans le réseau de la figure 13 a exactement un arc d'entrée et exactement un arc de sortie. Les réseaux de Petri qui ont cette propriété sont connus comme des graphes marqués. Les graphes marqués permettent la représentation de concurrence mais pas des décisions.

Deux évènements e_1 et e_2 sont en conflit si e_1 ou e_2 peut avoir lieu et ne pas les deux, et ils sont concurrents si tout les deux peuvent avoir lieu dans n'import quel ordre sans conflits.

3.5.3 Calcule de flux de données

Les réseaux de Petri peuvent être utilisés pour représenter pas seulement le flux de contrôle mais aussi le flux de données. Le réseau illustrer dans la figure 21 est une représentation par réseau de Petri d'un calcule de flux de données. Dans un calculateur de flux de données les instructions sont permises d'être exécutés par l'arrivée de ses opérandes, et peuvent être exécutés en concurrence. Dans une représentation par réseau de Petri de calcule de flux de données, les jetons représentent les valeurs des données courants comme étant une disponibilité des données. Dans Le réseau de Petri de la figure 21, les instructions représentés par des transitions t1 et t2 peuvent être exécutés d'une manière concurrente et déposent les donnés résultants (a+b) ou (a-b) dans la place de sortie respective.

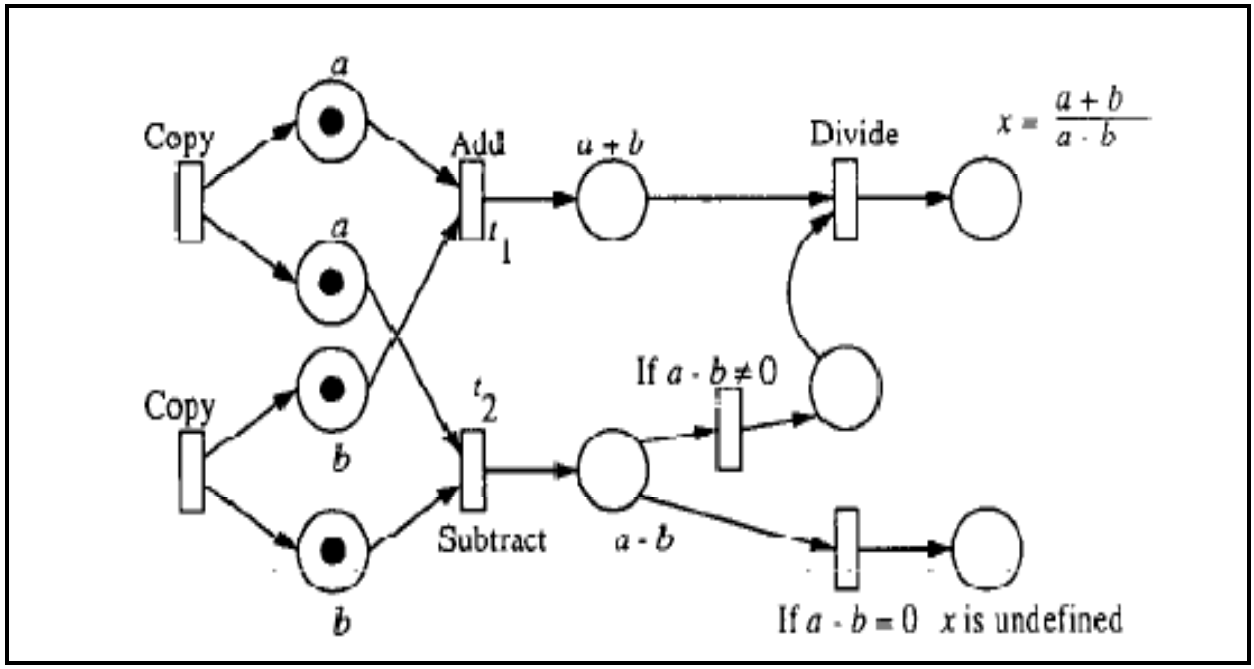

Figure 21: Calcule de flux de données

3.6 Conclusion

Les réseaux de Petri ont été définis comme un outil de modélisation des systèmes présentant des activités concurrents et synchrones. Le facteur majeur qui peut affecter son acceptation concerne la puissance de modélisation et de décision de modèle. Aussi les réseaux de Petri ne sont pas le seul moyen de modélisation des systèmes concurrent et synchrones, ils existent d'autres modèles. En plus ils ont une certaine clarté qui permet une représentation naturelle de plusieurs systèmes. Pour cela ils acquièrent une acceptation de plus en plus dans le domaine de modélisation, et leur utilisation est en augmentation continue.

Dans ce chapitre on a donné une brève présentation des réseaux de Petri, qui constituent pour notre travail l'approche de modélisation de l'opération de génération de configurations. Et à travers cette vue on espère donner la motivation de choisir cet outil de modélisation.

Chapitre 4 : Adaptabilité dans les systèmes multi agents

4.1 Introduction

Les logiciels d'aujourd'hui répondent à des besoins de plus en plus spécifiques et complexes. A cause de cette complexité, il n'est pas rare qu'un logiciel ne corresponde plus parfaitement à ses spécifications après un certain temps de son implémentation, Il peut aussi arriver que les spécifications initiales d'un logiciel ne soient plus pertinentes après un certain temps d'utilisation, ou même pendant sa réalisation.

C'est ainsi que s'impose le besoin d'adaptation des logiciels. Le verbe « adapter » est utilisé comme « se conformer à des conditions nouvelles ou différentes ». Si le changement des conditions a été prévu alors une étape de reconfiguration du logiciel par un ensemble de configurations déterminées à l'avance peut suffire. Par contre, si le changement des conditions n'a pas été prévu, alors il faut adapter le logiciel en modifiant une partie, voire la totalité.

Plusieurs travaux ont été mis dans ce chemin, et particulièrement dans le domaine des systèmes multi agents, puisque ceux-ci sont par nature exposés à des changements fréquents de contexte, dues aux déplacements de site à un autre, tel est le cas des agents mobiles, qui, leur raison d'être est de déplacer avec sont code et ces données pour accomplir des tâches au profit de son propriétaire.

L'adaptation des systèmes multi agents est abordé à travers, deux points de vue, l'un attaque le coté d'adaptation de l'organisation en totalité, par le phénomène d'émergence qui consiste à avoir un comportement global complexe et intelligent à partir de comportements locales simples, donc l'adaptation concerne aussi bien les relation entre les agents plus que la structure des agents eux-mêmes, bien que un deuxième point de vue va tenter d'adapter l'agent lui même en lui dotant de nouvelles compétences, voir de changer sa structure complètement pour le rendre capable de confronter de nouvelles situations.

Ce chapitre, qui constitue un état de l'art de notre travail, à travers lequel on va éclairer des travaux, dans les deux onglets de l'adaptabilité des agents. Pour chacun des travaux présentés on a donné l'approche proposé et les étapes suivie pour le développement.

4.2 Étapes de l'adaptation[1]

Une opération d'adaptation peut être vue comme la succession de trois étapes suivantes :

[1] Thomas LEDOUX, Etat de l'art de l'adaptabilité.

4.2.1 Déclenchement

L'étape de déclenchement consiste à détecter et à notifier un changement. Ce déclenchement peut être effectué par différents acteurs (utilisateur, sonde logicielle) et à différents moments (déploiement, exécution).

4.2.2 Décision

La décision consiste à déterminer les modifications qui doivent être effectuées en réponse aux changements détectés. Ces choix peuvent être effectués par différents acteurs, externes (programmeur, utilisateur) ou interne dans l'application elle même (mécanisme d'adaptation intégré par exemple), et concerne différents sujets à adapter (un composant ou un ensemble de composants, les relations entre les composants, ou même toute la configuration de composants) et suivre différentes règles ou stratégies d'adaptation.

4.2.3 Réalisation

La réalisation de l'adaptation concerne tous les moyens qui vont être mises en œuvre pour appliquer la décision prise. La réalisation peut être effectuée par différents acteurs (programmeur, utilisateur) et selon différents mécanismes. La réalisation s'opère sur les sujets à adapter qui ont été définis dans l'étape de décision.

4.3 Dimension de l'adaptation

La première chose à définir est la dimension de l'adaptation. C'est-à-dire que ce qu'on va adapter ? Pour pouvoir y répondre, nous allons tout d'abord supposer qu'un système logiciel est une configuration composée d'un ensemble de composants reliées entre eux. La cible de l'adaptation peut être soit un composant, soit une interface entre deux composants, soit la configuration globale (c-à-d la composition de plusieurs composants).

4.3.1 Le composant

Sa granularité peut être plus ou moins fine et sa durée de vie, dans le cycle de vie du système, plus ou moins longue. L'adaptation d'un composant consiste à appliquer un processus de modification sur ce dernier. Cela peut aller d'un simple paramétrage à une transformation complète.

4.3.2 Interface (ou liaison)

Une interface est une image externe d'un composant. La modification de l'interface entre deux composants, par changement de type suppression ou autre façon de modification, sont des manières potentielles de l'adaptation.

4.3.3 Configuration

La cible de l'adaptation peut être toute la configuration. Il s'agit alors de réaliser un nouvel assemblage de composants. Il peut être mis en œuvre de différentes manières : soit par retrait, remplacement, reconfiguration de certaines composants et/ou interfaces, soit par ajout de nouvelles.

4.4 Niveau de l'adaptation[1]

On a déjà vu les trois étapes de l'adaptation, mais dans chacune de ces étapes, différents niveaux peuvent être impliqués. Ainsi, le niveau déclenchant l'adaptation peut être différent de celui qui va décider de la démarche à suivre. Nous distinguons trois types de niveaux :

4.4.1 Programmeur

Lorsque l'adaptation est prévue à l'avance, le programmeur peut décider de figer une ou plusieurs des étapes d'adaptation. Les choix sur les sujets à adapter, les règles d'adaptation et les mécanismes peuvent être fixés à l'implémentation. Éventuellement, les règles d'adaptation peuvent être fusionnées avec le sujet de l'adaptation. L'adaptation est automatique, mais ses étapes sont fusionnées et donc non modifiables.

4.4.2 Administrateur

Si l'on désire ne pas figer une des étapes de l'adaptation à l'implémentation, il est possible d'effectuer les choix au lancement du système ou pendant son exécution. Ces choix peuvent être faits par un administrateur - nom générique représentant un déployeur ou un utilisateur - qui peut alors décider du déclenchement de l'adaptation, des sujets à adapter, de la stratégie ou des mécanismes. L'adaptation (ses étapes) est réifiée, mais elle n'est pas automatique.

[1] Thomas LEDOUX, Etat de l'art de l'adaptabilité.

4.4.3 Logiciel

Si l'on se place dans la même hypothèse que ci-dessus (c.-à-d. étapes non figées à l'implémentation et choix retardés), les choix d'adaptation peuvent également être pris par une entité logicielle. Par exemple, le déclenchement de l'adaptation peut s'opérer grâce à des sondes pour l'observation de ressources et des notifications de changement de l'environnement. La décision d'adaptation peut, par exemple, être assurée par un système expert. La réalisation de l'adaptation peut être laissée à un intergiciel spécialisé. D'un point de vue systémique, on considère que ce dernier cas propose la notion d'auto-adaptation puisque le logiciel peut modifier son propre comportement en fonction de son contexte d'exécution. L'adaptation est automatique.

4.5 Moments de l'adaptation

Les moments de l'adaptation dépendent des choix faits précédemment, à savoir la dimension et le niveau de l'adaptation (quoi adapter et qui adapter). Il existe trois temps pour l'adaptation: avant l'exécution du système, au lancement du système et pendant l'exécution du système.

4.5.1 Avant l'exécution du système

Une adaptation statique peut être réalisée à la compilation, à l'édition des liens en fonction de la plate-forme cible. Pour autoriser des adaptations ultérieures, il est nécessaire en phase de conception et développement de prévoir les possibilités d'adaptabilité du système et de faire des choix en conséquence. La conception modulaire et à base de composants font partie des pré-requis pour l'adaptabilité. De plus, si l'on s'intéresse à l'adaptabilité à l'exécution, il faut choisir une approche permettant aux composants et/ou à leurs interfaces d'exister explicitement à l'exécution, et pas seulement au moment du codage.

4.5.2 Au lancement du système

Il s'agit d'une adaptation statique juste avant l'exécution du programme. Au moment du déploiement, l'administrateur connaît l'état de l'environnement d'exécution et sa topologie, et cela qui fait la différence avec l'adaptation statique précédente. À cet instant ponctuel, il est donc amené à faire des choix pertinents d'adaptation.

4.5.3 Pendant l'exécution du système

On parle dans ce cas d'adaptation ou de reconfiguration dynamique. Dans certains cas, il est possible de modifier à l'exécution le sujet d'adaptation. Le fait qu'il faille désactivé (arrêter) le système pour faire une mise à jour nous ramène au cas précédent avec une configuration statique. Dans ce type d'adaptation une opération de transfert d'état peut être nécessaire pour continuer l'exécution depuis l'étape où on est stoppé pour l'adaptation.

4.6 Adaptabilité dans les systèmes multi agents

Dans cette section on va revenir sur le point 4.3 concernant la dimension de l'adaptation et étudier l'adaptation dans les systèmes multi agents en distinguant deux dimensions connus à savoir l'adaptabilité au niveau de l'agent qui va utiliser les dimensions composant et configuration, et l'adaptation au niveau de l'organisation qui concerne aussi bien la dimension interface.

4.6.1 L'adaptabilité au niveau de l'agent

Dans cette partie on va présenter des travaux qui ont choisi de confronter les changements de l'environnement en modifiant l'agent lui-même par un ajout ou retrait de certaines fonctionnalités (représentés par des composants), ou même la remise en cause de son architecture (la configuration).

4.6.1.1 MADCAR- AGENT[1]

Dans ce modèle MADCAR-AGENT (Model for Automatic and Dynamic component Assembly Reconfiguration) le problème de l'adaptation d'applications ubiquitaires[2] est abordé. Et il est proposé de fournir une infrastructure à composants pour simplifier *la construction d'agents par assemblage de composants* et *l'adaptation de ces agents de manière autonome*. En plus de la préservation de l'autonomie des agents lors des adaptations, différents problèmes sont abordés:

- *Abstraction* : se détacher des spécificités de chaque modèle de composants;

[1] Guillaume Grondin, MaDcAr-Agent : un modèle d'agents auto-adaptables à base de composants.
[2] Applications ubiquitaires ou informatique ubiquitaire est une notion qui défende l'omniprésence et l'invisibilité de l'informatique dans tous les aspects de la vie quotidienne (de la même source précédente).

- *Dynamicité* : le moment de l'adaptation est lors de l'exécution (*i.e.* réaliser les adaptations dynamiquement sans stopper complètement l'exécution de l'agent);

- *Ouverture* : pouvoir configurer le processus d'adaptation et permettre aussi bien les adaptations légères (comme une modification partielle du comportement) que les adaptations profondes (comme une modification totale de la structure) ;

- *Réutilisation* : favoriser le principe de réutilisation en limitant le couplage entre la spécification de l'agent, la spécification des adaptations et les composants utilisés, cela est facilité grâce à une conception modulaire et à base de composants;

- *Sensibilité au contexte* : pour déclencher et décider des adaptations en fonction de l'évolution du contexte (CPU, mémoire, connectivité, etc.) par des sondes logicielles observant l'état de l'environnement.

Le modèle s'articule autour de deux axes : la programmation par composants et les systèmes multi-agents.

En ce qui concerne la programmation par composants, l'objectif est de fournir une solution générale au problème de l'assemblage automatique et dynamique de composants. MADCAR, est un modèle de moteur d'assemblage de composants. Ce moteur permet la conception d'applications auto-adaptables, le moteur étant considéré comme partie intégrante de l'application afin de préserver son autonomie lors de choix des adaptations.

En ce qui concerne le deuxième axe relatif aux systèmes multi-agents, on trouve MADCAR-Agent, un modèle d'agents auto-adaptables. Comme son nom l'indique, il s'agit d'intégrer dans chaque agent le moteur MADCAR en charge des adaptations. Cependant le concept d'agent étant plus riche qu'une application classique, les spécificités inhérentes au domaine des SMAs notamment en ce qui concerne l'interaction et la coopération entre agents sont prises en compte. Ainsi, la proposition consiste également en un protocole d'échange de composants entre agents afin de permettre à un agent d'obtenir des composants supplémentaires de la part d'autres agents, en fonction de ses besoins. Les échanges sont configurables à travers une politique de gestion du contenu de l'agent.

