République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de
la Recherche Scientifique

Université Hadj Lakhdar de Batna
Faculté des Sciences de l'Ingénieur
Département de l'Informatique

Thèse de Magister en Sciences de l'Informatique
Option : Génie Logiciel et Intelligence Artificielle

Par

Ahmed TLILI

Etude et Réalisation d'une Plate Forme Multi-Agents

Directeur de Thèse
Dr Kamel KHOUALDI

Devant le Jury :

Président	**Pr Mohamed Benmohamed**	**Université Mentouri de Constantine**
Rapporteur	**Dr Kamel Khoualdi**	**Université Hadj Lakhdar de Batna**
Examinateur	**Dr Allaoua Chaoui**	**Université Mentouri de Constantine**
Examinateur	**Dr Brahim Belatar**	**Université Hadj Lakhdar de Batna**
Examinateur	**Dr Azzedine Bellami**	**Université Hadj Lakhdar de Batna**

TABLE DES MATIERES :

Chapitre I

Problématique et objectif du travail

1 Présentation du domaine de recherche :

L'Intelligence Artificielle classique (IA) se contente en effet de construire des programmes informatiques, capables d'exécuter des taches complexes, en s'appuyant sur une centralisation et concentration de « l'Intelligence » ou du « savoir faire » au sein d'un même et unique système qu'on appelait Systèmes Expert (SE). Mais il en résulte un certain nombre de difficultés, dues pour l'essentiel à la nécessité d'intégrer, au sein d'une seule base de connaissance, l'expertise, les connaissances et les compétences d'individus qui, dans la réalité, communiquent et collaborent pour l'accomplissement d'une tâche commune.

A la différence de l'Intelligence Artificielle classique qui modélise donc le comportement intelligent d'un agent sous le pseudonyme de système expert, l'intelligence Artificielle Distribuée (IAD), née dans les années 1970 et dont le but est de trouver des parades aux difficultés rencontrer par l'IA, s'intéresse à des comportements intelligents qui sont le produit de l'activité **coopérative** de plusieurs agents, autrement dit à des systèmes multi-Experts.

Aujourd'hui l'IAD est un champ scientifique qui rassemble plusieurs disciplines incluant l'Intelligence Artificielle, la sociologie, l'économie, l'organisation, la planification …etc. Cette pluridisciplinarité rend la tâche de définir l'IAD un peu difficile. Ferber [6] a donné une définition qui, approximativement, rend compte de l'IAD actuelle « l'IAD est l'étude, la conception et la réalisation des Systèmes Mutli-Agents (SMA), c'est à dire de systèmes dans lesquels des agents intelligents et qui interagissent, poursuivent un ensemble de buts ou réalisent un ensemble d'actions ».

Le concept d'intelligence symbolise ici la capacité pour un agent de mené à bien un comportement coopératif, rationnel, dynamique et souple dans un environnement bougeant et souvent non déterministe.

L'Intelligence Artificielle Distribuée consiste à élaborer des systèmes constitués d'un groupe d'agents plus ou moins autonomes, chacun étant doté d'une certaine autonomie et devant être capable de planifier, d'agir et de travailler dans un environnement commun tout en prévoyant de gérer les situations conflictuelles éventuelles.

L'IAD conduit ainsi à la réalisation de systèmes dits « Multi-Agent », qui permettent de modéliser le comportement d'un ensemble d'entité (ou agents) plus ou moins expertes, plus ou moins organisées selon des lois de type social. Ces entités (ou agents) dispose d'une certaine autonomie, et sont immergés dans un environnement dans lequel et avec lequel elles interagissent.

2 Objectif de notre travaille :

La simulation orientée agents est aujourd'hui utilisée dans un nombre croissant de secteur, où elle remplace progressivement les anciennes techniques connues comme la simulation orientée objet, ceci est dû, pour une part, à sa capacité à appréhender des modèles très différents d'individu, depuis des entités très simples comme les agents réactifs jusqu'aux entités plus complexe comme les agents cognitifs. Ce succès est cependant

ambiguë; Malgré que la plupart des chercheurs semblent s'accorder sur une terminologie commune pour designer les concepts centraux des systèmes Multi-Agents utilisés en simulation orientée agents, il apparaît de plus en plus que cet accord est au niveau syntaxique seulement, la sémantique associée diffère considérablement d'un modèle à un autre. Autrement dit, les Agents définis dans la conception diffèrent des agents implémentés en réalité, pourquoi ? Parce qu'il n y a pas de consensus autour d'une approche orientée agent, allant de la phase d'analyse passant par la phase de conception et ce termine par la phase d'implémentation sur une plate forme multi-agents qui regroupe les différents modèles d'agents connus dans la littérature de la même manière que l'on parle d'approche orientée objets.

Néanmoins, actuellement le besoin d'une telle approche est pressenti par la plupart des équipes de chercheurs qui travaillent dans ce domaine et notre travail s'inscrit dans cette ligné mais limité à l'étude et la réalisation conceptuelle d'une plate forme multi-agents.

Les plates formes Multi-Agents constituent donc notre centre d'intérêt et notre objectif consiste a concevoir une plate forme où la notion d'agent est défini en tant que paradigme de programmation, de la même façon que l'approche orientée objet, et pas un paradigme de construction d'application comme l'utilise beaucoup de plates formes multi-agent. Cette plate forme, que nous voulons être une architecture ouverte et générique, permettra aux utilisateurs concepteurs de définir leurs propres systèmes multi-agents correspondant au domaine d'application en vu de modélisé.

Pour des raisons scientifique, notre projet se veut une extension et une prolongation naturel du paradigme objet, à ceci prêt l'idée essentiel consiste à prendre l'objet tel qu'il est définit par l'approche orientée objet et de voir comment peut on le transformer en un agent.

3 Organisation de la thèse :

Nous entamons notre travail par le premier chapitre où nous présentons la problématique et l'objectif de notre travail, nous entamons ensuite le chapitre deux sur l'Intelligence Artificielle Distribuée IAD et les Systèmes Multi-Agents où les notions de base et les différents concepts sont exposés, on présente un bref historique traçant l'évolution du domaine de l'Intelligence Artificielle classique en premier lieu, en second lieu les idées de bases qui ont conduit à l'IAD sont mises en lumière, enfin l'accent est mis sur l'une des disciplines de l'IAD, à savoir les Systèmes Multi-Agents et leurs apport dans le domaine informatique.