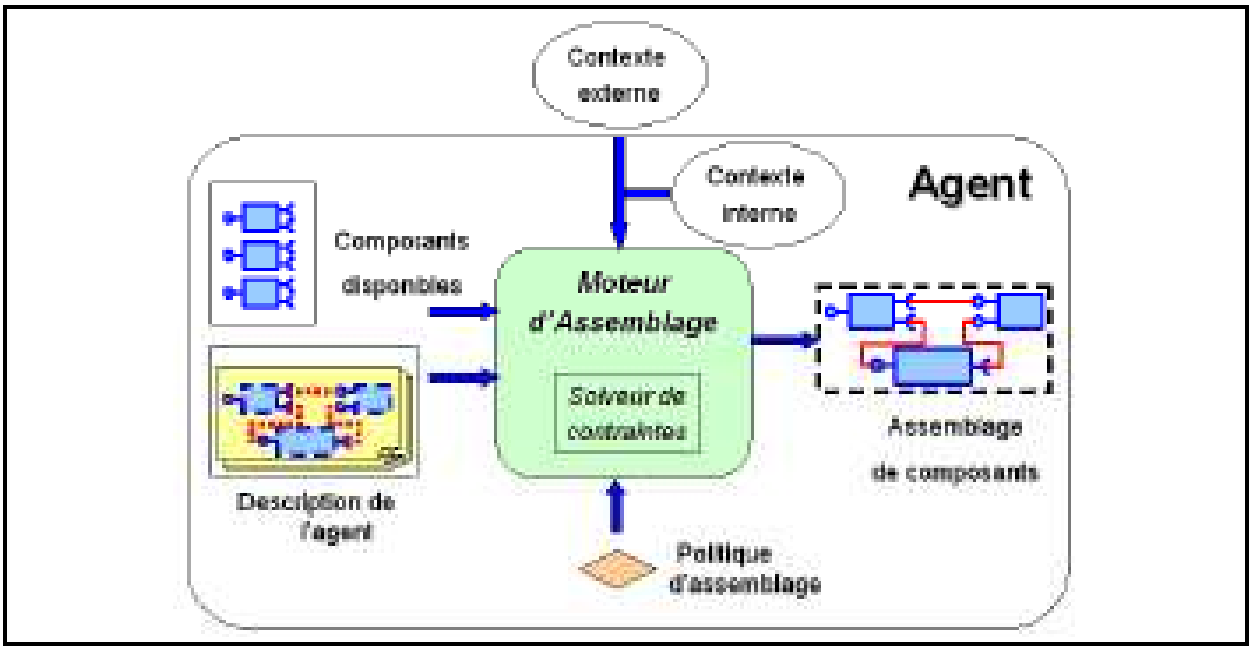

Figure 22: Structure d'un agent MaDcAr

Structure d'un agent avec MADCAR-Agent

Comme illustrer sur figure ci-dessus, dans un agent MADCAR-Agent, on trouve le moteur d'assemblage au milieu et les autres différents composants comme des entrées/sorties pour ce moteur. Il utilise pour la construction du comportement de l'agent quatre entrées, à savoir :

- Un ensemble de composants à assembler ;
- Une description de l'agent : qui représente la spécification de l'assemblage ;
- Une politique d'assemblage : elle dirige les décisions d'assemblage ;
- Un contexte : qui en fonction de ses changements est déclenchée l'opération d'adaptation, on peut distinguer deux parties de contexte :
  - *Interne* : Représente l'état de l'agent,
  - *Externe* : Représente l'état du monde dans lequel évolue l'agent.

Description d'un agent avec MADCAR-Agent

La description d'un agent avec MADCAR-Agent est donnée par un ensemble d'assemblages valides à l'aide d'un ensemble de configurations alternatives. Une

configuration représente un assemblage de rôles. Un **rôle** est une description abstraite de composants, constituée d'un ensemble de *contrats*. Ces contrats doivent au moins spécifier :

1) Un ensemble d'interfaces (fournies ou requises) symbolisant ses éventuelles interactions avec les autres rôles,

2) Un ensemble d'attributs dont les valeurs permettent d'initialiser les composants, et

3) Deux multiplicités. Les *multiplicités* d'un rôle représentent le nombre minimal (min où min >= 0) et le nombre maximal (max où max >= min) de composants qui peuvent remplir ce rôle simultanément.

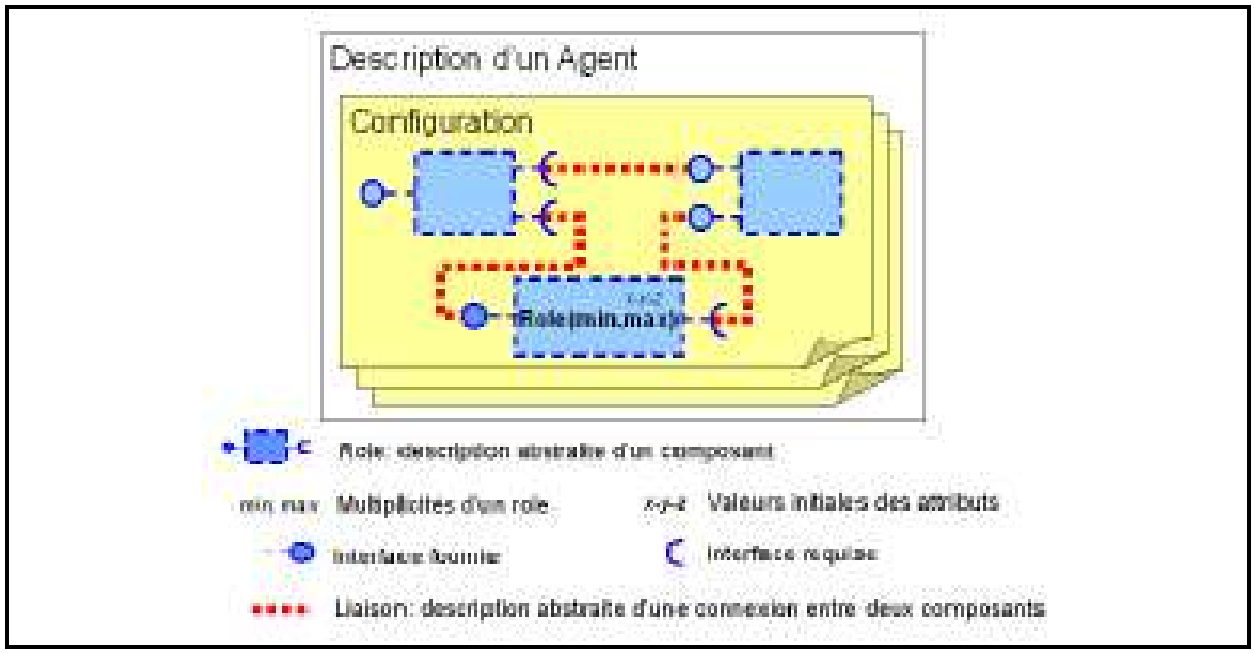

Figure 23: Description d'un agent aces MaDcAr

Processus de re-assemblage de MADCAR

MADCAR est, d'après son auteur, adapté aussi bien à créer automatiquement un assemblage à partie de composants déconnectés, que pour adapter dynamiquement un assemblage existant, ce processus de re-assemblage est décomposé en cinq étapes successives :

1) *Déclenchement :* le ré-assemblage est déclenché suite à la détection d'un événement important ;

2) *Identification des configurations éligibles* : une configuration est dite éligible lorsque tous les composants nécessaires à son assemblage sont présents ;

3) *Sélection d'une configuration* : une configuration est sélectionner pour être mis en marche ;

4) *Sélection des composants* : Selon la configuration choisie et la politique d'assemblage, le moteur d'assemblage choisi un ensemble de composants à intégrer dans la nouvelle configuration ;

5) *Re-assemblage* : Suivant la configuration adoptée le moteur fait l'assemblage des composants sélectionnes.

Opération d'adaptation dans MADCAR

L'opération d'adaptation dans MADCAR peut être exprimée en deux parties :

a. *Détection de changement* : il s'agit d'utiliser des sondes logiciel pour surveiller périodiquement l'état de l'environnement et une fois détecter un évènement qui nécessite une opération d'adaptation ces sondes jouent deux rôles, en premier ils permettent le déclenchement des adaptations et en deuxième ils servent à recueillir des informations internes ou externes à l'agent, qui sont utiles pour les décisions d'adaptation.

b. *Décision* : MADCAR prend les d'adaptation en se basant sur le contexte interne et externe de l'agent, et pour cela il pose plusieurs choix :

   a. *Lorsque plusieurs configurations satisfont la politique d'assemblage de l'agent*, l'une d'entre elles est sélectionnée arbitrairement (à moins que la configuration actuelle ne satisfait la politique d'assemblage) ;

   b. *Lorsque aucune configuration ne satisfait la politique d'assemblage de l'agent*, une configuration éligible est sélectionnée automatiquement et arbitrairement (à moins que la configuration courante ne soit éligible) ;

   c. Lorsque aucune configuration n'est éligible l'agent se met en attente de nouveaux composants, et peut être de nouvelles configurations ;

4.6.1.2 MAST (Multi-Agent System Toolkit).

Le modèle MAST, développé par l'équipe SMA de l'Ecole des Mines de Saint-Etienne, est un environnement de développement et de déploiement d'applications multi-agents. Dans MAST, une application multi-agents est construite comme un assemblage de composants. La conception, reprend la décomposition VOYELLES (également noté AEIOU),

considère qu'un SMA est composé des quatre dimensions Agent, Environnement, Interaction Organisation et Utilisateur qui est ajouté comme une autre dimension. La conception s'applique sur deux niveaux et le modèle de composant proposé tient compte de ces niveaux en distinguant deux types de composants. Au *niveau système*, un SMA est composé d'agents (vus comme des composants) et de certaines entités fonctionnelles nommées composants orientés système (COS). Au *niveau agent*, un agent est également conçu comme un assemblage de composants au sens plus classique du terme, dits composants orientés agent (COA).

Composants orientés système

Un composant orienté système concerne une des dimensions voyelles, et on peut même trouver plusieurs composants orientés système COS dans une même dimension voyelle. Par exemple :

- Agent : tous les agents du système ;
- Environnement : environnement situé ;
- Interaction : intergiciel de communication ;
- Organisation : support organisationnel ;
- Utilisateur : interface graphique de l'application,…

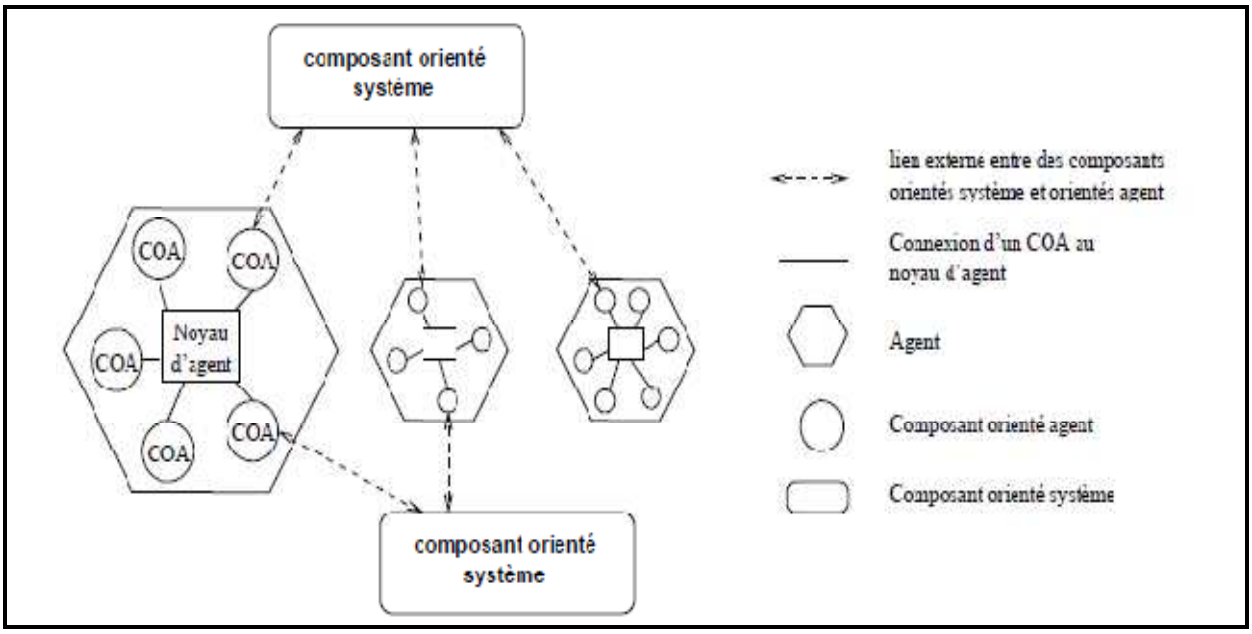

Figure 24: Composants COS et composants COA

Composants orientés agent

Un agent comme le système est aussi formé par un assemblage de composants, ces composants sont appelés des composants orientés agent COA. La base d'un agent est un noyau auteur duquel sont connectés tous les composants orientés agent, et c'est à lui de gérer les interactions entre ces composants.

Construction d'un agent MAST

Pour construire un agent MAST, il suffit de lui ajouter un composant « noyau » et un ensemble de composants fonctionnels (les COAs). MAST permet de connecter automatiquement et de manière flexible les composants qui forment un agent. En effet, les événements émis par les différents composants de l'agent sont capturés par le *composant noyau*. Ce dernier se base sur la description sémantique de chaque composant (*i.e.* leur rôle) pour transmettre chaque événement reçu aux composants capables de le traiter. L'évènement est transmis aux COAs -successivement selon l'attribut « priorité » qui les caractérise- jusqu'à ce que l'évènement soit consommé par l'un des COAs. Pour définir un assemblage de composants, le concepteur doit définir la priorité (un réel) de chaque interface des composants présents, cette priorité va déterminer l'ordre de passer un évènement vers les composants, ce qui influence directement sur le comportement de l'agent.

Ces connexions implicites sont intéressantes pour la flexibilité qu'elles procurent à l'assemblage de l'agent, puisqu'elles sont automatiquement déduites en fonction des interfaces des composants. Ainsi, l'adaptation des agents qui consiste à lui ajouter ou supprimer des composants n'est pas réalisée automatiquement, elle nécessite l'ajout ou le retrait manuel d'un ou de plusieurs composants.

Interface d'un agent MAST

L'interface d'un agent comporte les champs suivants :

- `name`: un nom symbolique pour le composants ;
- `dimension` : à lesquelles des dimensions Voyelles appartient le composant ;
- `priority` : c'est la priorité par défaut de composants représentée par une valeur numérique, cette valeur est utilisée lors de l'envoi d'un évènement ;
- `provided roles` : c'est un ensemble de noms de rôles joués par le composant ;

- `required roles` : l'ensemble des rôles requis par le composant ;
- `sent events` : les évènements que le composant est capable d'émettre ;
- `handled events` : les évènements que le composant peut traiter.

4.6.1.3 Magique

MAGIQUE est un Framework multi-agents permettant de construire des systèmes multi agents où les agents sont organisés hiérarchiquement. Les systèmes multi agents produits s'apparentent à un système d'objets concurrents répartis dans lequel l'organisation hiérarchique des agents permet à chacun d'invoquer des fonctionnalités quelle que soit leur localisation grâce au principe de délégation des tâches. En effet, pour chaque invocation, l'agent essaie de traiter la tâche concernée s'il possède la compétence nécessaire. Dans le cas contraire, il essaie de faire traiter la tâche par l'un de ses voisins directs ou de ses subalternes, ou alors il délègue la tâche à son supérieur hiérarchique. Ce mécanisme de délégation permet aux agents de faire appel à des compétences sans préciser de destinataires.

Un agent est vu comme une coquille contenant des *compétences*. Chaque compétence est un «ensemble cohérent de fonctionnalités» permettant de réaliser une tâche précise qui est regroupée dans un composant logiciel. Donc, les compétences peuvent être développées indépendamment des agents qui les utiliseront. Pour invoquer une opération offerte par une compétence interne ou externe à l'agent, il faut utiliser la primitive `perform("nomOperation", arguments)`sans besoin de nommer le destinataire, ce qui peut poser problème quand des opérations sémantiquement différentes ont la même signature dans une hiérarchie d'agents. Pour créer un agent, il faut ajouter un ensemble de compétences à sa «coquille vide», en particulier une compétence pour communiquer et une compétence pour pouvoir acquérir d'autres compétences. Enfin, si l'on veut que l'agent soit proactif, alors il doit posséder une *compétence d'action* (qui implémente la méthode `action()`) qui représente le cœur du comportement de l'agent.

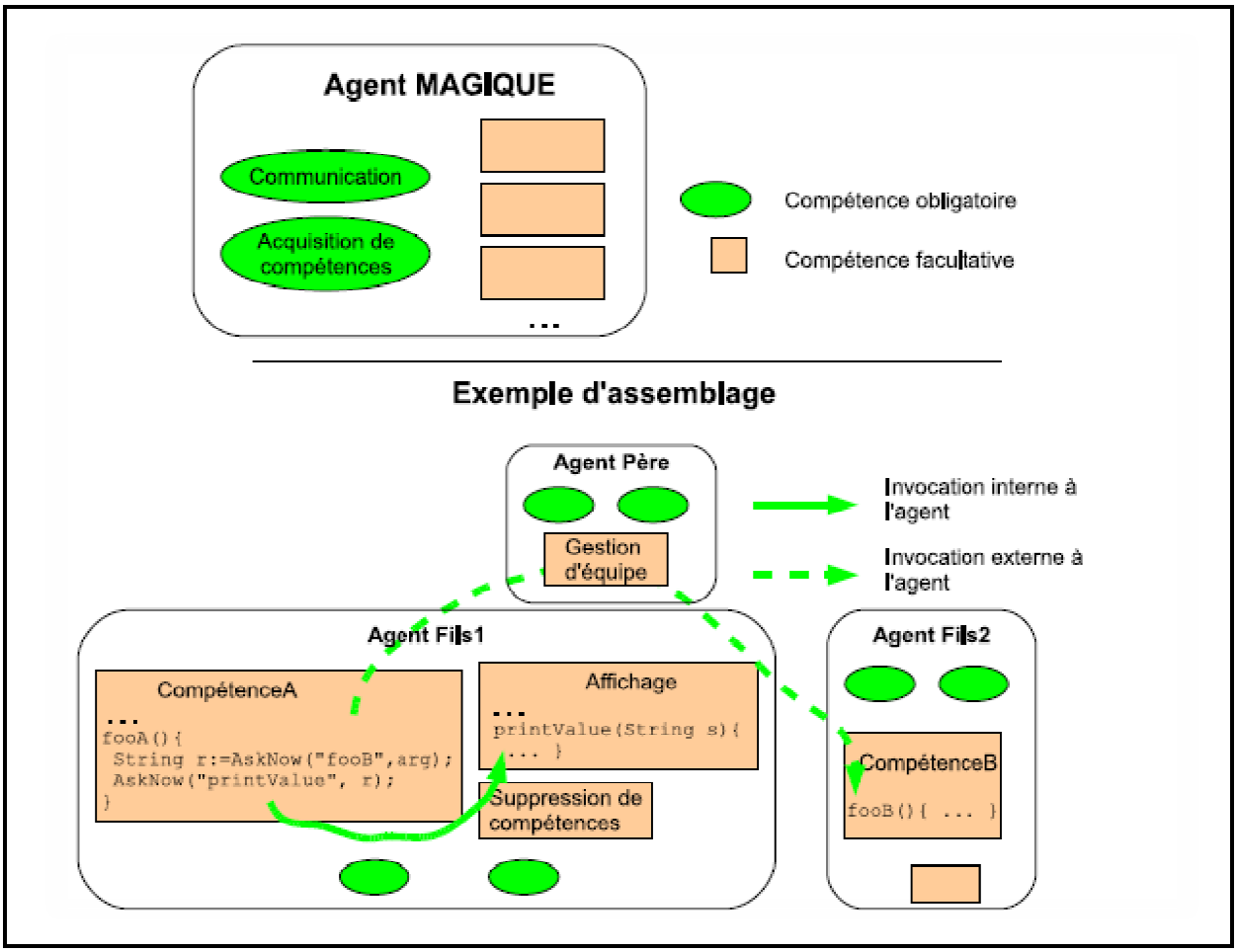

Figure 25: Agent social avec MAGIQUE

Un agent MAGIQUE peut acquérir (ou oublier) et invoquer dynamiquement de nouvelles compétences. Par exemple, si un agent A_1 utilise beaucoup une compétence d'un agent B_2 alors il peut demander à B_2 de lui envoyer cette compétence plutôt que de continuer à l'utiliser «à distance». Cet échange est réalisé sans difficulté grâce aux compétences de communication et d'acquisition de compétences dont disposent les agents.

Dans le modèle MAGIQUE, un agent est principalement vu comme un fournisseur de compétences pour les autres agents du système multi agents. Ainsi, l'adaptation d'un agent par échange de comp étences tient plus du fait d'une optimisation du déploiement des compétences au sein du système multi agents que des besoins propres à cet agent. C'est pourquoi, ce modèle n'offre pas de mécanisme pour spécifier du comportement d'adaptation de l'agent de manière découplée par rapport au comportement applicatif de l'agent : tout doit être codé dans la compétence d'action. En outre, si une compétence n'est disponible chez aucun agent, alors ce sont tous les agents qui utilisent cette compétence qui sont paralysés. Pour éviter ce genre de problème, le concepteur du système doit essayer d'assurer que chaque compétence soit répliquée chez plusieurs agents.

4.6.1.4 MALEVA

MALEVA est un modèle de conception de systèmes multi-agents. Ce modèle est utilisé pour construire des agents logiciels dont le comportement et la structure peuvent être adaptés. Il prévoit deux sortes d'interaction entre composants : un *flot de données* et un *flot de contrôle*. Le principal avantage d'avoir un flot de contrôle explicite dans un assemblage de composants est de faciliter l'identification de problème de concurrence. MALEVA a notamment été utilisé dans les domaines des sociétés artificielles et de la simulation multi-agents.

Un concepteur d'agents dispose d'une bibliothèque de composants représentant des comportements élémentaires. Il peut assembler plusieurs composants existants dans un composant composite de manière à constituer des comportements complexes réutilisables dans différents contextes ou différentes familles d'agents. Les auteurs utilisent aussi une forme simplifiée de patron de conception permettant d'instancier un comportement complexe abstrait et spécialisable : un précâblage entre des composants fixes et des composants à insérer. Concevoir un agent revient à composer, de manière structurelle et fonctionnelle, différents composants comportementaux au sein de l'agent. Par exemple, le comportement «Proie» est défini en connectant trois composants : la fuite (le composant `FuirAgent`), le mouvement aléatoire (le composant `MvtAléatoire`), et un composant de contrôle appelé Switch qui permet de réifier la conditionnelle perception/non-perception. La gestion du contrôle revient à définir et ajouter des composants spécifiques qui ont à charge de traiter les compositions associées aux données et celles associées aux contrôles.

Les adaptations peuvent être déclenchées par l'agent après qu'un de ses composants de comportement émette une requête particulière. Cette requête est transmise à un composant spécifique appelé « méta composant », qui se comporte comme un fournisseur de « profils comportementaux » préalablement définis par le concepteur de l'agent. Ces profils prédéfinis décrivent des assemblages spécifiques de composants. Donc, ils améliorent le contrôle du concepteur de l'agent sur les assemblages à générer en interdisant les assemblages non prédéfinis.

4.6.1.5 DIMA[1]

DIMA (Développement et Implémentation de Systèmes Multi-Agents) est un environnement de développement de systèmes multi-agents dont le développement a débuté en 1993. La première version de DIMA a été implémentée en Smalltalk-80 et a été ensuite portée en JAVA.

[1] Zahia Guessoum. Thomas Meurisse et Jean-Pierre Briot. Construction modulaire d'agents et de systèmes multi-agents adaptatifs en DIMA.

DIMA est basée sur une architecture modulaire et une bibliothèque de composants. Elle est fondée sur l'extension des facilités de modélisation et d'implémentation de langages à objets. Les auteurs de DIMA veulent rendre un environnement à objets plus approprié aux problèmes de systèmes multi-agents en lui incorporant des représentations spécifiques et des structures de contrôle.

Le modèle d'architecture d'agents DIMA propose de décomposer chaque agent en différents composants dont le but est d'intégrer des paradigmes existants notamment des paradigmes d'intelligence artificielle. Un agent peut ainsi avoir un ou plusieurs composants qui peuvent être réactifs ou cognitifs. Ce modèle permet de dépasser la dichotomie classique réactif/cognitif en permettant de définir des agents hétérogènes au sein d'une même application. Chaque agent est composé d'un ou de plusieurs composants.

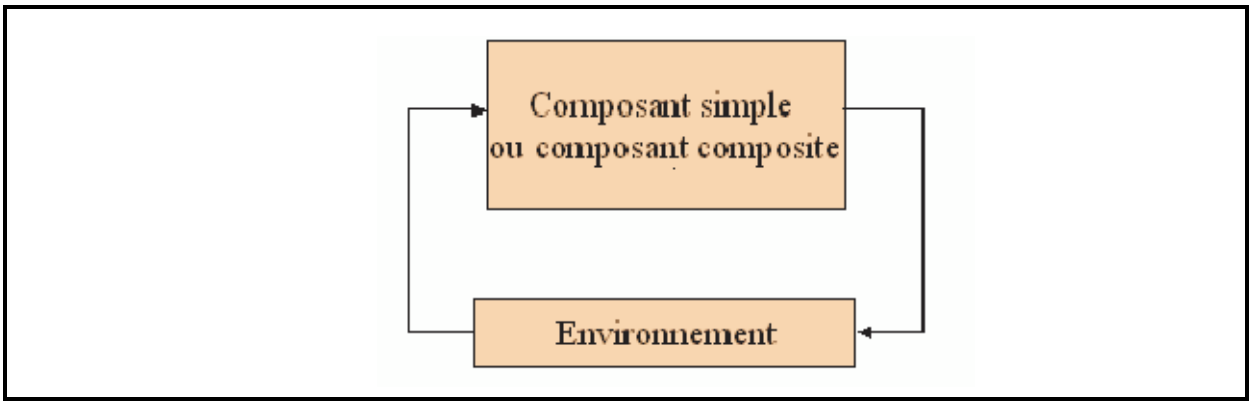

Figure 26: Architecture d'un agent DIMA

La modularité offre ainsi plusieurs avantages à cette architecture :

- Possibilité de définir des agents à granularité variable.
- Possibilité de définir des agents à structure adaptative. Chaque agent peut dynamiquement changer ces composants ainsi que les relations entre ces différents composants.
- Possibilités d'intégrer différents modèles d'agents.
- Possibilité de définir des bibliothèques de composants réutilisables.
- Etc.

Le modèle d'architecture d'agents DIMA est donc fondé sur la conclusion suivante : les travaux sur les architectures d'agents ont engendré plusieurs résultats intéressants. Il est

très difficile de comparer ces architectures dans le but de trouver la meilleure. Chacune est bien appropriée à une catégorie d'agents. Une bonne plate-forme intègre les différentes architectures d'agents existantes ainsi que d'éventuelles nouvelles architectures. Ainsi, le modèle d'architecture d'agents proposé par DIMA peut être vu comme un modèle 'ouvert', une proposition d'agent minimal est faite permettant, par raffinements successifs, d'ajouter des fonctionnalités fournies par les différentes bibliothèques de la plate-forme. La force de DIMA réside donc à la fois dans sa proposition de modèle d'architecture d'agents modulaire, mais également dans les différentes bibliothèques disponibles. Chaque agent est un composant simple ou composant composite qui gère l'interaction de l'agent avec son environnement et qui représente son comportement interne. L'environnement regroupe tous les autres agents ainsi que les entités qui ne sont pas des agents.

Différents paradigmes (Automate, règles de production,...) sont réutilisés pour définir de nouvelles classes de composants proactifs. Par exemple, le comportement d'un composant proactif peut être modélisé par un ATN (Augmented Transition Network) dont les états correspondent aux différents états d'un composant proactif. A chaque état est associé une ou plusieurs transitions. Une transition est définie par une condition et une action. L'action de transition est l'activation d'une compétence.

La Figure suivante donne des exemples de composants proactifs. Dans la classe `ATNBasedProactiveComponent`, la méthode `step)` permet d'activer une transition dont la condition est vérifiée.

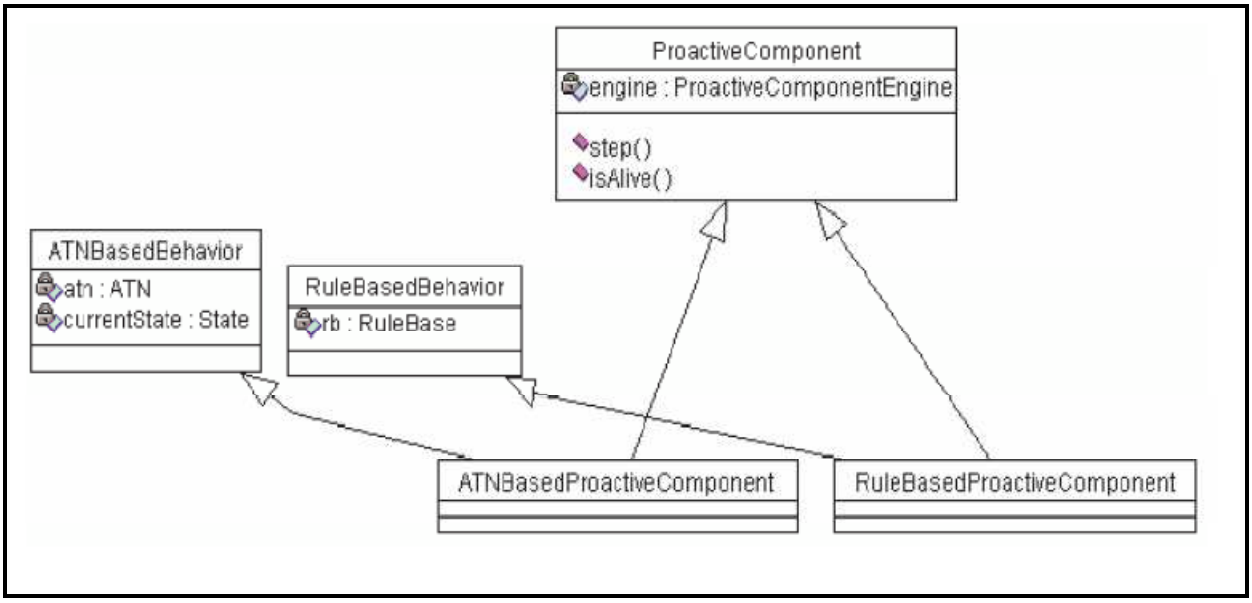

Figure 27: Exemple d'un composant proactif

Agent adaptatif avec DIMA

L'agent adaptatif de DIMA se base sur une cumulation des avantages des deux approches cognitif et réactif en utilisant des couches réactives, pour réagir rapidement aux stimuli connus, et des couches cognitives, pour adapter les réactions de la couche réactive aux variations imprévues de l'environnement.

Il faut donc pouvoir doter l'agent d'un système de décision principal, appelé système comportemental, faisant le lien entre les données reçues en entrée et les données qu'il doit fournir en sortie. Ce système comportemental peut être suffisant pour permettre à l'agent d'interagir dans son environnement, mais pour permettre une adaptation de l'agent, il devient nécessaire de lui greffer un ou plusieurs autres systèmes décisionnels qui s'en chargent. Ces autres systèmes, bien que pouvant gérer directement le comportement de l'agent, sont surtout là pour mesurer, tester et améliorer la réactivité de l'agent. Pour ce faire, ils doivent disposer d'informations sur l'agent et son environnement, mais surtout d'informations concernant le fonctionnement du système comportemental, des méta-informations. Ils doivent de plus adapter ce système comportemental par le biais de méta-actions et, ainsi, adapter l'agent à son environnement.

Un méta-comportement est introduit dans l'architecture d'agents adaptatifs de DIMA. Ce méta-comportement donne à chaque agent la capacité de prendre des décisions appropriées au sujet de contrôle ou d'adapter son comportement avec le temps à des nouvelles circonstances. Il fournit à l'agent un mécanisme d'auto-contrôle pour adapter dynamiquement ses comportements selon son état interne et celui de son monde.

Le méta-comportement se base sur les données de l'agent lui-même, son univers, et le système de décision utilisé.

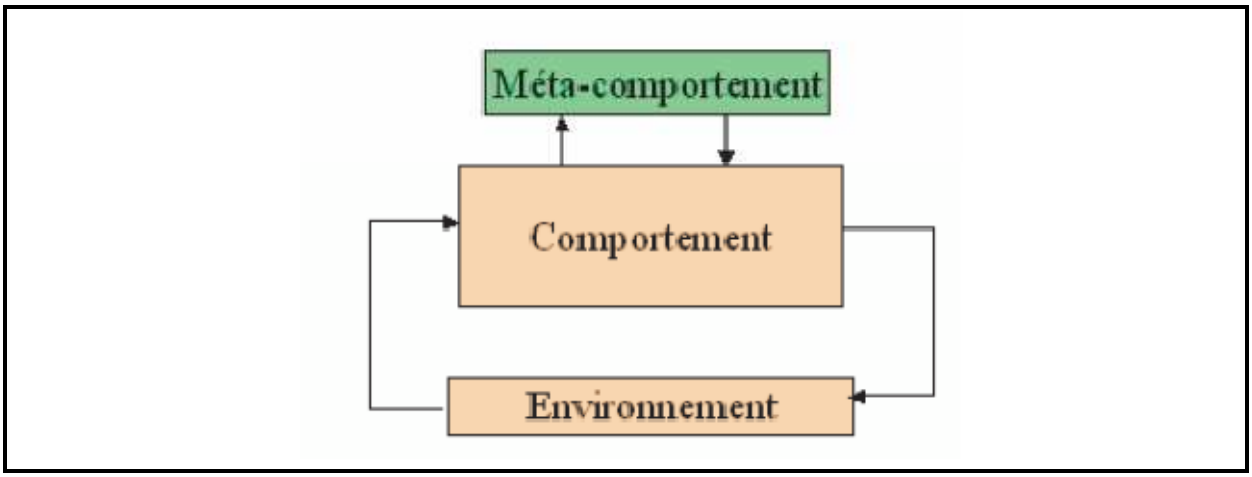

Figure 28: Architecture d'agent adaptatif avec DIMA

Actuellement DIMA intègre trois types de méta-comportements : *RuleBasedMetaBehavior*, *GeneticBasedMetaBehavior*, *ClassifierBasedMeta-Behavior*. Chacun de ces méta-comportements est utilisé avec deux types de comportements : *RuleBasedProactiveComponent* et *ATNBasedProactiveComponent*.

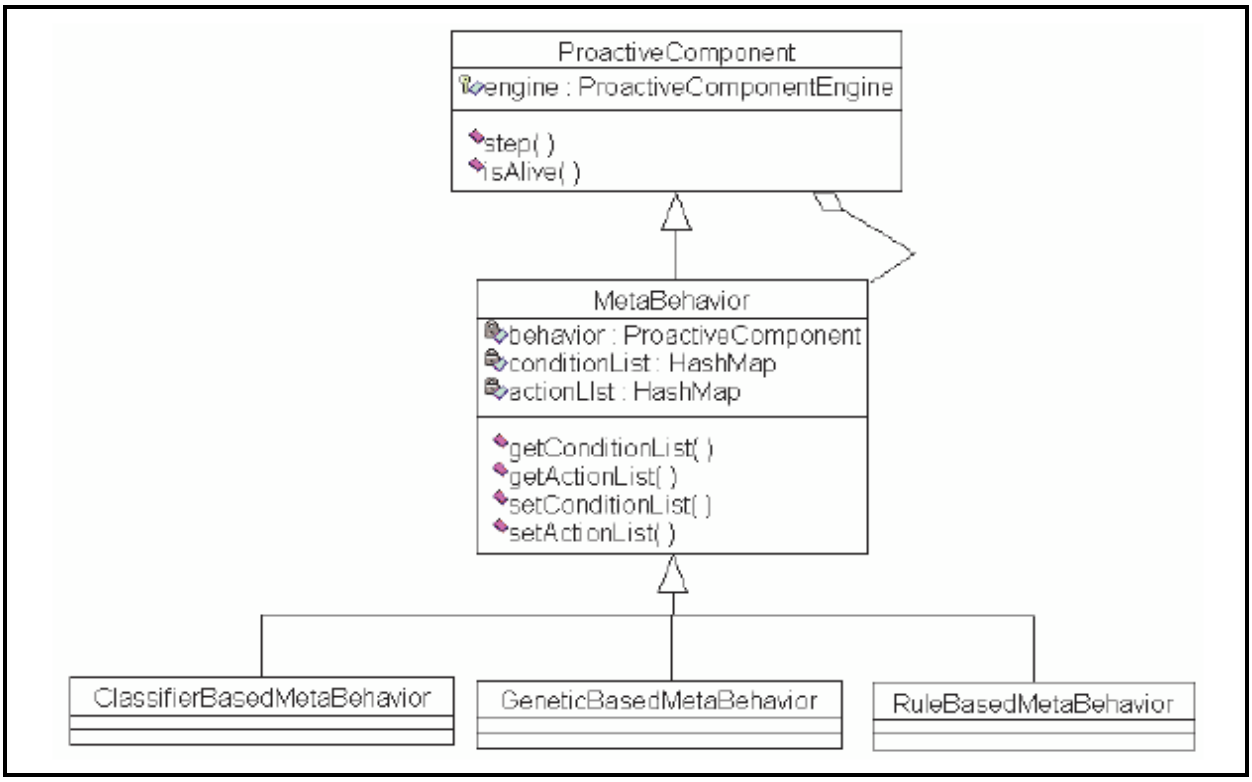

Figure 29: Hiérarchies des classes représentants les agents adaptatifs

Ce méta-comportement a deux principaux rôles :

- **Adaptation structurelle** : il s'agit d'adapter la structure de l'agent à l'évolution de son environnement.