Dans le troisième chapitre intitulé « état de l'art des plates formes Multi-Agents », nous présenterons, à titre d'exemple, l'architecture de deux plate forme multi-agents : la plate forme MadKit pour Multi-Agent Développement KIT et la plate forme OpenCybel.

Le quatrième chapitre est consacré aux modèles organisationnels des SMAs, notion centrale dans le développement d'architecture multi-agents

Le cinquième chapitre, intitulé « Conception de la Plate forme « NewObject » pour Nouveau Objet, représente le fond de notre travail, il commence par bref rappel des concept de l'approche orientée objet puis se lance dans la construction proprement dite de la plate forme brique par brique.

Le sixième chapitre est un tour d'horizon des méthodologies de conception de systèmes multi-agents, car on ne peut parler de paradigme de programmation sans lui associe des méthodes adéquates de conceptions et de modélisations.

Enfin, nous terminons notre exposé par la partie conclusions et perspectives où l'on mentionne les possibles extensions de notre approche, sur le plan théorique que pratique, par l'introduisant des actes de langages pour améliorer la communication entre agents, ainsi que les possibilités d'une éventuelle implémentation de cette plate forme, en Java, qui offrirai un environnement de programmation et d'exécution dans un contexte proprement multi-agents.

Dans la bibliographie, on trouve les différents articles et documents, qui nous ont inspirés et guidés le long de notre travail.

En Annexe, on trouvera plus de détaille sur certains concepts mentionnés dans la thèse; et qu'on juge, nécessaire leurs approfondissements.

Mots clés : Approche Orientée Objet, Système Multi-Agent, Plate Forme Multi-Agent, Intelligence Artificielle Distribuée (IAD).

Chapitre II

Intelligence Artificielle Distribuée
Et
Systèmes Multi-Agents

Résumé : ***l'évolution*** *des domaines d'application de l'Intelligence Artificielle Classique dans la modélisation des domaines complexes et hétérogènes tels que l'aide à la décision, la reconnaissance des formes …etc, a montré les limites de l'IA classique qui s'attache à la centralisation et à la concentration de l'expertise au sein d'un seul et même système. Les travaux réalisés aux débuts des années 1970 sur la concurrence et la distribution, ont contribués à la naissance d'une nouvelle bronche, à savoir l'Intelligence Artificielle Distribuée (IAD). L'objectif initiale de l'IAD était de remédier aux défaillances de l'approche IA classique, en offrant la distribution de l'expertise sur un groupe d'agents, suffisamment autonome,s pour pouvoirs êtres capables de travailler et d'agir dans un environnement commun et résoudre les conflits éventuels. Par la suite on baptiser ces systèmes par Systèmes Multi-Agents.*

1 Introduction

L'évolution des domaines d'application de l'Intelligence Artificielle classique (IA) vers des domaines aussi complexes et hétérogènes tels que l'aide à la décision, la reconnaissance des formes, la conduite des processus industriels, etc., a montré les limites de cette approche ; celle-ci s'appuie sur une centralisation de l'expertise au sein d'un système unique (un seul agent). Ainsi, les besoins en parallélisme et distribution ont été très tôt ressentis et les recherches sur la concurrence[1] et la distribution ont contribués à la naissance d'une nouvelle discipline : l'Intelligence Artificielle Distribuée (IAD). A la différence de l'Intelligence Artificielle classique qui modélise le comportement intelligent d'un seul agent, l'intelligence artificielle distribuée (IAD) s'intéresse à des comportements intelligents qui sont le produit de l'activité coopérative de plusieurs agents.

Le passage du comportement individuel au comportement collectif est considéré non seulement comme une extension mais aussi comme un enrichissement de l'IA, d'où émergent de nouvelles propriétés et de nouvelles disciplines en l'occurrence, les Systèmes Multi-Agents (SMA).

L'IAD et les SMA étudient la manière de répartir un problème - pour sa résolution - sur un certain nombre d'entités (ou agents) autonomes, qui coopèrent et interagissent dans un environnement commun. Elles étudient aussi la manière de coordonner leur comportement selon des lois sociales. L'aspect collectif de ces agents nécessite l'étude de nouveaux concepts et de nouvelles théories qui émergent d'une société d'agents.

Dans ce chapitre, nous commençons par introduire l'IAD et présentons quelques modèles de base de ses systèmes, ensuite nous mettons l'accent sur l'évolution de cette discipline vers les SMA et nous décrivons les propriétés et lois qui émergent en passant de l'aspect individuel (le comportement d'un agent) à l'aspect collectif (son comportement dans une société d'agents).

2 L'intelligence artificielle distribuée

2.1 Historique

En 1973, le premier système d'IAD a pu voir le jour [10]. Il s'agit du système HEARSAY pour la reconnaissance de la parole, basé sur l'architecture de blackboard.

Dans les recherches qui ont suivi, diverses approches liées à la distribution se sont succédées, notamment : les acteurs de Hewitt, la société de l'esprit de Minsky, le système DVMT de Lesser, le Contract Net de Smith, le système Mace de Gasser et les micro-robots de Brooks [10].

Dans la suite de cette section, nous présentons quelques modèles qui ont marqué l'histoire de l'IAD.

2.1.1 Modèle Acteur

Le modèle acteur est développé au MIT par l'équipe de C. Hewitt[2] [12]. On lui doit un bon nombre d'aspect repris en IAD. Ce sont les travaux sur ce modèle qui ont abouti à une nouvelle conception des systèmes d'IAD. A l'origine, ce modèle a été conçu pour résoudre des problèmes d'IA. Il a abouti à une conception des systèmes

[1] La concurrence fait référence au déroulement logique simultané d'actions par opposition au parallélisme qui est le déroulement physique simultané d'actions.

[2] Devenue depuis, le Message Passing Semantics Group.

distribués, en l'occurrence, les systèmes ouverts [5]. Le modèle acteur est un modèle sous-jacent aux systèmes ouverts.