- **Adaptation comportementale** : il s'agit d'adapter le processus de décision de l'agent à l'évolution de son environnement.

Ces deux rôles peuvent être combinés pour avoir à la fois des agents qui adaptent leur structure et leur comportement.

4.6.1.6 BOND

BOND est un Framework d'agents et d'objets distribués. Il a été le premier système d'agents basés sur Java à offrir un haut degré d'adaptabilité. Les auteurs définissent la notion d'agent *mutable,* c'est-à-dire un agent qui est auto-adaptable ou qui peut être adapté par une

entité externe. Le modèle BOND a été appliqué pour concevoir des agents fortement adaptables et mobiles, notamment dans le cadre de systèmes hautement hétérogènes tels que les réseaux *ad hoc*, les workflows adaptatifs ou les services multimédia.

La couche agent de BOND est une extension de la plateforme agent JADE. L'architecture d'un agent est spécifiée par Blueprint, qui est un langage déclaratif conçu pour décrire la structure interne des agents BOND. Les principaux composants d'un agent BOND sont :

- Une base de connaissances,
- Un ensemble de plans d'automates,
- Un ensemble d'états, Ainsi que les transitions entre ces états.

Les transitions dans BOND représentent un changement de «stratégie», chaque stratégie étant un comportement caractérisé par des informations méta (pré-conditions post-conditions, équivalence avec une autre stratégie, ressources matérielles requises, ...). Les plans de l'automate multi-plans expriment des activités parallèles, c'est-à-dire une exécution simultanée des diagrammes état-transition.

Pour adapter dynamiquement des agents, les auteurs introduisent une technique de mutation appelée «chirurgie d'agents» (« *agent surgery* »). Cette technique décrit les mutations comme des séries d'opérations primitives dans un automate multi-plan. Les différentes opérations primitives sont : *l'ajout ou la suppression d'un état ou d'une transition, le remplacement de la stratégie d'un état* (*ex.* passer de l'utilisation d'une interface graphique à l'utilisation de lignes de commandes lors d'une migration de l'agent sur une machine dépourvu d'environnement graphique), *le découpage d'une transition en ajoutant un état intermédiaire, la suppression d'un état intermédiaire dans une transition* (*ex.* suppression du mode débogage), *l'ajout et la suppression de plans* (*ex.* ajouter, remplacer ou supprimer une fonctionnalité telle que le contrôle à distance, le mécanisme de raisonnement ou la négociation), *la fusion ou la fission d'agents* (*ex.* fusionner des agents de contrôle pour avoir un contrôle unifié et optimiser la communication et la mémoire utilisée dans le système, fissionner un agent pour appliquer des priorités différentes à ses fonctionnalités), et *la réduction d'agents* (*ex.* supprimer avant une migration les états et les transitions par lesquels on ne doit pas ou plus passer).

4.6.2 Adaptation au niveau de l'organisation

Dans ces modèles de systèmes l'adaptabilité est considérée comme un phénomène émergent de l'interaction entre l'ensemble des agents. Par émergence on désigne un

comportement non prévu résultant de la succession naturelle des interactions des agents entre eux. Ce phénomène est guidé par des tendances propres à chaque type de société appelées tendances fondamentales.

Donc l'adaptabilité dans ce type de systèmes se fait par le changement de relations entre agents pour qu'ils soient conformes aux changements de leur contexte.

4.6.2.1 Tendances fondamentales

"*Certains systèmes, naturels ou artificiels, sont conçus pour qu'ils satisfassent à des besoins très généraux qui orientent leur comportement de manière décisive. Ces besoins généraux, multiples et contradictoires, seront appelés tendances fondamentales du système*."[1]

Donc les systèmes dont le comportement est conduit par des tendances fondamentales, tel que, se nourrir, se protéger, sont appelés des systèmes adaptatifs, puisque leur comportement et même leur structure vont être changés pour satisfaire ces besoins.

4.6.2.2 L'émergence

L'émergence est un phénomène d'un comportement qui apparaît suit aux interactions entre les composants d'un système, sans qu'il soit initialement voulu par le concepteur, ni même pas connu à l'avance.

"*L'émergence ne se réduit ni à la somme ni à la différence des forces co-opérantes. Les propriétés inter-reliées, communes qui permettent d'identifier un phénomène comme émergent sont:*

- *L'observation d'un phénomène ostensible (qui s'impose à l'observateur) et radicalement nouveau au niveau globale ou macro, irréductible à des propriétés associées à des composants du micro niveau."d'une part, l'émergence présuppose qu'il y a apparition de nouveauté – propriété, structure, formes ou fonctions –, et d'autres part, elle implique qu'il est impossible de décrire, d'expliquer ou de prédire ces nouveaux phénomènes en termes physiques à partir des conditions de base définies aux niveaux inférieurs"*

- *La cohérence et la corrélation du phénomène (il a une propre mais liée fortement aux parties qui le produisent). "L'émergence fait référence à l'apparition durant le processus d'auto-organisation dans un système complexe*

[1] Alain Cardon, Zahia Guessoum. SYSTEMES MULTIAGENTS ADAPTATIFS.

de structures ou de schémas ("patterns") ou de propriétés nouvelles et cohérentes".

- *L'observation d'une dynamique particulière (le phénomène n'est pas pré-donné, il y a "auto-maitien" du phénomène). Langton définit l'émergence en termes de relations de feedback entre les niveaux dans un système dynamique. Les micro-dynamiques locales causent les macro-dynamique et les macro-dynamiques globales contraignent les locales."*[1]

4.6.2.3 La théorie AMAS

La théorie AMAS (Adaptive Multi-Agent Systems) est une théorie qui guide la réalisation de systèmes complexes adaptatifs, et qui donne des critères locaux de conception des agents qui permet l'émergence d'une organisation et de sa fonction globale dans le système.

La résolution de problèmes avec les AMAS s'articule ainsi autour de trois notions essentielles :

- Apprentissage : le processus de résolution est incomplètement spécifié, donc au cours de son activité le système doit progresser à partir d'observation sur l'environnement.

- auto-organisation : est le résultat de l'opération d'apprentissage, et qui influence une action de coopération entre les entités formant le système, à savoir les agents.

- Emergence : l'émergence signifie que le résultat obtenu n'est pas prévu à l'avance, par la spécification de système, ni même pas par le processus d'apprentissage.

L'adéquation fonctionnelle d'un système

Elle concerne un jugement effectué par un observateur sur la patience de son activité dans l'environnement

[1] J Georgé, M P Glaizes, P Glize. Conception de systèmes adaptatifs à fonctionnalité émergentes: la théorie Amas

Théorème : *Pour tout système fonctionnellement adéquat, il existe au moins un système à milieu intérieur coopératif qui réalise une fonction équivalente dans le même environnement.*[1]

Adapter le système par ses parties

Cette propriété signifie que l'agent a l'autonomie de changer ses liaisons avec les autres agents pour avoir une coopération dans l'organisation. Donc l'adaptation aux changements de l'environnement émerge sous forme d'organisation entre les agents.

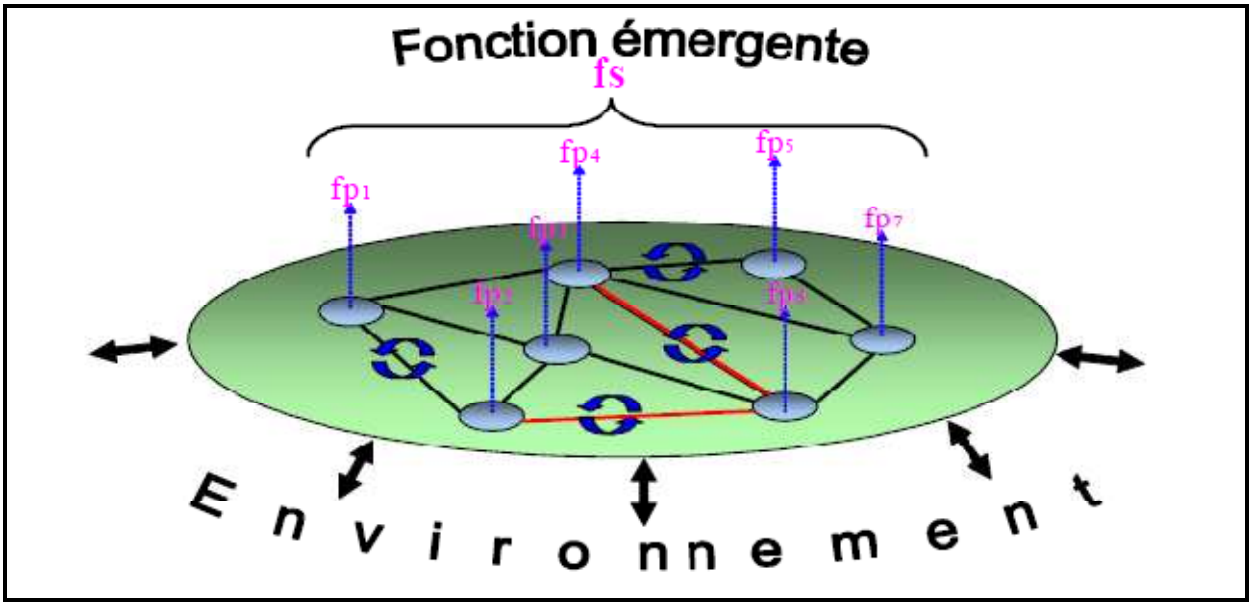

Figure 30: Adaptation avec émergence

Les composants d'un agent AMAS

Les agents dans un AMAS sont programmés pour être dans une situation coopérative avec les autres agents de système. Cela se traduit par le fait qu'à tout instant, un agent reçoit les informations pertinentes pour réaliser sa fonction et qu'il transmet des informations utiles à d'autres.

[1] J. Georgé, M. Gleizes, P. Glize. Conception de systèmes adaptatifs à fonctionnalités émergentes : la théorie Amas.

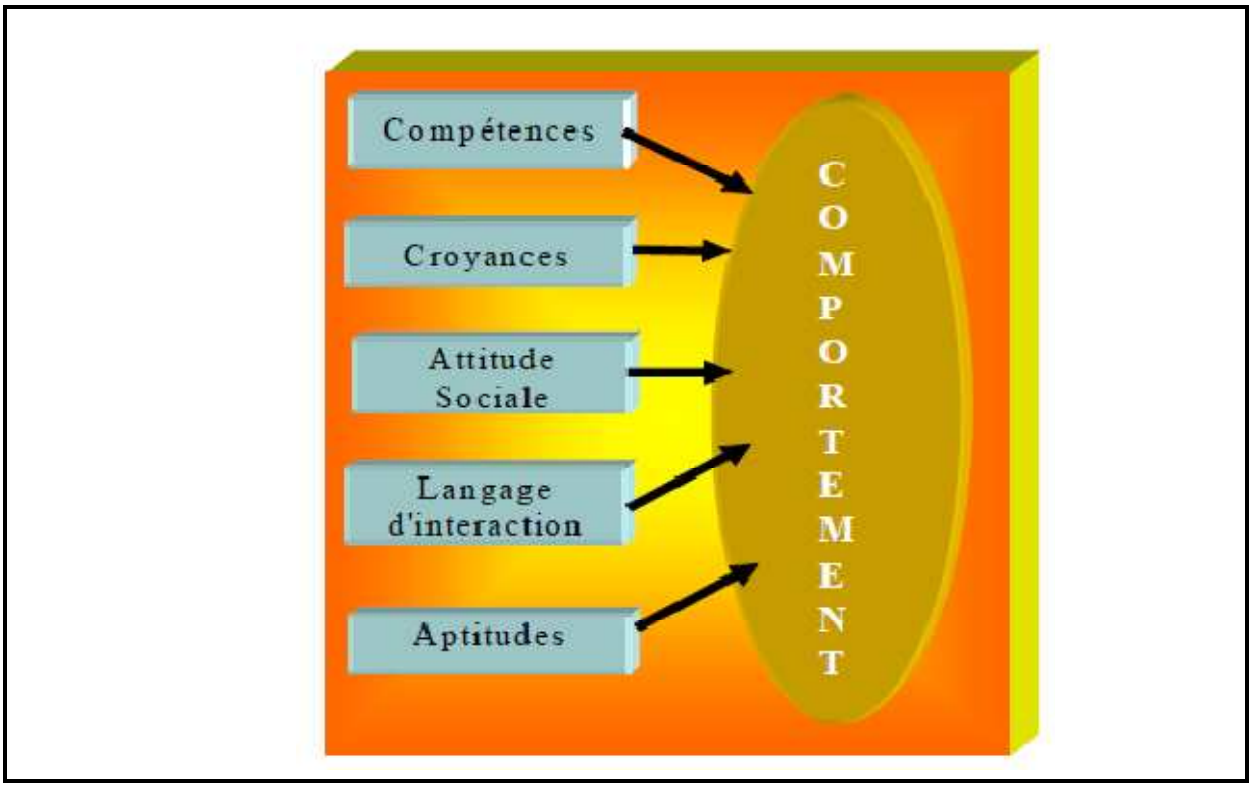

Figure 31: Les composants d'un agent AMAS

En générale le comportement d'un agent coopératif doit être composé de trois composants essentiels suivants :

- **Compétences :** Ce sont les connaissances que possède l'agent pour accomplir une tâche particulière qui lui a été assigné

- **Croyances :** se sont les informations que l'agent possède sur lui-même, sur les autres agents et sur son environnement.

- **Attitude sociale :** c'est le composant le plus important de la théorie, elle représente les critères permettant à l'agent de décider de son comportement et la réorganisation avec les autres agents.

- **Langage d'interaction :** il permet à l'agent de communiquer soit directement par des messages ou indirectement par l'influence sur l'environnement.

- **Aptitudes :** sont les capacités de l'agent pour réviser ses représentations et sa connaissance.

Les situations de non coopération

Lors de l'interaction entre agents dans une organisation multi agents trois situations de non coopération, peuvent avoir lieu :

1. Un signal est incompris ou possède de multiples interprétations (ambigüité). Dans ce cas, un agent coopératif ne va pas ignorer le signal car il le considère nécessaire à l'activité du système. Il va donc tenter de le transmettre à d'autres agents qu'il estime plus compétents ou bien se faire aider par d'autres pour enlever l'ambigüité.

2. L'information reçue est déjà connue ou n'a aucune conséquence logique. L'agent coopératif ne peut pas tirer profit de cette information pour transformer le monde. Il va donc chercher d'autres agents qui pourraient en bénéficier.

3. Compte tenu de ses croyances courantes, l'agent considère que la transformation de l'environnement qu'il peut opérer n'est pas bénéfique à autrui. Cette situation englobe les notions de conflit et de concurrence qui sont fréquemment étudies dans le domaine. Par exemple, un conflit de résultat peut survenir si l'agent aboutit à une conclusion opposée à celle d'un autre. Une concurrence est détectée si l'agent aboutit à des conclusions identiques à celles d'un autre.

4.6.2.4 La méthodologie ADELF[1]

ADELFE (Atelier de DEveloppement de Logiciels à Fonctionnalités Emergentes) est un projet RNTL (2001-2003) dont l'objet est de réaliser un Atelier de DEveloppement de Logiciels à Fonctionnalité Emergente destiné à des concepteurs informatiques non spécialistes de ce type de logiciels. La réalisation de cet atelier de développement consiste à fournir :

- Une notation basée sur UML,

- Une méthodologie de conception,

- Une plate-forme conçue sous l'outil OpenTool, comprenant un outil de modélisation graphique, une bibliothèque de composants permettant des simulations et un prototypage rapide.

- Une technique d'auto-assemblage de composants logiciels que sont les agents auto-organisateurs.

[1] Carole Bernon, Valérie Camps, Marie-Pierre Gleizes, Pierre Glize, La conception de systèmes multi-agents adaptatifs : contraintes et spécificités.

Dans ADELFE, il s'agit de réutiliser et d'étendre UML (Jacobson et al., 1999) pour avoir des capacités de description adaptées aux agents appartenant à des systèmes multi-agents adaptatifs. En effet, l'interface d'une classe est statique en programmation par objets et donc en UML, alors qu'elle est dynamique pour un agent. Le spectre d'utilisation de la notation UML peut être étendu d'une part, par la notion de stéréotype et d'autre part, par la définition de nouveaux types de schémas pouvant être spécifiés au niveau du méta-modèle décrivant la notation UML elle-même.