Les systèmes d'acteurs sont caractérisés par la communication par envoi de messages et un traitement local : un agent (acteur) détient des informations locales et son comportement est indépendant de celui des autres. Ils présentent une distribution à la fois des connaissances, des résultats partiels et des méthodes utilisées pour la résolution du problème. En revanche, le comportement global d'un système d'acteurs est difficile à analyser. De même, les langages d'acteurs restent peu diffusés et sont loin d'être des produits finis. Les travaux menés dans ce sens tiennent plus de la conception de langages que de systèmes de résolution de problèmes. Malgré cet aspect limité, les systèmes d'acteurs restent les piliers et points de départ des principes de l'IAD.

2.1.2 Blackboard

Le modèle de *blackboard* [10] définit une architecture qui organise la résolution de problèmes par coopération de plusieurs modules, appelés sources de connaissances (*Knowledge Source* '*KS*') autour d'une base de données partagée appelée blackboard. Cet aspect centralisé donne une vision globale du problème. Chaque *KS* vient lire ou écrire sur le blackboard. D'une façon générale, le rôle d'une *KS* est de résoudre un sous-problème particulier en fonction de l'état du blackboard. Ce modèle vise la mise en œuvre d'une résolution opportuniste, où le choix de la *KS* à activer est déterminé en fonction des critères de contrôle actifs. Ce modèle est repris dans beaucoup d'autres travaux et d'autres domaines.

2.1.3 Contract Net

Le *Contract Net* [12] est un système de résolution de problème distribué, conçu par R. Davis et R. Smith. L'objectif principal de ce système étant la distribution, il procède par allocation des tâches à un ensemble de résolveurs de problèmes et utilise le concept de négociation pour adjuger des contrats.

Au départ, un problème est posé à un agent appelé *manager*. A travers la compréhension qu'il a de ce problème et de ces capacités, il délègue une certaine quantité de tâches. Pour ce faire, il lance un appel d'offre (sous-problème à résoudre) auprès des autres agents, les *contractors*. S'il accepte de résoudre le problème qui lui est proposé, un 'contractant' effectue une offre (proposition) au manager précisant ses capacités. Le manager sera ainsi libre de choisir l'agent responsable de la résolution.

Un contractant peut lui-même déléguer certaines tâches (en utilisant le même appel d'offre). L'architecture de base contient des nœuds ayant des rôles de chef et d'exécutants.

2.1.4 DVMT

Distributed Vehicle Monitoring Testbed [12] est un système d'analyse de trafic routier par synthèse des observations d'un certain nombre de capteurs. Ce système fournit un exemple d'application dans lequel la perception et la connaissance sont distribuées. DVMT utilise une collection de systèmes identiques à base de blackboard pour résoudre des problèmes d'interprétation des données à partir d'un ensemble de capteurs séparés géographiquement et couvrant une région. Les données sont acheminées d'un sous-ensemble de capteurs à destination de chaque nœud participant à la résolution du problème. Les nœuds coopèrent pour construire une image globale du trafic routier.

2.2 L'intelligence artificielle distribuée

L'IAD propose la distribution de l'expertise sur un groupe d'agents devant être capables de travailler, d'agir et de s'organiser pour arriver à résoudre un problème posé, dans un environnement commun et résoudre les conflits éventuels. Cette distribution de l'expertise peut être géographique et/ou fonctionnelle. D'où la naissance de notions nouvelles en IA, telles que la coopération, la coordination d'actions, la négociation et l'émergence [12].

La distribution de l'expertise fait que chaque agent dispose d'une connaissance parcellaire, qui ne requiert qu'un mécanisme de raisonnement simplifié. Cette simplification est acquittée par la complication des interactions entre agents, qu'il s'agisse de déterminer les structures de communication ou de définir la connaissance commune dont ils disposent.

L'IAD peut alors être définie comme étant la branche de l'IA qui s'intéresse à la modélisation de comportement 'intelligent' par la coopération entre un ensemble d'agents. Nous distinguons ainsi trois axes fondamentaux dans la recherche [10] :

- Les systèmes multi-agents (SMA) : Il s'agit de faire coopérer un ensemble d'agents dotés d'un comportement intelligent et de coordonner leurs buts et leurs plans d'actions pour la résolution d'un problème.

- La résolution distribuée des problèmes (RDP) : Elle s'intéresse à la manière de diviser un problème particulier et la connaissance relative sur un ensemble d'entités distribuées et coopérantes et d'en obtenir la solution. Cette décomposition devrait parvenir à une réduction de la complexité d'une tâche.

- L'Intelligence artificielle parallèle (IAP) : Elle concerne le développement de langages et d'algorithmes parallèles pour l'IAD. L'IAP vise l'amélioration des performances des systèmes d'intelligence artificielle sans, toutefois, s'intéresser à la nature du raisonnement ou au comportement intelligent d'un groupe d'agents. Cependant, il est vrai que le développement de langages concurrents et d'architectures parallèles peut avoir un impact important sur les systèmes d'IAD.

Pour le cas de notre étude, il est évident que le deuxième point est celui qui nous intéresse. En effet, nous nous intéressons à trouver une façon de décomposer le problème de surveillance de procédés en un ensemble de sous problèmes, dont la résolution devrait être simplifiée. Les SMA représentent, par contre un aspect que nous exploiterons avec simple adaptation au domaine.

2.3 Problématique de l'IAD

Les problèmes que l'IAD s'attache à résoudre, sont les problèmes classiques de l'IA qui ont pris une nouvelle dimension dans le contexte multi-agents et les nouveaux problèmes proprement liés au thème de l'IAD, on peut citer [10] :

- La modélisation de la connaissance et le problème de sa répartition sur les différents agents regroupés en des sociétés : comment formuler, décomposer, allouer des problèmes et synthétiser les résultats d'un groupe d'agents ?

- Les problèmes de génération de plans d'actions où il faut prendre en considération la présence d'autres agents. Ces problèmes sont liés au comportement d'un agent au sein d'un groupe. On s'intéresse alors à ses capacités sociales : la répartition des tâches, le partage des ressources, le raisonnement sur les autres agents (pouvoir modéliser leurs connaissances et être en mesure de connaître leurs plans d'actions et de raisonner en fonction de ces plans) ;

- La gestion des conflits entre les agents et le maintien de la cohérence des décisions et des plans d'actions ;

- Le problème de la communication : comment permettre la communication et l'interaction entre les agents ? Quel langage et quel protocole faut-il employer ? Une communication dans les univers multi-agents n'est plus une simple tâche d'entrée-sortie, mais doit être modélisée comme un acte pouvant influer sur l'état des autres agents.