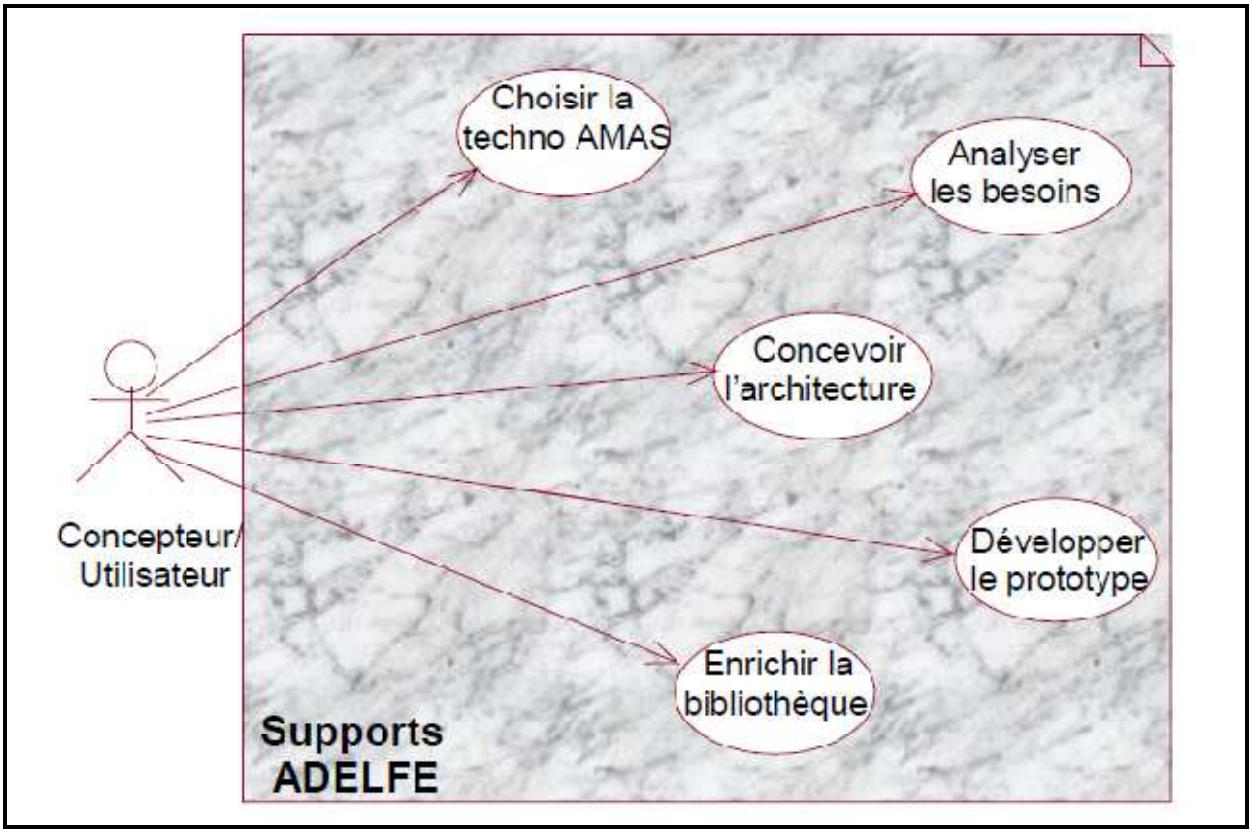

Figure 32: Les fonctions d'ADELFE

La phase d'adéquation des AMAS

Dans une première étape, un outil interactif permet au concepteur de savoir si un système multi-agent adaptatif est utile pour l'application proposée. Cet outil vérifie la pertinence de l'utilisation de la méthodologie, selon les critères suivants :

- mauvaise définition de la tâche (cas sans solution évidente),
- avancée très progressive par essais successifs,
- grande complexité, absence d'autres méthodes,

- frontières mal définies,
- distribution physique ou fonctionnelle,
- environnement évolutif,
- besoin d'interactions - coopération entre entités.

La phase d'analyse

La phase d'analyse comporte l'étape classique d'expressions des besoins c'est-à-dire exprimer ce que le client souhaite que son système fasse. L'expression des besoins consiste à définir les fonctions globales que doit réaliser le système. Pour les systèmes multi-agents, l'expression des besoins se traduit, en général, par des descriptions de scénarios possibles plus ou moins génériques entre des agents (par exemple, le réseau de contrats). Pour ADELFE, l'expression de la fonction globale que doit réaliser le système n'intervient pas dans la phase de conception des agents cette fonction émergera du comportement du collectif et sera apprise. Les scénarios d'interactions entre les agents, exprimés au niveau de l'analyse comme dans la plupart des méthodologies orientées agents, sont utilisés pour décrire une partie de leurs interactions (une partie seulement car le traitement des situations non coopératives par la mise en œuvre d'interactions n'apparaît pas au niveau de l'analyse mais seulement au niveau de la conception).

La phase de conception

La phase de conception consiste à donner un comportement aux agents et à l'environnement du système multi-agents. Pour la conception d'un agent, ADELFE propose une description d'agent générique composée de sept modules :

- Un module de communication avec les autres agents,
- Un module de communication avec l'environnement,
- Un module de croyances sur lui-même,
- Un module de croyances sur les autres agents,
- Un module de croyances sur son environnement,
- Un module de compétences,

- Un module d'attitude sociale lié à la coopération.

4.7 Conclusion et critiques

Dans cette partie de notre mémoire on a met en claire le domaine dans lequel, on est en train de travailler "l'adaptation dans les systèmes multi agents". Dans un premier temps nous avons définies les notions généraux et critères liés à l'adaptation. Ensuite on a présenté les deux onglets de travaux qui traitent l'adaptabilité dans les systèmes multi agents. Le premier onglet concerne l'adaptation au niveau de l'agent lui même et quant au deuxième il traite l'adaptation au niveau de l'organisation, puis on a cité quelques travaux récents dans chacun des deux onglets. On a sorti avec la conclusion que les différents travaux étudies arrivent à un consensus que le comportement adaptatif de l'agent doit être séparé de son comportement normal, d'une manière à concevoir une architecture à deux niveaux, un niveau concerné par le comportement d'adaptation et un niveau concerné par le comportement habituel.

Après cette tourné dans le domaine, on a choisi d'entamer le travail dans l'adaptation au niveau de l'agent, et particulièrement l'approche à base de composants.

Chapitre 5: Agent auto adaptable à base de composants

5.1 Introduction

Dans un système multi agents on a besoin de l'adaptation lorsque l'agent évolue dans un environnement où il ne peut pas connaître d'une manière exhaustive toutes les situations qu'il peut rencontrer.

Nous avons choisi pour le processus de l'adaptation l'utilisation de l'approche par composants qui permet la réutilisation des composants déjà disponibles, donc un gain très important en termes de temps et d'effort. De plus, le modèle par composants facilite l'adaptation par le fait que les relations entre les composants sont des interfaces bien définies, qui facilite l'ajout de nouveaux composants, et/ou le retrait de ceux inutiles. D'autre part la conception de l'agent en deux niveaux, le niveau de supervision et le niveau de fonctionnement, permet de mieux éclairer la structure de l'agent et facilite son adaptation du fait que seule la partie de fonctionnement doit être abordée par le sujet d'adaptation.

On a utilisé une modélisation par réseau de Petri de l'opération de génération de nouvelles configurations à partir des composants disponibles et de la description du changement de l'environnement.

5.2 L'approche suivie

L'agent dans notre modèle est constitué de deux blocs, un bloc d'adaptation et un bloc de fonctionnement. Dans le bloc de supervision sont positionnées les fonctions de contrôle et d'adaptation, donc ce bloc est chargé d'assurer le bon fonctionnement de l'agent, et son adaptation aux évolutions de son environnement. Le deuxième bloc est le bloc d'exécution (de fonctionnement) qui est une sorte d'agent classique, ce bloc a la propriété d'être conçu par des composants indépendants, qui sont sélectionnés et assemblés au besoin.

Cette séparation entre les fonctions liées à la supervision, et les fonctions liées au comportement normal permet de faciliter l'adaptation de l'agent, en reconstituant le bloc de fonctionnement (par réassemblage de composants) sans remettre en cause le bloc de supervision.

La relation entre les deux niveaux est comme suit, le niveau de supervision est chargé du bon fonctionnement du système, donc il peut reconfigurer le niveau de fonctionnement par réassemblage pour rester performant avec les changements de l'environnement. Le niveau bas (l'agent), quant à lui, est le responsable de l'exécution des tâches qui lui sont octroyés.

L'agent perçoit son environnement par le *mécanisme d'observation de l'environnement* qui permet la collecte des informations sur l'état de l'environnement. Et agit

sur celui-ci via ses effecteurs, par les actions qu'il est chargé de faire (déplacer, nourrir, …) et aussi par la sollicitation des autres agents ou de l'utilisateur (demande d'aide par exemple pour des composants qui manquent dans sa base).

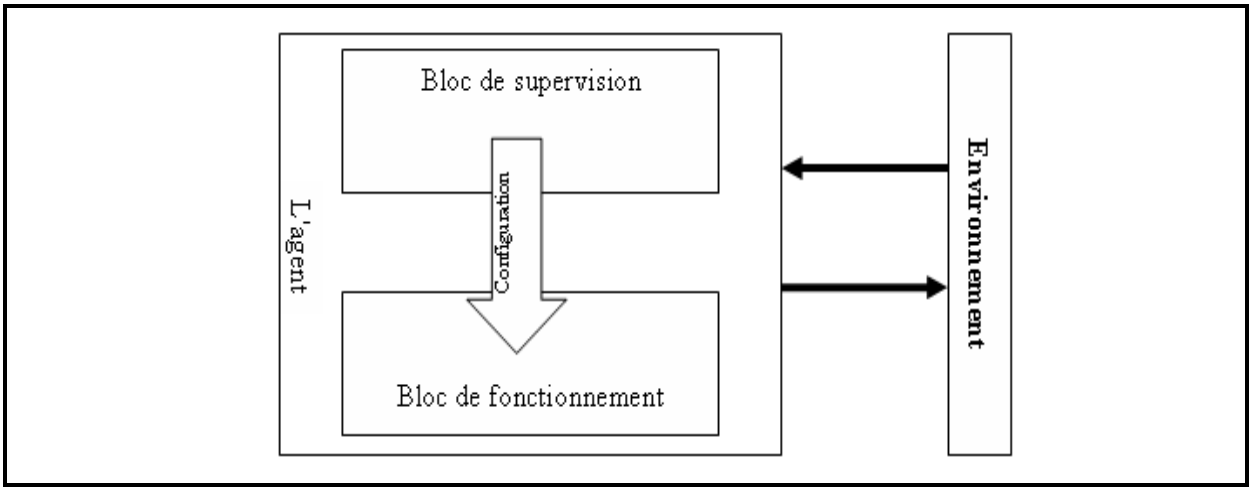

Figure 33: Le modèle d'agent en deux niveaux

5.3 Structure de l'agent auto-adaptable

La figure suivante, montre la structure de l'agent auto-adaptable à base de composants, conçu en deux blocs. Le bloc d'adaptation et de supervision, est formé (composé) par un ensemble de composants :

- **Un mécanisme d'observation de l'environnement** qui permet de capter les évènements de l'environnement, les analyser pour reconnaître les changements et les envoie soit au bloc *générateur de comportement* ou bien au bloc de fonctionnement,

- **Un générateur de comportement** il est le cœur du bloc de supervision, il permet de construire (constituer) le comportement de l'agent par assemblage de composants (de la base de composants) en fonction des changements de l'environnement, émis par le mécanisme d'observation de l'environnement, en cas où les composants disponibles dans la base ne suffisent pas pour répondre aux évènements de l'environnement un **module d'extension** permet de demander l'aide des autres agents ou bien de l'utilisateur.

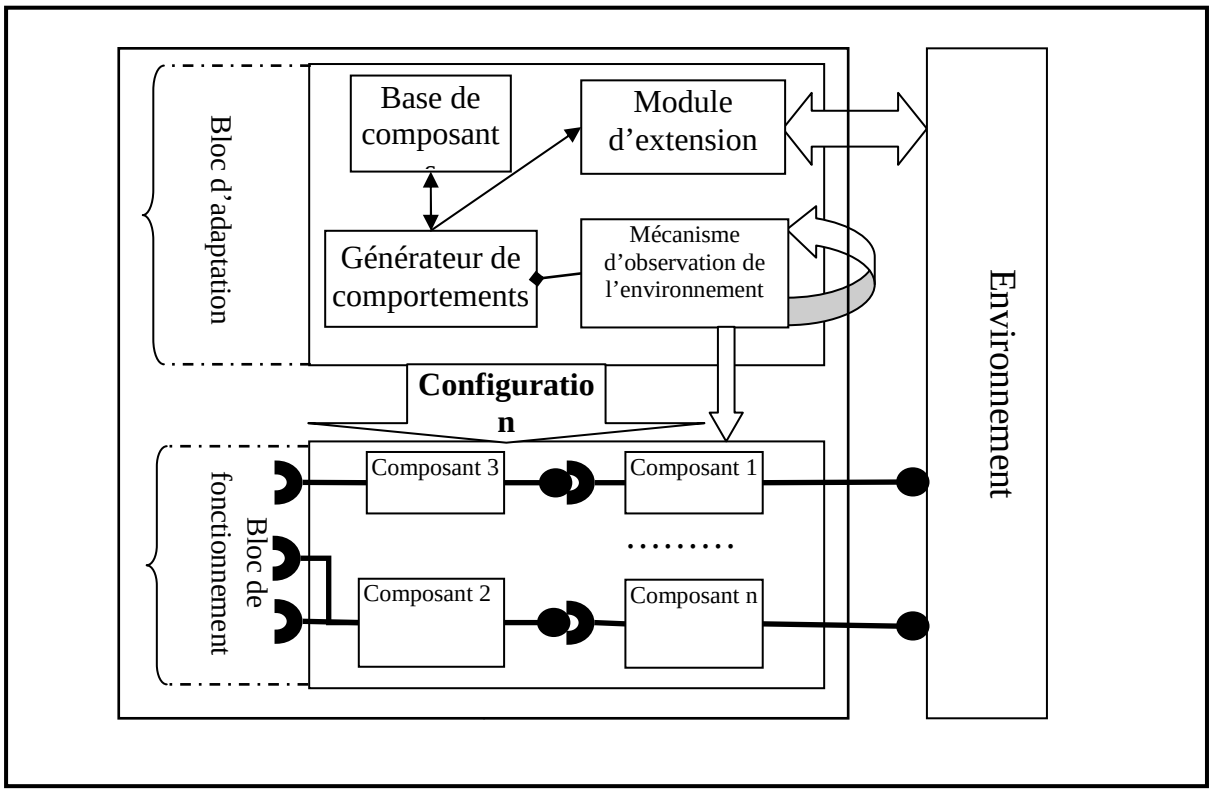

Figure 34: Architecture en deux niveaux d'un agent auto-adaptable à base de composants

5.3.1 Le bloc d'adaptation et de supervision

Le rôle de niveau d'adaptation et de supervision est de:

1) Capter les évènements venus de l'environnement sur lequel l'agent se trouve, est cela grâce au mécanisme d'observation de l'environnement ;

2) Si l'évènement est normal (changement pris en charge), il l'envoi au niveau d'exécution ;

3) Si c'est un changement qui nécessite une adaptation, il choisit les composants appropries pour la situation reconnue ;

4) Et enfin, il lance l'exécution de la composition de l'agent obtenue par les nouveaux composants (exécute la composition de l'agent, par les nouveaux composants).

5.3.1.1 Le mécanisme d'observation de l'environnement

C'est grâce à ce mécanisme que l'agent peut observer l'état de son environnement, et reconnaître les différents évènements qu'il émit.

Le mécanisme d'observation de l'environnement capte les évènements émis par l'environnement à la destination de l'agent, ces évènements sont faits sous forme de paires de deux ensembles d'attributs Ev(E, S), où E : représente l'ensemble des attributs d'entrée de l'évènement et sont captés par les interfaces requises de l'agent, lorsque un évènement arrive ces attributs prennent des valeurs à être exécuter et pour rendre les résultats sur les interfaces fournies, et S : l'ensemble des attributs de sortie de l'évènement et qui sont donnés par les interfaces fournies de l'agent.

Le mécanisme d'observation de l'environnement compare les deux ensembles E et S, avec les interfaces requises et fournies par les différents composants de l'agent, s'il ne trouve pas une similarité entre ces deux ensembles alors l'évènement n'est pas pris en charge par la configuration actuelle de l'agent et une opération d'adaptation est nécessaire pour prendre en charge l'évènement, donc l'évènement (les deux ensembles d'attributs E et S) est envoyé au générateur de comportement. Dans le cas contraire, l'évènement est envoyé au bloc de fonctionnement, et les valeurs des attributs d'entrée de l'évènement sont exécutés et les résultats sont retournés à l'utilisateur à travers les interfaces fournies.

Algorithme 1 : Mécanisme d'observation de l'environnement

Debut

Boolean Reconfiguration=Faux ;
Boolean Trouve= Faux;

Lire (évènement) ;

Pour « toute un élément e de l'ensemble E » **Faire** // interfaces requises par l'évènement

Pour « tous les interfaces i requise de l'agent I» **Faire**

Si « e == i » **alors**

Trouve = vrai ;

sortir de la boucle ;

FinSi

FinPour

Si Trouve==Faux **alors**

Reconfiguration ==vrai ;

Aller à Etq ;

FinSi

Trouve=Faux ;

FinPour

Trouve=Faux ;

Pour « toute un élément s de l'ensemble S » **Faire** // requises par la requête

Pour « toute interface o requise de l'agent O » **Faire**

Si « s == o » **alors**

Trouve = Vrai ;

sortir de la boucle ;

FinSi

FinPour

Si Trouve==Faux **alors**

Reconfiguration ==vrai ;

Aller à Etq ;

FinSi

Trouve=Faux ;

FinPour

Etq :

Si (Reconfiguration == Vrai) **alors**

générateur de comportement (Ev(E,S)) ;

Sinon

bloc de fonctionnement (Ev(E,S)) ;

FinSi

Fin.

5.3.1.2 Le générateur de comportement

C'est le cœur du bloc d'adaptation, il est activé par *le mécanisme d'observation de l'environnement* suite à un évènement, qui fait référence à une fonction qui n'est pas prise en charge par l'agent dans son état actuelle.

Pour déterminer l'architecture du nouveau rôle (comportement) le bloc va parcourir un réseau de Petri de la base de composants, et prenant en compte les paramètres de changement de l'environnement et génère automatiquement une configuration convenable.

Plusieurs configurations peuvent répondront à une situation, c'est-à-dire générer par l'algorithme donc le choix d'une configuration pour l'implémentation peut dépendre du contexte et d'une évaluation maximisant les bénéfiques.

5.3.1.3. La base de composants

Elle contient une représentation de l'ensemble de composants que l'agent peut utiliser pour l'assemblage de son comportement, cette représentation spécifie pour chaque composant les interfaces fournies et requises, ainsi qu'une identification unique dans la base (le nom de composant). *La base de composants* est ouverte à l'extérieur pour des mises à jour par l'ajout de composants, soit en réponse à des sollicitations du *module d'extension*, soit par l'utilisateur.