- Les problèmes spécifiques au groupe d'agents, qui portent sur l'organisation, l'architecture de l'ensemble des agents et les paradigmes de coopération et d'action ;

- D'autres thèmes de recherche sont présents dans le contexte multi-agents, à savoir, le raisonnement temporel, le raisonnement hypothétique, la représentation de la connaissance imprécise, etc.

3 Les Systèmes Mutli-Agents

3.1 Introduction

Un modèle est une représentation simplifiée de la réalité. Les simplifications apportées dépendent de la problématique du modélisateur. Il existe plusieurs classifications des modèles selon des axes d'opposition différents [6] :

- les modèles statiques contre les modèles dynamiques ;

- les modèles prédictifs ou empiriques contre les modèles explicatifs et descriptifs ;

- les modèles déterministes contre les modèles stochastiques ;

- les modèles qui peuvent être résolus analytiquement contre les modèles qui nécessitent des

- simulations numériques.

Dans le cas d'études de systèmes hétérogènes ou complexes, une démarche consiste à analyser le problème et le représenter avec des processus ou des objets indépendants mais en interaction. C'est une des composantes des systèmes multi-agents. Une telle modélisation est souvent accompagnée par des séries de simulations qui

permettent d'évaluer le modèle par rapport à son objectif. La modélisation alliée à des simulations constitue une démarche de recherche au même titre que la recherche théorique ou l'expérimentation pure . On parle pour cette recherche par des simulations informatiques, de recherche *in silico* [14]. Les systèmes multi-agents sont d'abord utilisés pour la modélisation de systèmes complexes et conduisent la plupart du temps à des séries de simulations exploratoires.

On distingue les Systèmes Multi-Agents (SMA) informatiques des systèmes multi-agents avec une réalité spatialisée comme les systèmes de robots, seuls les premiers sont présentés dans cette partie.

3.2 Historique

Les systèmes multi-agent sont apparus au carrefour des recherches sur l'intelligence artificielle distribuée et sur la vie artificielle. Ces systèmes sont développés à partir de schémas de raisonnement ou d'organisations empruntés aux domaines de la vie et de la société [6].

3.2.1 L'intelligence artificielle

L'objectif de l'intelligence artificielle est d'étudier les modes de raisonnement à partir de systèmes virtuels. Ces systèmes sont capables de résoudre un problème en utilisant des symboles, c'est à dire un langage simplifié. Dans un premier temps, l'idée était de distribuer l'intelligence ou la connaissance en utilisant une assemblée de spécialistes virtuels. Les spécialistes se concertaient par le biais d'un « tableau noir » [10]. Ce tableau noir était l'espace commun de mémoire. Chaque spécialiste pouvait y déposer, modifier et effacer des données. Au bout d'un moment, sur le tableau, la solution devait émerger des actions des différents spécialistes. Une autre méthode simulée était l'élaboration par l'assemblée des spécialiste le spécialiste possédant toutes les connaissances pour résoudre, à lui seul, le problème posé. Hewitt [12] en 1991 remarqua alors l'importance du contrôle des interventions des spécialistes. Il s'orienta vers un contrôle distribué et non plus celui d'un choix séquentiel. Il développa ce contrôle distribué par l'envoi de messages entre les différents acteurs. Ce fut la base du langage d'acteur.

3.2.2 La vie artificielle

L'objectif des recherches sur la vie artificielle est de comprendre les principes qui gouvernent les systèmes vivants. Pour cela, ces études modélisent différents principes afin d'aisément les tester de façon artificielle. Epstein [15], en 1988, a posé la problématique de la façon suivante : la vie artificielle est « l'étude de la vie telle qu'elle pourrait être et non de la vie telle qu'elle est. »

Le développement de ces modèles s'appuie sur les concepts d'autonomie, de comportement individuel répondant à des stimuli de l'environnement, de viabilité, d'adaptation et de reproduction. Il n'y a pas de processus de réflexion comme en intelligence artificielle par la manipulation de symboles. Les agents n'ont pas des intelligences évoluées mais des règles de comportement simples. Cependant, il est possible d'obtenir un comportement collectif complexe par l'interaction de plusieurs agents relativement simples.

3.2.3 Les premiers systèmes muti-agents

D'après Ferber [6], deux systèmes ont marqué le développement des systèmes multi-agent :

1- Le modèle DVMT (Distributed Vehicule Monitoring Test) permettait d'obtenir une image du trafic routier. Il traitait les informations transmises par plusieurs capteurs. Celles ci pouvaient être contradictoires, redondantes ou bruitées. Ce modèle a permis d'étudier notamment les protocoles de coopération, de négociation et de planification dans les systèmes multi-agent [6].

2- Le système MACE [13] a démontré que la communication n'est pas suffisante si elle n'est pas alliée à une représentation de son environnement par l'agent.

3.3 Les systèmes multi-agent

3.3.1 Définitions

Les systèmes multi-agent constituent une nouvelle technique de modélisation qui place l'objet d'étude au centre de sa démarche. Ces modèles représentent les actions individuelles, les interactions entre les acteurs et les conséquences de ces interactions sur la dynamique du système [5].

Les systèmes multi-agent empruntent à l'intelligence artificielle distribuée les modes de communication et de concertation entre agents. Ils reprennent les idées d'autonomie et d'émergence du résultat final à partir des interactions individuelles à la vie artificielle. La distribution des processus ou des connaissances au niveau d'individus actifs se retrouve dans les deux domaines présentés ci-dessus.

Ferber [6] (1995) donne la définition suivante :

« un système multi-agent est un système composé des éléments suivants:

- un **environnement** c'est à dire un espace disposant généralement d'une métrique,
- un ensemble d**'objets** situés dans l'espace, ils sont passifs, ils peuvent être perçus, détruits, créés et modifiés par les agents,
- un ensemble d**'agents** qui sont les entités actives du système,
- un ensemble de **relations** qui unissent les objets entre eux,
- un ensemble d**'opérations** permettant aux agents de percevoir, de détruire, de créer, de transformer et de manipuler les objets,
- un ensemble d'opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification (les lois de l'univers) ».