5.3.1.4. Le module d'extension

Ce module est déclenché par *le générateur de comportement*, suite à un cas de manque de composants dans *la base de composants*, il est chargé de lancer une demande d'aide à la destination de l'environnement (Les autres agents, ou l'utilisateur), contenant une spécification des composants demandés, sous forme des interfaces fournies et requises.

Les composants recherchés et localisés sont directement insérés dans la base de composants.

5.3.2 Le bloc de fonctionnement

Le bloc de fonctionnement (voir Figure 3) constitue une sorte d'agent classique, constitué d'un ensemble de composants, relié entre eux. Ce bloc est vu de l'extérieur comme un ensemble d'interfaces fournies et d'interfaces requises.

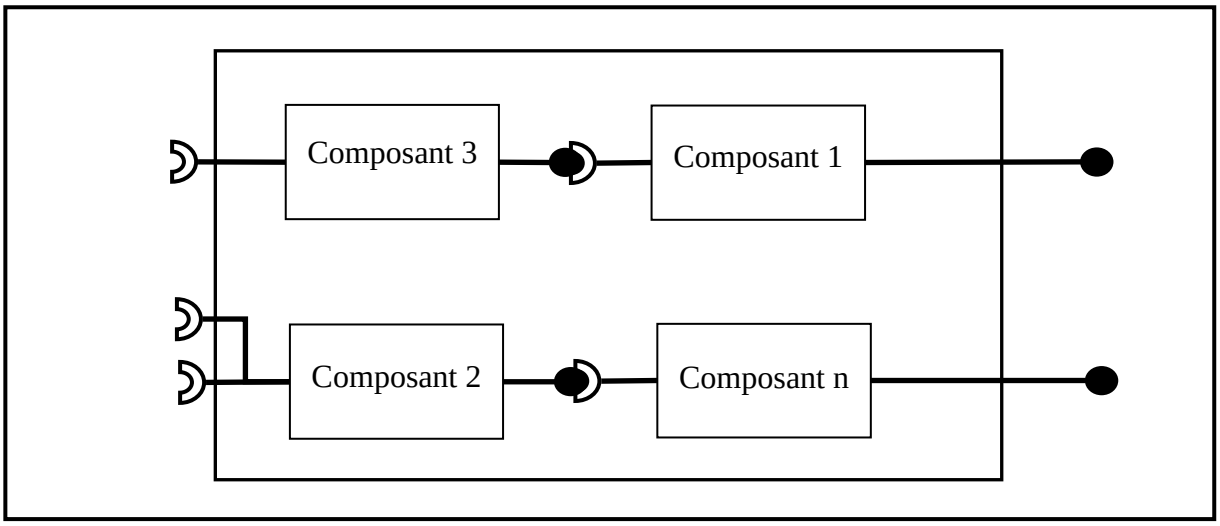

Figure 35: Le bloc de fonctionnement

5.4 Génération de configurations

Pour automatiser l'opération de génération de nouvelles configurations par le générateur de comportement on a utilisé une modélisation par réseau de Petri, et on a suivi l'approche suivante.

On a représenté les composants par des transitions, et les interfaces requises et fournies des composants par des places. Donc pour une transition ses places d'entrés représentent les interfaces requises de composant, et ses places de sorties représentent ses interfaces fournies. La base de composants BC, sera donc un réseau de Petri dont les places sont tous les interfaces des composants, et les transitions représentent tous les composants.

Et pour le système global, l'ensemble des places qui n'ont pas de transition d'entrées sont les interfaces requises de l'agent, et l'ensemble des places qui n'ont pas des transitions de sorties sont les interfaces fournies de l'agent.

L'évènement reçu de l'environnement est présentée sous forme d'un paire Ev(E, S), E : l'ensemble des interfaces requises, et S: l'ensemble des interfaces fournies.

Le problème de répondre au changement de l'environnement sera comme suit, En partant du marquage initial (qui correspond à l'ensemble {E}) et en collectant les transitions franchises à chaque étape, si on arrive à tous les places correspondantes aux attributs de sortie de l'évènement (l'ensemble {*S*}) dans ce cas on donne la suite de transitions suivi pour atteindre le marquage final comme étant l'ensemble des composants à utiliser dans l'assemblage suivant. Dans le cas contraire, c.-à-d. Si on n'arrive pas à tous les interfaces fournies souhaités alors l'agent va demander l'aide de l'utilisateur ou de son l'environnement.

5.4.1 Représentation de la base de composants

En premier lieu on va représenter chaque composant par un simple réseau de Petri comme illustrer dans la Figure 37, avec un ensemble d'interfaces requises et un ensemble d'interfaces fournies, et la BC par un réseau de Petri complexe tel que :

BC = (I, C, F) où :

- I : l'ensemble des places représentant les interfaces requises et fournies, par tous les composants ;
- C : l'ensemble de tous les composants, disponibles dans la base ;

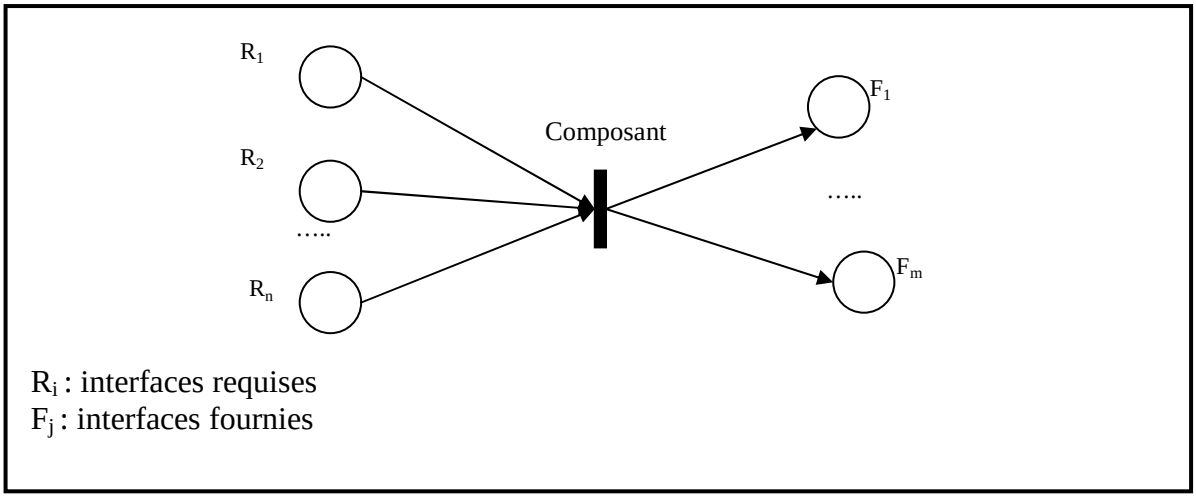

Figure 36: Représentation par un RDP d'un composant

- F :(I×C) U (C×I) → {0,1} une fonction de correspondance qui indique la présence (F(i,c)= 1) ou non (F(i,c) = 0) d'un arc entre une place et une transition ou l'inverse.

Donc la base sera une sorte d'un ensemble de composants et chacun de ces composants et un simple réseau de Petri contenant une seule transition et un ensemble de places représentent ses interfaces requises et fournies.

5.4.2 Ajout de nouveaux composants à la base

Pour ajouter à la base un nouveau composant c(R,F), avec R :ensembles d'interfaces requises et F :ensemble d'interfaces fournies, on va poursuivre comme suit :

1. Une transition correspondante au composant est ajoutée à l'ensemble C ;

2. Pour tous les interfaces requises du composant sont ajoutés des places à l'ensemble I, et

3. Pour tous les interfaces fournies du composant sont ajoutés des places à l'ensemble I ;

5.4.3 Représentation de l'évènement

Pour l'évènement Ev il est présenté en deux paramètres Ev (E, S) , où :

- E : L'ensemble des interfaces requises;

- S : L'ensemble des interfaces fournies;

Ces deux paramètres représentent les interfaces de la configuration, qui doivent résulter après la phase d'adaptation.

5.4.4 Configuration initiale

C'est le marquage initial du RDP, tel que, toutes les places correspondantes aux interfaces requises (l'ensemble E) possèdent des jetons, et aucune des autres places ne possède de jetons.

En fonction du marquage initiale, plusieurs transitions peuvent être franchissables, une transition est franchissable SSI tous ses places d'entrée contiennent des jetons.

5.4.5 Règles de franchissement

A chaque franchissement d'une transition des jetons sont retirés de ses places d'entrée et insérée dans ses places de sortie, indiquant le passage des interfaces requises d'un composants vers ses interfaces fournies, et cela jusqu'à l'arrivé à l'ensemble des interfaces fournies de l'évènement.

Pour toute transition S on défini :

$c^{\bullet}$ = {*i* € I / F(c,i)=1}, l'ensemble de tous les places de sorties de c (les interfaces fournies de composant),

${}^{\bullet}c$ = {*i* € I / F(i , c) = 1} , l'ensemble de tous les places d'entrées de c (les interfaces requises de composant)

5.4.6 Séquence de franchissement

Une séquence de franchissement ∂ est une suite de transitions, part du marquage initiale M_0 vers un autre marquage M_n.

Si M_n représente un marquage finale, c.-à-d un marquage où tous les places qui contiennent des jetons sont des places correspondantes aux interfaces fournies spécifies dans l'évènement de l'environnement alors, l'ensemble des transitions de la séquence $\partial=\{c_1,\ldots,c_n\}$, représente la configuration obtenue en réponse à l'évènement reçu.

5.4.7 Problème de composition de l'agent

Le problème de la composition de l'agent consiste, à partir de l'évènement Ev (E , S) et de la BC=(I, C, F), de trouver une séquence ∂ de transitions franchises, partant du marquage initiale (où seulement les places correspondent E sont marqués) vers le marquage finale (Dont seulement les places correspondent a l'ensemble S sont marqués), cette séquence représente l'ensemble de composants à assembler pour obtenir le nouveau comportement de l'agent.

5.4.8 Sélection de composants

Dans cette section on va automatiser à travers un algorithme l'opération de sélection des composants à intégrés dans l'assemblage, de comportement de l'agent.

L'algorithme consiste à parcourir le RDP de la base de composants depuis le marquage initiale qui influence les interfaces requises par l'environnement (E) vers un marquage finale qui correspond aux interfaces fournies spécifies dans l'évènement (*S*).

Algorithme 2 : Génération de configuration

Entrées

BC = (I, C, F) /* un réseau de petri représentant la base de composants, */

E: ensemble des places, représente l'ensemble des interfaces requises de composants,

∂={} /*égale à l'ensemble vide et correspond à l'ensemble de composants à utilisé dans l'assemblage suivant.*/

L'évènement Ev = (E , S) ;

M_c : marquage courant

M_f : marquage final

Franchissable : ensemble des transitions franchissables à un moment donné.

Début

Etq :

M_c= E ;

M_f =Vide;

Franchissable = Vide;

Tanque ($M_c \neq M_f$) **faire**

Pour (Tous i € M_c) **faire**

Si (i € S) **alors**

M_c= M_c – i ;

M_f = M_f U i ;

FinSi

FinPour

Franchissable= Franchissable U {c € C/∀ i € ˙c , i€M_c}

∂ = ∂ U Franchissable ;

M_c = Vide ;

Pour (Tous c€ Franchissable) **faire**

$M_c = M_c$ U c˙ ;

Franchissable= Franchissable-c ;

FinPour

FinTanque

Si (M_f ==S) **alors**

retourner ∂ ;

Sinon

Extension = Module_extension (demande d'aide, Ev(E,S),) ;

Si (Extension = Vrai) **alors**

Aller à Etq ;

Sinon

Retourner (‘’ L'agent n'est pas en mesure de prendre la situation courante ‘’) ;

Finsi

Finsi

Fin

5.5 Conclusion

Dans ce travail on a proposé un modèle d'un agent auto-adaptable à base de composants, la conception est faite en deux niveaux, un niveau haut pour l'adaptation et la supervision et un niveau bas pour le fonctionnement habituel et normal. Cette conception a bien éclairée l'architecture et facilite l'adaptation de fait de la séparation entre le niveau de supervision qui est concerné par l'opération d'adaptation et reste inchangé quelque soit la situation, par contre le niveau fonctionnel évolue d'une manière continue avec l'évolution de l'environnement.

On a ensuite présenté une modélisation de l'opération de l'adaptation par la génération de configurations à l'aide d'un réseau de Petri, notamment la phase de sélection des composants à intégrés dans la nouvelle configuration.

Cette phase a été automatisée par un algorithme qui permet d'aller d'une base de composants, et en fonction de la description du changement de l'environnement et d'arriver à un ensemble de composants à utiliser dans l'assemblage de la nouvelle configuration.

Cette étape assure la génération automatique de l'ensemble de composants qui permettent d'avoir un assemblage capable de confronter le changement de l'environnement.

Mais il reste la vérification de la cohérence de l'assemblage obtenu, puisque l'algorithme n'assure qu'un ensemble de composants une fois mets les uns avec les autres peuvent donner les interfaces requises et fournies, mais ne garantie pas une cohérence d'exécution.

Chapitre 6 : Etude de cas et implémentation

6.1 Introduction

Dans cette partie on va valider notre modèle proposé (agent auto-adaptable à base de composants) sur le cas d'un agent de fourniture de services web. L'agent dans notre exemple offre un ensemble de services web et cela suivant la requête utilisateur.

Une application est développée dans le cadre de ce travail permettant la création d'un modèle par réseau de Petri de la base de composants, ensuite exécutant ce modèle suivant la requête utilisateur pour arriver à une configuration convenable.

6.2 Présentation d'application développée

La figure 37 montre la première interface de l'application qui permet entre autre la création d'une nouvelle configuration, à travers le bouton New-Configuration, l'ajout et/ou le retrait de composant (le bouton Composants/Transition, c à d le composant est représenté par une transition) ou d'interface (qui est représenté par une place, par le bouton, Interface/Place). D'autre possibilités sont encore disponible, tel l'ajout d'une liaison entre Interface>composants ou l'inverse, l'annulation d'un élément (composant ou interface), son déplacement et d'autres possibilités qui ne sont pas encore implémentées comme la sauvegarde de réseau de Petri correspondant à la configuration dans un fichier, et son chargement depuis un emplacement spécifier.

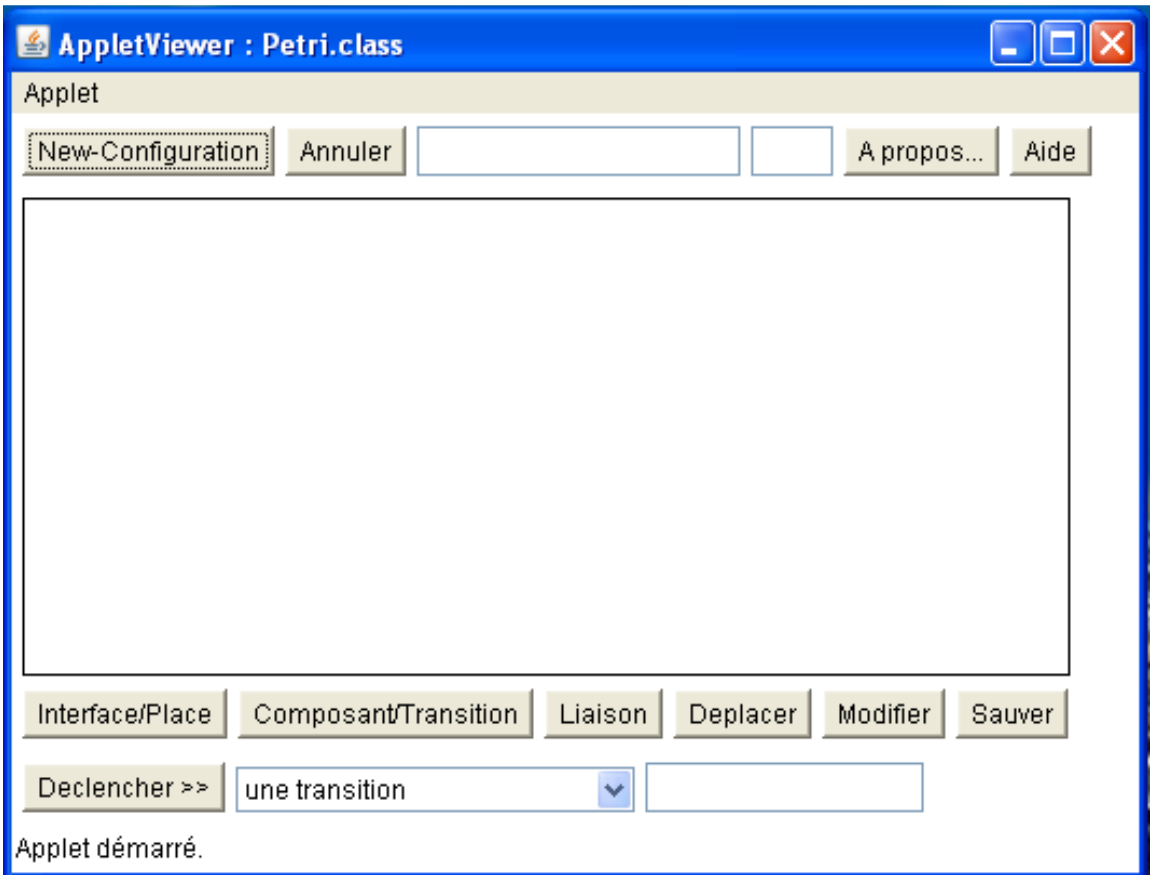

Figure 37: Interface de l'application

6.3 Exemple d'exécution

L'exemple consiste sur un agent qui fournit des services web, et chaque service web est rendu par un composant de l'agent, cette agent a un ensemble de 9 composants, qui sont cités dans le tableau suivant, chacun avec ses interfaces requises et ses interfaces fournies :

Tableau 4: La base de composants

	Composant	Entrées	Sorties
1	a	AuthorCod, Inst	PubCod
2	b	AuthorName	PubCod
3	c	PubCod	Title
4	d	PubCod	ConfCod
5	e	PubCod	ConfCod,ConfName

6	f	ConfCod	ConfName,ConfDate
7	g	Inst	AuthorCod
8	h	ConfCod	ConfPlace
9	i	AuthorCod	ConfCod

Ces composants pour être utilisables par l'algorithme doivent être modélisés par un réseau de Petri qui représente la base tel qu'il est présenté dans la figure suivante.

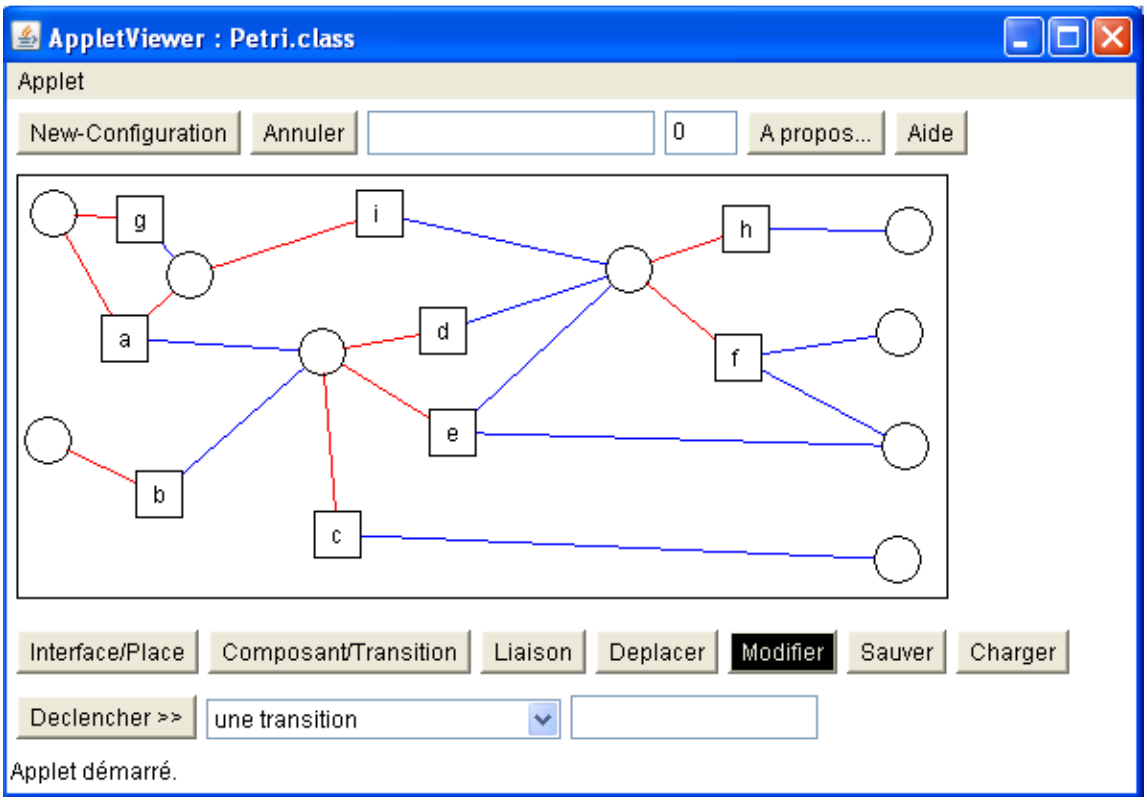

Figure 38: La base de composants

6.4 Un scénario d'exécution

Un évènement de l'environnement Ev(E,S) tel que, E={Inst} et S={ConfName,ConfDate}.

On suppose que la configuration actuelle de l'agent est comme suit {d, c, h}, tel que, d, c et h correspondent au composants présents dans la configuration courante tel qu'il est montré dans la Figure 39.

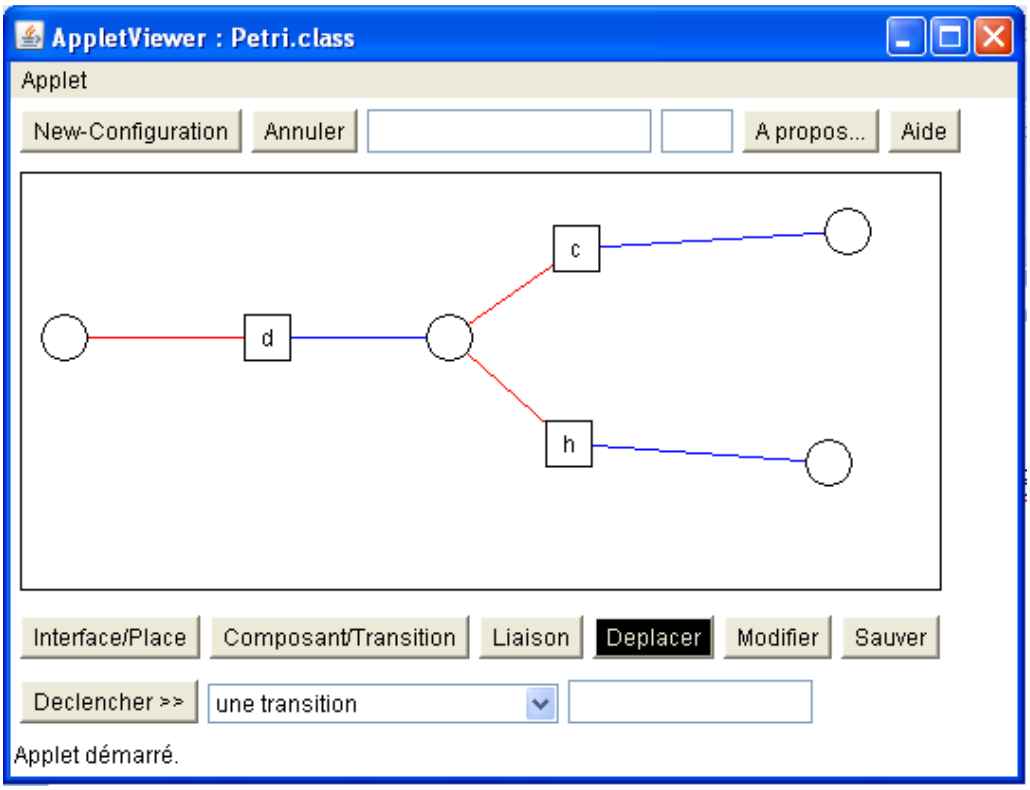

Figure 39: Une configuration de l'agent avant l'adaptation

Lorsque l'évènement Ev(E,S), est reçu par le *mécanisme d'observation de l'environnement*, celui-ci compare les deux ensembles E et S avec les interfaces requises et fournies de la configuration actuelle (Figure 39) et comme il ne trouve pas une similarité, il envoie l'évènement au générateur de comportement, qui va exécuter l'algorithme de sélection de composants pour arriver à une configuration convenable.

Cet algorithme va prendre l'ensemble E et met un jeton dans chaque place correspondante à un élément de cet ensemble. Dans ce cas c'est seulement {Inst} qui est la place d'entrée du composant g qui va avoir un jeton. Comme illustrer dans la figure suivante.

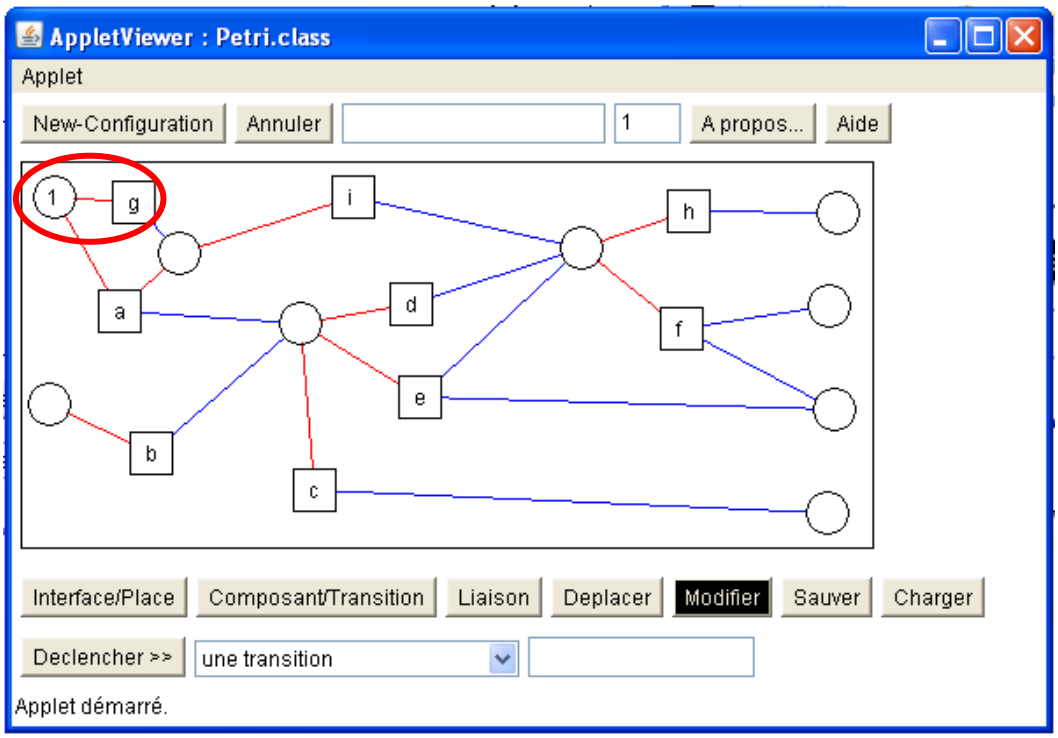

Figure 40: Exécution de l'adaptation (première étape)

La transition g est seulement franchissable, donc :

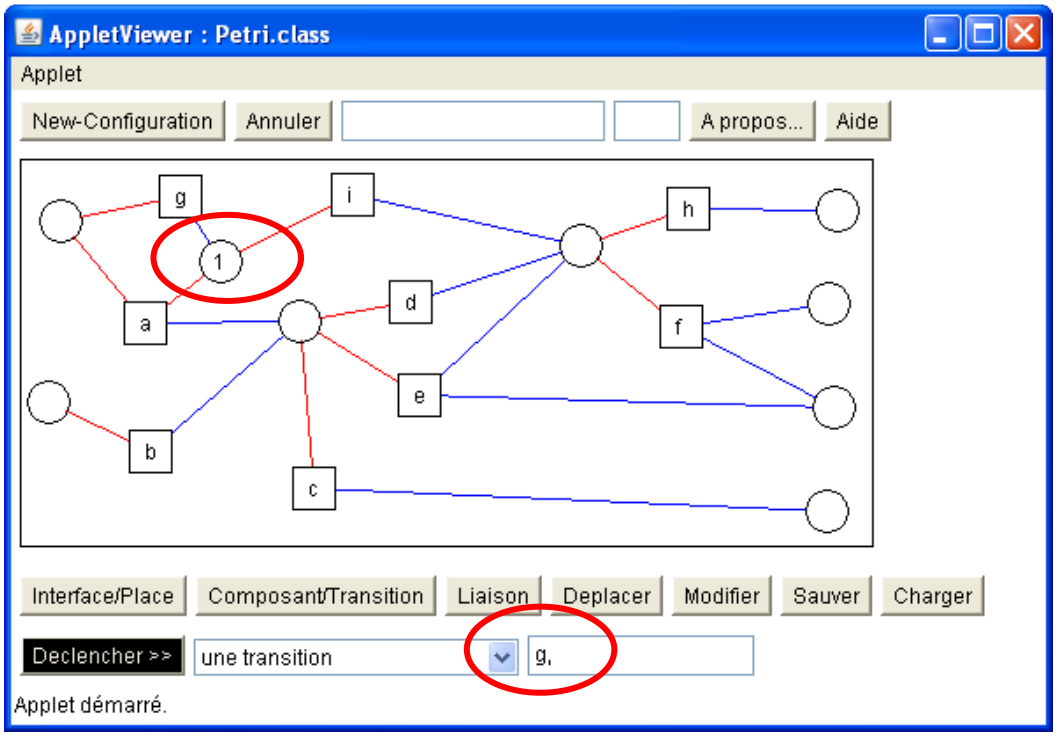

Figure 41: Exécution de l'adaptation (deuxième étape)

La transition i sera ensuite franchissable.

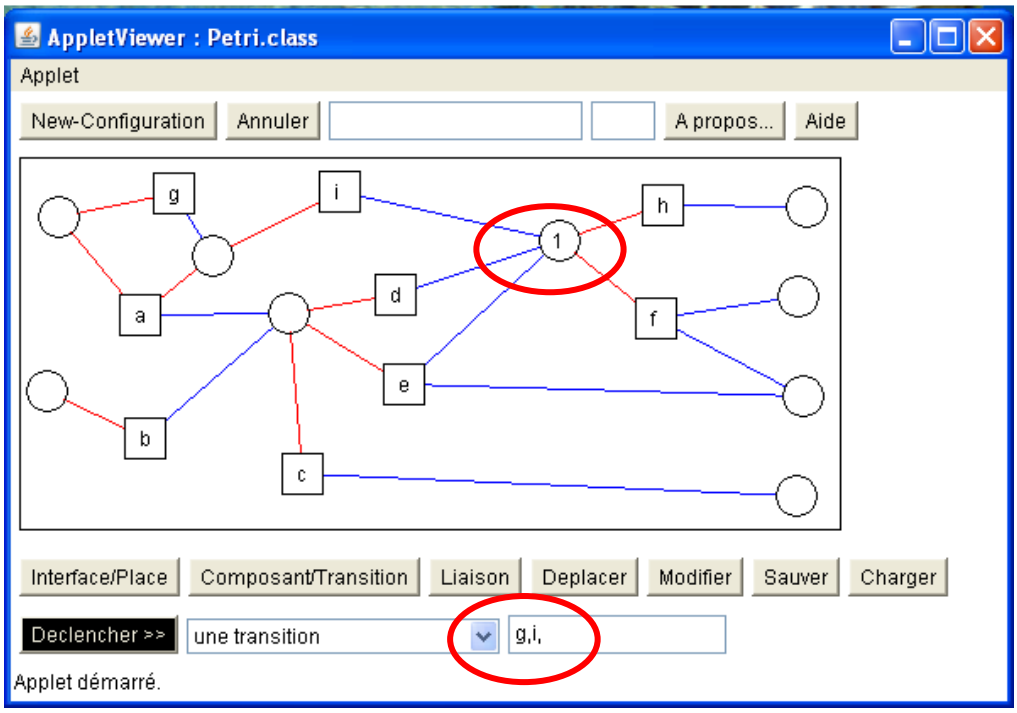

Figure 42: Exécution de l'adaptation (troisième étape)

Dans ce cas on observe qu'il y a en même temps deux transitions franchissables, et on peut franchir l'une ou l'autre indifféremment, on a déclenché f.

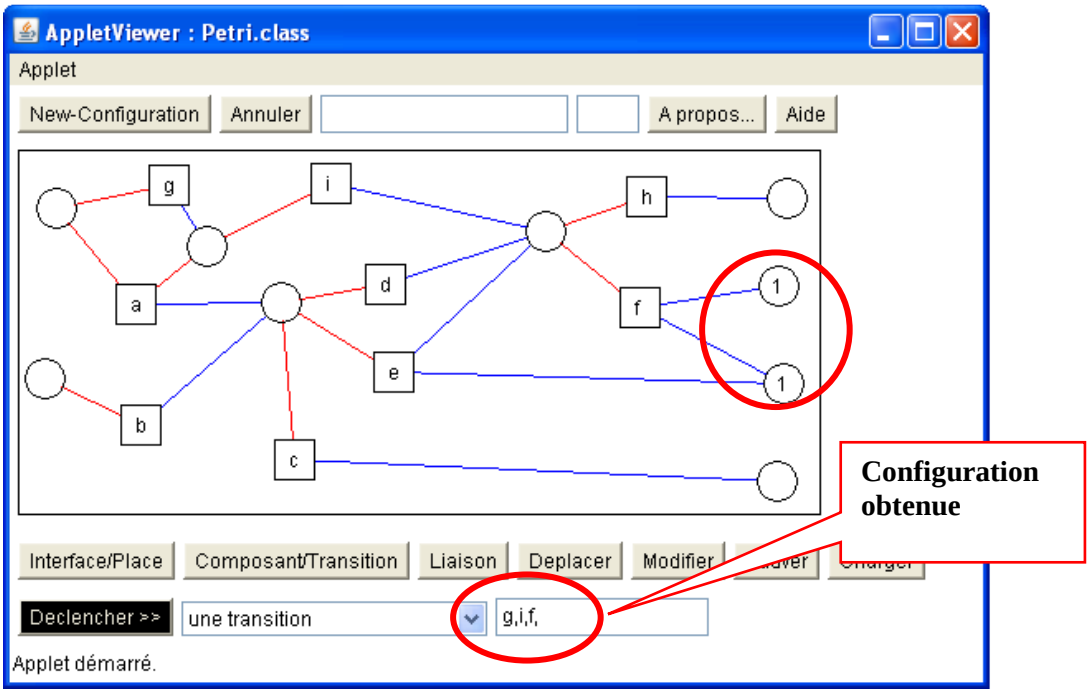

Figure 43: Exécution de l'adaptation (quatrième étape)

Après exécution de l'algorithme ce dernier va aboutir à la suite de composants suivante : ∂ = g; i; f. Qui vont être assemblés pour avoir la prochaine configuration (Voir figure 5).