Cette définition propose une description générale de la structure d'un système multi-agent. Cependant, un système de gestion du temps comportant plusieurs objets et agents et utilisant des modes de fonctionnements distribués ne serait pas un système multi-agent selon la définition précédente, puisqu'il n'a pas d'environnement. Les idées directrices de conception d'un système multi-agent doivent être dissociées des

domaines d'application. Souvent dans ces applications, les agents représentent des êtres vivants. Cette proximité peut conduire à une généralisation à tous les systèmes multi-agent de principes spécifiques aux sociétés tels que la communication ou la reproduction. La définition proposée ici essaie de décrire les principes de la philosophie multi-agent sans que les domaines d'applications puissent être limitant.

Un système multi-agent est un ensemble d'entités autonomes actives (les agents). Les phénomènes ou les comportements sont distribués au niveau individuel des agents. Chacun est alors spécialisé et agit de façon autonome. De ces actions individuelles, émerge la solution ou le comportement général du système, soit par une interaction due à la modification par les agents de l'environnement où ils évoluent, soit par une communication directe entre agents par le biais d'un langage symbolique, par exemple.

Cependant les SMAs peuvent se distinguer les uns des autres au niveau des agents eux-même, des interactions entre les agents et des environnements dans lesquels évoluent les agents. Le tableau suivant donne une idée des différents types de SMAs en fonction des différentes valeurs des caractéristiques ***:***

L'élément de différenciation	Attribut	***Valeurs***
Agents	***Nombre***	***Deux et plus***
	Uniformité	***Homogène ou hétérogène***
	Buts	***Contradictoires ou complémentaires***
	Architecture	***Réactive ou délibérative***
	Compétences	***Simple ou avancée***
Interaction	***Fréquence***	***Faible ou élevée***
	Persistance	***Court terme ou long terme***
	Niveau	***Envoi de signale ou manipulation de connaissances***
	Contrôle et flux de données	***Décentralisé ou centralisé***
	Propos	***Compétitive ou coopérative***
Environnement	***Prédiction***	***Possible ou impossible***
	Accessibilité et connaissance	***Limitées ou illimitées***
	Dynamique	***Fixe ou variable***
	Variété	***Pauvre ou riche***
	Disponibilité des ressources	***Abondante ou restrictive***

Tableau 2.1 *critères de différenciations entre SMAs*

3.3.2 Les processus de décision ou mise en action

En fonction de leur mode de fonctionnement et de leur représentation de leur environnement, les agents peuvent être cognitifs ou réactifs. Un agent est cognitif s'il possède des connaissances sur son objectif et sur son environnement. En fonction des informations disponibles, il agit pour satisfaire son objectif, au besoin en faisant intervenir des processus de planification et de communication avec les autres agents [6]. Un ensemble d'agents cognitifs ressemble à un groupe d'individus qui doit coopérer et se concerter pour pouvoir agir. Les analogies sociales amènent les chercheurs dans ce domaine, à utiliser des travaux de sociologie lors de la mise

au point de protocoles de négociation et de concertation [6]. Un agent est réactif s'il agit en réponse à des stimuli de son environne-ment, sans connaissance d'un objectif. Ses actions sont régies par des règles prédéfinies de comportements.

Un système d'agents réactifs peut présenter un comportement intelligent ou satisfaire à un but, c'est le phénomène d'émergence. Dans un modèle, plusieurs types d'agents peuvent cohabiter.

3.3.3 Interaction et communication

L'idée d'interaction est parfois liée à celle de communication. La communication la plus évoluée nécessite un langage commun manipulant des symboles. Toutefois l'action sur l'environnement et la modification de celui-ci par un agent influencent les actions des autres agents. Dans ce cas, il y a interaction et non communication. Un système peut évoluer vers la résolution d'un problème sans qu'il y ait communication directe entre les agents. L'exemple du « tableau noir » commun peut être considéré comme un mode de communication si l'agent le modifie intentionnellement pour passer l'information aux autres agents. S'il n'y a pas d'intention, il ne s'agit pas alors de « communication » ; cependant la transformation des données du tableau, comme dans la transformation de l'environnement, peut constituer

un signal d'action pour les autres agents. L'action de ces agents dépend alors de l'action d'agents voisins ; il y a une interaction de fait qui ne peut être assimilée à une communication.

3.3.4 Concept d'agent

Un agent peut être défini comme une entité (physique ou abstraite, informatique ou robotique) poursuivant un but individuel et qui a un comportement autonome qui est la conséquence de ses observations, de sa connaissance et des interactions avec les autres agents, avec lesquels il est capable de communiquer. Il est capable d'agir sur lui-même et son environnement, ayant une capacité de perception et disposant d'une représentation partielle de cet environnement..

L'univers dans lequel se situent les agents sera défini principalement par :

- Un espace, muni de structures (topologiques par exemple),
- Un ensemble de ressources,
- Un temps universel et/ou local,
- Des lois physiques pertinentes pour l'application visée.

L'agent dans cet univers peut être défini par :

- Sa spécialité, si les agents sont typés,
- Ses objectifs et ses fonctionnalités,
- Ses caractéristiques physiques,

- Ses capacités de perception de son état interne et de l'environnement extérieur,
- Ses connaissances et ses croyances,
- Ses capacités décisionnelles,
- Ses capacités d'apprentissage : modification de ses croyances et de ses connaissances.

Les agents ont deux tendances : une tendance sociale tournée vers la collectivité (les mécanismes et connaissances associés concernent les activités du groupe) et une tendance individuelle avec des mécanismes et des connaissances contenant les règles de fonctionnement interne de l'agent.

3.3.4.1 Déterminant d'un agent

On appelle déterminant d'un agent, l'ensemble nécessaire et suffisant de ses caractéristiques structurelles, environnementales et comportementales, qui permet d'expliquer ses façons d'agir. Les caractéristiques environnementales d'un agent sont liées à la représentation que se fait l'agent de son environnement et de lui-même. Les caractéristiques structurelles d'un agent déterminent l'ensemble de ses composants, alors que les caractéristiques comportementales contraignent l'ensemble de ses comportements pour qu'ils soient en accord avec les caractéristiques environnementales [16].

Par exemple, un avion est un agent dont les caractéristiques structurelles sont la capacité, la vitesse, etc. Les caractéristiques comportementales sont le pilote et le plan de vol et sa caractéristique environnementale est sa position.

3.3.4.2 Agents cognitifs et réactifs

La granularité[3] des agents impliqués dans une application varie selon deux écoles (tableau 3.1) coexistants aujourd'hui : l'école cognitive et l'école réactive. Suivant le type d'agent utilisé, on parlera de systèmes cognitifs ou de systèmes réactifs. Dans le tableau 3.1, les différences entre les deux approches sont résumées.

3.3.4.2.1 Agents cognitifs

Les systèmes d'agents cognitifs sont fondés sur la coopération d'agents capables à eux seuls d'effectuer des opérations complexes. Un système cognitif comprend un petit nombre d'agents qui disposent d'une capacité de raisonnement sur une base de connaissances, d'une aptitude à traiter des informations diverses liées au domaine d'application, et d'informations relatives à la gestion des interactions avec les autres agents et l'environnement. Chaque agent est assimilable, suivant le niveau de ses capacités, à un système expert plus ou moins sophistiqué. On parle d'agent de forte granularité (*coarse grain*). C'est l'école cognitive qui, jusqu'à maintenant, a donné lieu aux applications les plus avancées.

3.3.4.2.2 Agents réactifs

Par opposition aux précédents, Les agents réactifs sont de plus bas niveau, ils ne disposent que d'un protocole et d'un langage de communication qui sont réduits, leurs capacités répondent uniquement à la loi *stimulus/réponse*.

[4]On entend par granularité le degré de détail des connaissances de l'agent. La granularité exprime la complexité des fonctionnalités d'un agent.

Cette approche propose la coopération d'agents de faible granularité (*fine grain*) mais beaucoup plus nombreux. L'argument mis en avant est que dans un système multi-agents, il n'est pas nécessaire que chaque agent soit individuellement 'intelligent' pour parvenir à un comportement global intelligent.

Dans les systèmes réactifs, l'intelligence émerge des interactions d'un grand nombre d'agents qui ne disposent que de peu d'intelligence. L'analogie classique de telle société d'agents est celle de la fourmilière ou de la ruche d'abeilles. Chaque insecte est relativement simple et ne dispose que d'un ensemble très réduit de comportements. Pourtant, la société d'insectes fonctionne comme si elle disposait d'un organe capable de prendre les décisions et donnant des ordres aux éléments de la société.

Les premiers travaux relatifs à cette approche ont été réalisés au MIT en 1986 par R. Brooks [24]. D'après lui, plusieurs milliers de micro-robots identiques, d'une taille aussi petite que possible, travaillant ensemble sur une tâche donnée pourront être plus efficaces qu'un seul gros robot spécialisé[4].

Systèmes d'agents cognitifs	**Systèmes d'agents réactifs**
Représentation explicite de l'environnement	Pas de représentation explicite
Peut tenir compte de son passé	Pas de mémoire de son historique
Agents complexes	Fonctionnement stimulus/réponse
Petit nombre d'agents	Grand nombre d'agents

***Tableau 2.2** - les agents cognitifs et réactifs*

Dans le même sens, une équipe de l'université de Bruxelles développe des sociétés de micro-robots, destinés à l'utilisation en milieu hostile ou au recueil d'échantillon sur d'autres planètes. D'autres travaux ont eu comme résultat des langages concurrents pour la représentation des connaissances qui peuvent être utilisé comme une plate-forme pour la conception de systèmes multi-agents réactifs.

La complexité des systèmes réactifs exige le développement de nouvelles théories dans le domaine de la coopération, de la communication et de la compréhension de nouveaux phénomènes telle que l'émergence. Toutefois, il est possible de concevoir des systèmes hétérogènes comportant les deux types d'agents. Il est aussi possible de doter les agents cognitifs de capacités de réactions aux événements : on parlera alors d'agents hybrides 17].

Notons qu'une polémique peut être lancée sur la complexité des agents cognitifs. Cette complexité peut-elle être d'un niveau de celle d'un système expert ? Si elle l'est, quelles seront les limites entre l'IA classique et l'IAD ? Et si elle ne l'est pas, jusqu'à quel niveau de complexité considère-t-on qu'un agent est cognitif ou réactif ? Ces questions montrent bien les insuffisances non encore palliées par les chercheurs dans le domaine des SMA !

[4] La construction de ces petits agents simples et dotés d'une autonomie réelle ne sera pas plus difficile que la fabrication des circuits intégrés d'aujourd'hui !

3.3.4.3 Caractéristiques d'un agent

3.3.4.3.1 Intentionnalité

Un agent intentionnel est un agent guidé par ses buts. Une intention est la déclaration explicite des buts et des moyens pour y parvenir. Elle exprime donc la volonté d'un agent d'atteindre un but ou d'effectuer une action. Notons que la notion d'intentionnalité n'existe pas chez Minsky.

3.3.4.3.2 Rationalité

Un agent rationnel est un agent qui suit le principe suivant : '*Si un agent sait qu'une de ses actions lui permet d'atteindre un de ses buts, il la sélectionne*.

Les agents rationnels disposent de critères d'évaluation de leurs actions et sélectionnent selon ces critères les meilleures actions qui leur permettent d'atteindre le but. De tels agents sont capables de justifier leurs décisions. La notion de rationalité se rapporte au comportement cognitif de l'agent.

3.3.4.3.3 Engagement

La notion d'engagement est l'une des qualités essentielles des agents coopératifs. Un agent coopératif planifie ses actions par coordination et négociation avec les autres agents. En construisant un plan pour atteindre un but, l'agent se donne les moyens d'y parvenir et donc s'engage à accomplir les actions qui satisfont ce but : l'agent croit qu'il est en mesure d'exécuter tout le plan qu'il a élaboré, ce qui le conduit (ainsi que les autres agents) à agir en conséquence. La notion de but *persistant* est prise au sens où : un agent adopte un but persistant si (i) ce but n'est pas encore atteint, (ii) il croit que le but sera atteint plus tard et (iii) il doit atteindre le but ou être persuadé qu'il ne l'atteindra jamais avant de l'avoir abandonner.