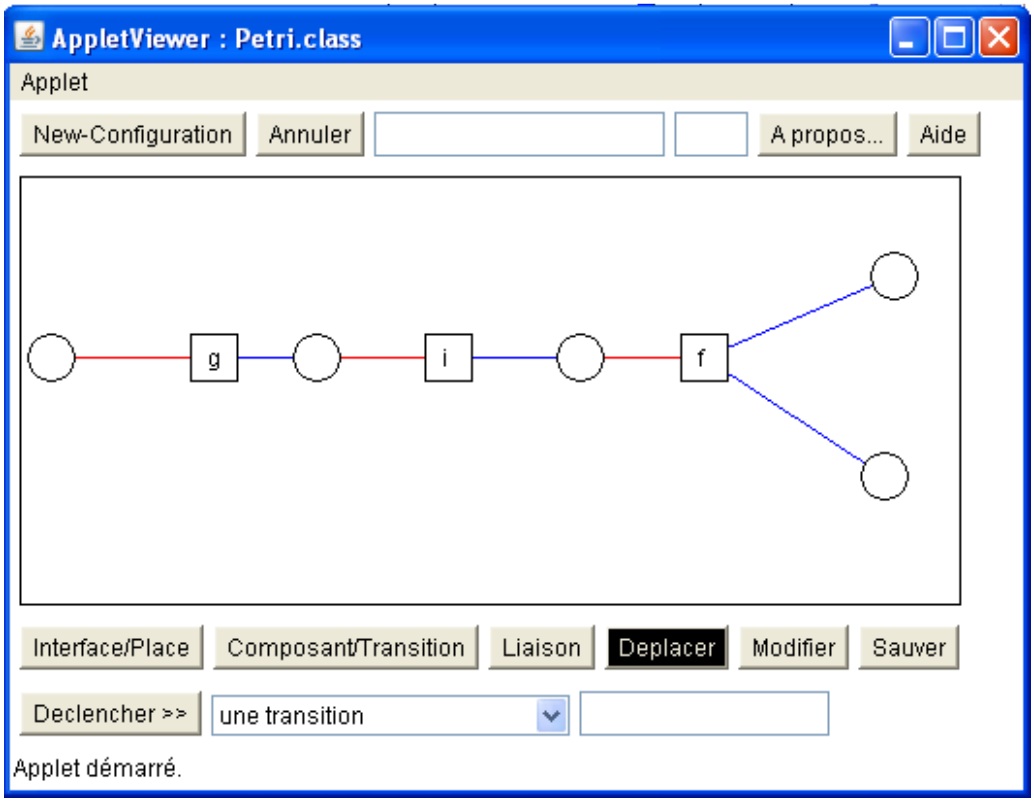

Figure 44: La configuration aprés l'adaptation

6.5 Conclusion

Dans cette partie on a donné un cas d'étude de notre approche proposée dans ce mémoire. Le cas étudié est un exemple d'un agent qui fournie des services web, en s'adaptant aux différentes formes de requêtes que peuvent formuler par les utilisateurs.

Une application est développée dans le cadre de ce travail permettant la création de réseaux de Petri et la génération automatiques de configuration.

On a fait la conception de l'agent selon notre modèle proposé de l'agent auto-adaptable à base de composants. Donc on a modélisé la base de composants comme proposer dans l'approche par un réseau de pétri, et les évènements de l'environnement sont aussi mis sous forme de deux ensemble E et S.

Un scénario d'exécution à été donné à travers lequel on a vu différent cas de comportement de l'agent.

Donc cette étude, l'exemple de services web nous a permet de valider l'approche de génération de configurations à travers les réseaux de Petri et en fonction de l'évènement qui représente l'état de l'environnement.

Dans la suite, il nous reste en premier de compléter l'application pour avoir d'autres possibilité tel que l'enregistrement des configurations dans des fichiers externes et la possibilité de la charger ainsi depuis un emplacement donné, et on doit aussi trouver une méthode de vérification de la validité de des configurations résultantes. Et cela pour être sûr du fonctionnement de notre système résultant.

Conclusion générale

Dans ce travail on a proposé un modèle d'agent auto-adaptable à base de composants, le modèle proposé est basé sur une séparation entre les tâches liée au fonctionnement habituelle assemblés dans un bloc de fonctionnement, et les tâches de supervisions et d'adaptations qu'ils sont mis dans un bloc de haut niveau dédié au comportement d'adaptation. Cette séparation a permet de facilité la conception de l'agent et la génération de nouvelles configurations du au fait que seule la partie basse (Le bloc de fonctionnement) va être touchée par modification lors des adaptations.

Nous avons présenté dans le premier chapitre l'avancement de l'utilisation des systèmes informatiques à travers un parcours depuis les premières utilisations des systèmes informatiques dans la résolution des problèmes complexes, qui commencent par des simples utilisations des petits applications de calcule, jusqu'à l'arrivée à des systèmes hautement adaptés aux besoins des utilisateurs mêmes lors de l'évolutions des ces besoins avec un changement très fréquent que nous avons vécus, et on a vu que le besoin actuel ne se limite plus à la résolution des problèmes complexes pour le profit des utilisateurs et des autres systèmes mais plutôt il s'oriente vers la prise en compte des changements au niveau de l'environnement et aussi dans les besoins des utilisateurs.

Dans le deuxième chapitre on a vu une présentation des systèmes multi agents qui constituent un paradigme prometteur et de l'actualité dans le développement de systèmes informatiques, nous avons présenté de la manière la plus claire possible ce domaine et ses notions de bases ainsi que ses relations avec les autres approches sous jacentes à savoir la programmation orientée objets et l'intelligence artificielle, et on a présenté les différences et les similarités existantes entre ceux-ci. Quelques plateformes de développement de systèmes multi agents sont présentées.

Dans le troisième chapitre nous avons étudié la modélisation des systèmes informatiques, notamment les méthodes formelles. Un cas particulier qui est les réseaux de Petri est bien détaillé, à cause de son utilisation dans notre travail. On a définit les notions liés aux réseaux de Petri, les définitions, et les différents types de systèmes peuvent être modélisé avec ceux-ci, un nombre de problèmes qui peuvent êtres résolus par les réseaux de Petri est donné pour montrer l'importance de cet outil et aussi son pouvoir d'expression, de vérification, et de validation enfin. On a constaté après cette étude l'importance et le pouvoir d'expression des réseaux de petri et ses différentes utilisations notamment pour la vérification des propriétés comportementales des systèmes avant leur implémentation, ainsi la validation formelles de modèles de systèmes à développer au cours de l'opération de conception.

Le sujet de l'adaptabilité est abordé dans le quatrième chapitre, ce sujet concerne aussi bien l'adaptabilité dans les systèmes multi agents malgré que l'adaptation est une caractéristique qui devienne généralisée pour la plupart des systèmes informatiques actuelles, du fait de leurs utilisation dans toutes les aspects de notre vie actuelle chose qui oblige leurs concepteurs à prendre en compte lors de la conception de développer des systèmes non pas seulement pour répondre aux besoins actuelles de leurs utilisateurs mais pour les accompagner dans ses différentes situations et rester performants de la même manière lors des changements de contexte. On peut citer es appareils mobiles comme exemple, qui se trouvent avec nous n'import où on se trouve, donc ils sont exposés aux changements de l'environnement, par exemple la dégradation de la qualité de réseau, la déconnexion soudaine de quelques appareils, un disfonctionnement de périphérique etc. Tous ces aspects doivent êtres prises en comte de manière efficace. Chose qui rende l'd'adaptation au contexte un point crucial et de grande importance dans le développement de systèmes de future.

Ce travail qu'à travers lequel on a proposé un modèle d'un agent auto-adaptable à base de composants entre dans ce chemin, notamment dans les systèmes multi agents que leurs utilisations deviennent de plus en plus généralisées, et aussi par leur nature qui dépend d'un environnement n'est pas souvent stable. Notre proposition s'intéresse au sujet de la génération de configurations à partir des composants disponibles dans une base de composants et de la description de changements de l'environnement exprimé sous de paramètres d'entrée et de sortie, qui permet à un agent de réagir conformément même avec des changements de conditions de son environnement.

L'agent proposé est conçu en deux niveaux, et l'approche par composants est utilisée pour concevoir chacun des blocs de l'agent en mettant une architecture de transfert d'informations et de contrôle entre les différentes parties. Cette architecture entre dans une approche qui, bien qu'elle n'est pas nouvelle, elle a connu une large utilisation qui ne cesse de s'augmenter du à ses succès en terme de réutilisation. Un autre bénéfice du développement par composants concerne le volet commercial, car il rend possible de développer des composants destinés pour des tâches connues, et avec des interfaces bien définies, et le développeur n'a que a chercher l'ensemble de composants qui convient à ces besoins et l'adapter pour se débarrasser d'une grande charge de travail de programmation.

Pour automatise la sélection de composants et la génération des configurations pour les changements rencontrés, une modélisation par réseaux de Petri a été utilisée. La base de composants a été modélisée par un réseau de Petri, cette approche consiste à ajouter tous les composants à la base. Pour la génération de configurations, le générateur de comportement, dont le fonctionnement est détaillé sous forme d'un algorithme, génère en fonction des composants disponibles et la spécification des changements de l'environnement des configurations de comportement de l'agent en parcourant le réseau de Petri de la base de composants, et en utilisant les paramètres l'évènement qui reflète l'état courant de l'environnement.

Pour valider notre approche et montrer le déroulement de l'approche suivie, un cas d'étude est montré. L'exemple de fourniture de services web dans lequel l'agent a répondu aux requêtes des clients de services web. Et on a vu à travers l'exemple le déroulement des différents algorithmes, sur lesquelles le comportement de l'agent est basé. Et cet exemple et le scénario donné nous a permettait de voir la génération de configuration à travers des réseaux de Petri et la configuration résultante.

Cette étude bien qu'elle nous a permettait de mettre en claire l'architecture de l'agent auto-adaptable, et l'automatisation de la génération de configurations, elle nous ouvre de nouvelles perspectives pour aller plus loin dans le modèle, en l'améliorant pour traiter d'autres cas plus concrets, chose qui peut nous éclairer des points pour les développer plus.

Bibliographie

[1] Hichem RAHAB, Ramdane MAAMRI, un agent auto-adaptable à base de composants, ICIST, Tebessa 24-26 April 2011. p 298-305.

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

[3] M. Wooldridge, Introduction to MultiAgent Systems. John Wiley and Sons, 2002.

[4] Politechnica University of Bucharest – 2002, Agents intelligents cours WEB intéractif http://turing.cs.pub.ro/auf2/html/chapters/chapters.html.

[5] Université LAVAL, Agents et Systèmes Multi-agents Cours IFT-64881 www.damas.ift.ulaval.ca/~coursMAS/

[6] Fabio bellifemine, Giovani Caire, Dominic Greenwood, Developing Multi agent systems with JADE. Willey series in agent technologie 2007.

[7] Tadao Murata, Petri Nets : Properties, Analysis, Applications, Proceeding of the IEEE, vol 77, NO 4, April 1989.

[8] JAMES L. PETERSON, Petri Nets, Computing Surveys, Vol 9, No. 3, Septembre 1977.

[9] L. Vercouter. MAST : Un modèle de composants pour la conception de SMA. In *Journées Multi-Agents et Composants (JMAC'04)*, pages 18–31, Paris, France, Novembre 2004.école des Mines de Paris.

[10] Alain Cardon, Zahia Guessoum. SYSTEMES MULTIAGENTS ADAPTATIFS. Proc. JFIADSMA 2000.

[11] Zahia Guessoum. Thomas Meurisse et Jean-Pierre Briot. Construction modulaire d'agents et de systèmes multi-agents adaptatifs en DIMA 2002.

[12] Gauthier Picard —Marie-Pierre Gleizes, Outils pour la réalisation de systèmes multi-agents adaptatifs dans le cadre de la méthode ADELFE, JFSMA 2003

[13] Jean-Pierre Georgé, Marie-Pierre Gleizes, Pierre Glize. Conception de systèmes adaptatifs à fonctionnalité émergente: la théorie des AMAS. Dans : Revue d'Intelligence Artificielle, Hermès Science Publications, Vol. 17, N. 4/2003, p. 591-626, 2003.

[14] Guillaume Grondin, MaDcAr-Agent : un modèle d'agents auto-adaptables à base de composants, thèse de doctorat, 2008.

[15] Guillaume GRONDING, N. Bouraqadi, L. Vercouter, Conception d'agents auto-adaptables à base de composants, 2007.

[16] Thomas Meurisse, J-P Briot: Une approche à base de composants pour la conception d'agents, 2001.

[17] L Vercouter: MAST : un modèle de composant pour la conception de SMA, 2004.

[18] Gabor Karsai, Akos Ledeczi, Janos Sztipanovits, Gabor Peceli and Gyula Simon, et al. An Approach to Self-Adaptive Software based on Supervisory Control, Lecture Notes in Computer Science, 2003, Volume 2614, Self-Adaptive Software: Applications, Pages 77-92.

[19] C courbis, P Degenne, A Fau et D Parigot, Un modèle abstrait de composants adaptables. INRIA Sophia Antipolis Project OASIS et W3C, 2004.

[20] G Grondin, N Bouraqadi, et L Vercouter: Assemblage automatique et adaptation d'application à base de composants, 2007.

[21] S. Leriche et J.-P. Arcangeli : Déploiement et adaptation de systèmes P2P: une approche à base d'agents mobiles et de composants. In Actes des Journées Multi-Agents et Composants, 2004.

[22] S. Leriche, Architectures à composants et agents pour la conception d'applications réparties adaptables. Thèse de doctorat, Université Toulouse III, 2007.

[23] S.Mansour et J.Ferber: Agent Groupe Rôle et Service: Un modèle organisationnel pour les systèmes multi-agents ouverts. JFSMA, 2007.

[24] C. Courbis, P. Degenne, A. Fau, D. Parigot : Un modèle abstrait de composants adaptables. Systèmes à composants adaptables et extensibles, 2003.

[25] Carole Bernon, Valérie Camps, Marie-Pierre Gleizes, Pierre Glize, La conception de systèmes multi-agents adaptatifs : contraintes et spécificités. Atelier de Méthodologie et Environnements pour les Systèmes Multi-Agents (SMA 2001), Plate-forme AFIA, Grenoble du 25 au 28 juin 2001.

[26] Yudith Cardinalea, Joyce El Haddadb, Maude Manouvrierb,c, Marta Rukozb,c,d: Web Service Selection for Transactional Composition, 2010.

[27] ZERNADJI Tarek, Une approche de modélisation des logiciels à base de composants par les réseaux de Petri, mémoire de magistère, université de Batna, présenté le 14/06/2009.

[28] Thomas LEDOUX, État de l'art sur l'adaptabilité, Ecole des Mines de Nantes, 12 décembre, 2001.

ملخـــــــص:

تتعرض الأنظمة المعلوماتية الحالية لوضعيات متغيرة، من الصعب التنبؤ بها مسبقا، من أجل وضع حلول مناسبة، خاصة **الأنظمة متعددة الأفراد** المتميزة بخاصة الاستقلالية، لأن **الأفراد** يتنقلون من موقع لآخر حيث يواجهون وضعيات مستجدة يجب التكيف معها.

تكييف تصرف **الفرد** مع محيطه يجب أن يتم بصفة داخلية، وهذا لتقليل تدخل المصمم والمحافظة على استقلالية الفرد، لكن من جهة أخرى هذه العملية تحتاج إلى نوع من التحكم اجتنابا لعمليات تكييف عشوائية قد تنتج تصرفات غير مرغوب فيها.

في إطار هذا العمل قمنا باقتراح نموذج لفرد ذاتي التكيف قائم على نموذج الأجزاء، ومصمم وفق منهجية ذات مستويين، في النموذج المقترح ميزنا بين التصرف العادي للفرد مجسدا بمجمع للتنفيذ والذي صورة لفرد تقليدي، والتصرف التكيفي المجسد بواسطة مجمع علوي للإشراف والتكيف، هذا الأخير هو مكلف بضمان تأدية الفرد في أحسن صورة عن طريق تزويده بالتصرفات المناسبة لمختلف الوضعيات التي قد تصادفه.

عملية التكيف تمت صياغتها بواسطة شبكات بتري للسماح بإنتاج بنيات جديدة بطريقة تلقائية، انطلاقا من الأجزاء المتوفرة في قاعدة الأجزاء، وباتباع وصف حالة المحيط.

الكلمات المفتاحية: انتقاء، إعادة الهيكلة، فرد، التكيف التلقائي، الجزء، التصرف.

Abstract :

Actual software systems are more and more exposed to new situations, which in difficult to all predict, and prepare an adequate processing, especially multi agents systems which are characterized by the autonomy property, because of their movements from site to another, when it finds new conditions to be adapt with.

Agents behavior adaptation with their environment must be an interior characteristic, and thus to minimize designer interventions and to preserve agent autonomy, but in another part it must have a manner of control, to don't come to random adaptations, and inadequate behavior.

In this work we have proposed a model of component based self adaptive agent, designed following a two levels approach. In pour model we have distinguished the agent normal behavior, materialized by a functional block which is a kind of classical agent, from its adaptation function realized by a hi level of adaptation and supervision, which the vocation is to ensure agent correct behavior by endowing it by good behaviors, directed to several situations it can meet.

The adaptation process have been modeled by a Petri net allowing automation of new configurations generation process from available components in the base, and following the environment changes description.

Key words : selection ; reconfiguration ; agent ; self-adaptation ; component; behavior.

Résumé

Les systèmes logiciels actuels sont de plus en plus exposés à de nouvelles situations, qui est difficile de tous prévoir à l'avance et de préparer un traitement convenable, notamment les systèmes multi agents qui sont caractérisés par sont autonomie, puisque ses agents vont déplacés d'un site à un autre où ils trouvent de nouvelles conditions avec lesquelles ils doivent s'adapter.

L'adaptation de comportement des agents avec sont environnement doit être une caractéristique interne, et cela pour minimiser l'intervention du concepteur et préserver l'autonomie de l'agent, mais d'un autre coté elle doit avoir une façon de contrôle pour ne pas arriver à des adaptations aléatoires et des comportements qui peuvent être inadéquats.

Dans ce travail nous avons proposé un modèle d'un agent auto-adaptable à base de composants, et conçu en suivant une approche à deux niveaux. Dans notre modèle on a distingué le comportement normal de l'agent matérialisé par un bloc de fonctionnement qui est une sorte d'agent classique, et le fonctionnement d'adaptation réalisé par un bloc supérieur d'adaptation et de supervision, ce dernier est chargé d'assurer le bon fonctionnement de l'agent en le dotant des bons comportements qui sont destinés aux différentes situations qu'il peut rencontrer.

L'opération d'adaptation a été modélisée par un réseau de Petri permettant d'automatiser le processus de génération de nouvelles configurations à partir des composants disponibles dans la base, et en suivant la description du changement de l'environnement.

Mots clés : sélection ; reconfiguration ; agent ; auto-adaptation ; composant; comportement.