3.3.4.3.4 Adaptativité

Un agent adaptatif est un agent capable de contrôler ses aptitudes (communicationnelles, comportementales, etc.) selon l'agent avec qui il interagit. C'est un agent d'un haut niveau de flexibilité.

3.3.4.3.5 Intelligence

En conclusion, on appelle agent intelligent un agent cognitif rationnel, intentionnel et adaptatif.

3.4 Architecture d'agent

3.4.1 Architecture Abstraite d'un Agent

Nous allons formaliser l'architecture interne de façon formelle d'un agent ""intelligent".

Soit $S = [s 1 , s 2 ,.....]$ l'ensemble des états de l'environnement.

Les compétences d'un agent sont représentées par l'ensemble A des actions qu'il peut réaliser :

$A = \{ a 1 , a 2 ,.....\}$

^

Un agent peut être vu comme une fonction

agir : $S^* \rightarrow A$, qui fait correspondre une séquence d'état de l'environnement avec des actions. Intuitivement l'agent choisit l'action suivante en analysant l'historique des actions précédentes associées aux différents états de l'environnement. Il intègre le passé dans son comportement courant..

3.4.2 Structure interne

La figure 3.1 décrit l'architecture globale d'un agent cognitif. On distingue essentiellement : le savoir-faire, les croyances, la connaissance de contrôle, l'expertise de l'agent et la connaissance de communication.

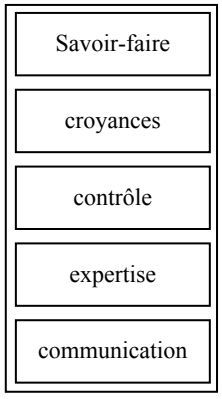


Figure 2.1*- structure d'un agent*

- ***Savoir-faire*** *:* Le savoir-faire est une interface permettant la déclaration des connaissances et des compétences de l'agent vis à vis des autres agents. Il permet la sélection des agents à solliciter pour une tâche donnée. Il n'est pas nécessaire, mais il est très utile pour améliorer les performances du système, quel que soit le mode de coopération utilisé.

- ***Croyances*** : Dans un univers multi-agents, chaque agent possède des connaissances sur lui-même et sur les autres. Ces connaissances ne sont pas nécessairement objectives, on parle alors de croyances d'un agent. La représentation de ces connaissances ou croyances est à la base de la conception de tout système multi-agents, puisque c'est elle qui détermine en grande partie le comportement intelligent de chacun des agents. Il existe cependant un problème, qui est la difficulté de garantir la consistance ces croyances. Les logiques des connaissances et des croyances s'intéressent à la formalisation de telles connaissances considérées comme incertaines. Par exemple, un agent A croit que P est vrai, tandis qu'un agent B croit que P est faux. Qui a raison ? Plusieurs solutions sont possibles. S'il existe plusieurs agents, on peut passer au vote et la valeur de croyance dépendra de son résultat. Une autre possibilité consiste à demander à chacun de se justifier. Le raisonnement de l'un des deux agents porte peut-être sur une information qui est maintenant erronée, ce qui infirme la validité de ses conclusions.

- ***Contrôle*** : La connaissance de contrôle dans un agent est représentée par les buts, les intentions, les plans et les tâches qu'il possède.

- ***Expertise*** : C'est la connaissance sur la résolution de problème. Par exemple, cette connaissance correspond à la base de règles pour un système expert utilisant le formalisme de règle.

- ***Communication*** : L'agent doit posséder un protocole de communication lui permettant d'interagir avec les autres agents pour une bonne coopération et une bonne coordination d'actions [17]. Un acte de communication est défini en tant qu'action modifiant l'état du monde, en l'occurrence l'état interne d'autres agents. D'autres connaissances de communication peuvent être disponibles, par exemple les connaissances sur les réseaux de communication (tous les agents ne sont pas forcément en liaison directe).

A ces différents types de connaissances, on peut ajouter la connaissance liée au mode de la coopération : fonctionnement en mode appel d'offre, en mode compétition ou en mode commande.

3.5 Fonctionnement

Les agents sont immergés dans un environnement dans lequel et avec lequel ils interagissent. D'où leur structure autour de trois fonctions principales : percevoir, décider et agir. Parmi les sous-fonctions importantes d'un agent on peut citer : la détection de conflits, la révision des croyances, la coopération (négociation, coordination), l'apprentissage, etc. (figure 3.2)

Un agent a la possibilité d'acquérir des connaissances sur l'environnement externe (perception). Il a aussi des capacités d'interaction avec les autres agents (communication, négociation). En fonction des connaissances et croyances dont il dispose et des buts qu'il se fixe suite à une perception ou à une interaction avec le monde extérieur, l'agent doit élaborer un plan d'action. Pour cela, il doit décider quel serait le but à retenir et à satisfaire en premier, ensuite planifier en fonction de ce but et passer à l'exécution.

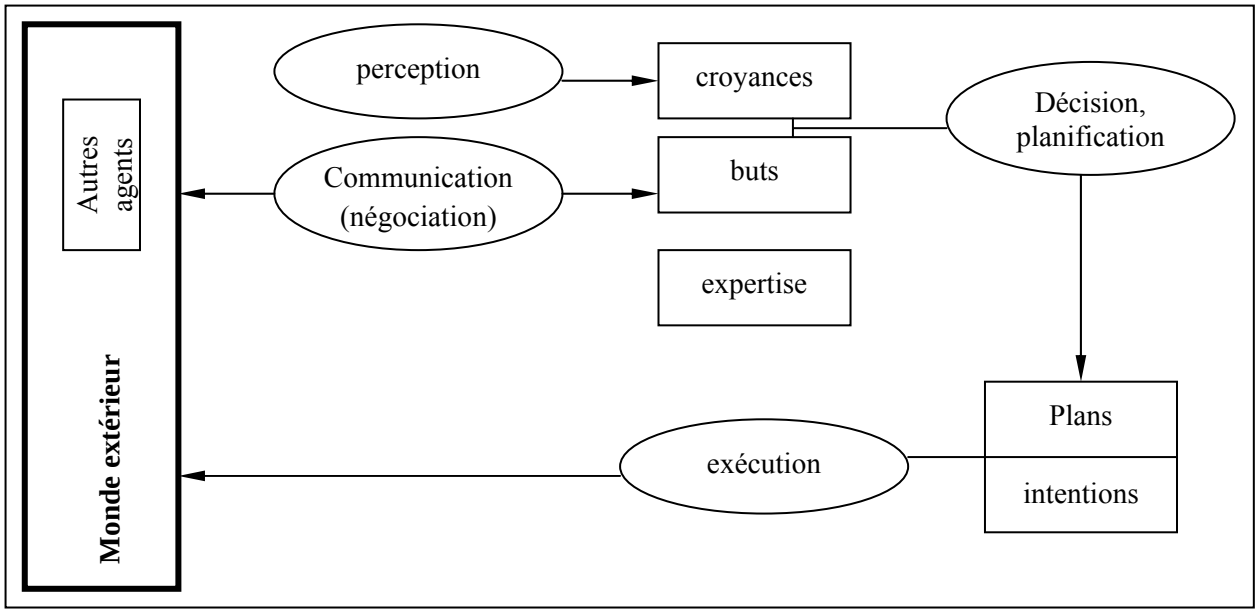


Fig 2.2 : *fonctionnement d'un agent*

3.5.1 Perception

Les connaissances d'un agent ont plusieurs origines :

- Le savoir initial de l'agent ;
- La perception de soi (perception proprioceptive) et du monde (perception extéroceptive) ;
- La communication avec les autres agents.

Généralement, les informations issues de la perception et du savoir initial de l'agent sont considérées comme des connaissances certaines puisqu'elles n'ont subi aucune mise à jour, alors que les connaissances provenant des autres agents sont considérées incertaines puisqu'elles évoluent sans que l'agent en soit forcément informé. On peut pour cela associer à chaque connaissance son origine afin d'en évaluer la crédibilité et d'en permettre la vérification.

3.5.2 Prise de décision

La prise de décision est une des caractéristiques des agents rationnels, l'agent tient compte de ses croyances pour faire son choix.

Durant son exécution, un agent se fixe un certain nombre de buts, suite à ses observations et ses interactions avec le monde (perception, communication, négociation). Il doit donc sélectionner le but à satisfaire en premier et pour chaque but, l'action qui permet de l'atteindre. Face à de telles situations, l'agent analyse les différentes alternatives en termes d'utilité et d'incertitude (quel intérêt et quelle chance possède l'action pour être effectuée ?). Parmi les techniques utilisées pour le choix des actions, on peut citer l'utilisation de la notion de carte cognitive (cognitive Map) [16].

3.5.3 Planification

La planification dans les systèmes d'IA classique repose sur l'hypothèse d'un univers statique où seules les actions du planificateur sont prises en compte. L'IAD, au contraire, offre des possibilités de négociation

autorisant une gestion locale des conflits et une planification dynamique. En effet, du fait de l'intervention d'autres agents, un plan peut être remis en cause. L'agent doit pour cela, alterner planification et exécution et réviser des parties de son plan, on parle alors de planification *réactive*.

La planification dans les systèmes multi-agents est distribuée : il n'existe pas de plan global, en revanche, chaque agent construit son propre plan en coordonnant avec les autres (cas d'agents coopératifs). Certains systèmes d'IAD présentent une planification centralisée où un organe central se charge de la gestion des conflits et de l'élaboration d'un plan global.

La planification d'un agent se traduit par des actions et de tâches à entreprendre vers des buts particuliers où il sera capable de :

- Choisir des buts, d'ordonner dynamiquement leur priorité, de les modifier ou de les abandonner,
- Déterminer par quelles actions et dans quel ordre un ou plusieurs buts peuvent être atteints,
- D'affiner/modifier les prédictions, voir remettre en cause les buts choisis, en fonction d'un retour sensoriel sur les effets réels des actions ou sur des événements imprévus,

4 Société d'agents

4.1 Introduction

Si l'on part du principe que les systèmes "intelligents" ne sont pas des systèmes isolés, mais qu'ils fonctionnent en interaction avec d'autres systèmes, alors on peut les considérer sous l'angle d'une société de systèmes (agents). Comme nous l'avons évoqué en introduction, les systèmes ouverts sont complexes, très grand, dynamiques et imprévisibles. Ces caractéristiques imposent à toute tentative d'instrumentalisation de tels systèmes, qu'elle soit répartie autant que possible, mais judicieusement dans leur environnement.

Les agents autonomes ont pour objectif de chercher, communiquer et gérer l'information. Ils doivent être résolument patients (durée de vie suffisamment longue), adaptatifs (exploration et apprentissage sur l'environnement) et sociaux (interaction et coordination avec d'autres agents . agents non omniscient).

Lorsque des agents évoluent en société, ils peuvent jouer des rôles particuliers au sein de groupes (petites sociétés). Le groupe définit donc ses rôles (pour ses agents), et le rôle définit des responsabilités (engagements) qui lui sont associées. Quand un agent rejoint un groupe, il le fait à travers un ou plusieurs rôles du groupe ; il acquière ainsi les responsabilités correspondantes. Les agents peuvent décider d'eux même pour rejoindre un groupe ; dans ce cas ils sont tenus de se plier aux rôles qu'ils endossent dans le groupe. Le groupe définit le contexte social (règles d'interaction) dans lequel l'agent va interagir avec les autres.

Les approches BDI, basées sur les états mentaux (intentions, désirs, croyances) des agents, ne suffisent pas pour aborder tous les aspects des interactions sociales. Les approches fondées sur les modèles économiques (loi du marché, offre/demande ...), bien que générales restent limitées en pratique, car la notion de fonction d'utilité reste centrée sur l'agent lui même (agent égoïste . gain personnel).

Un engagement collectif ou social engage un agent vis a vis d'autres agents et non vis a vis de lui même. C'est un moyen souple de contraindre le comportement d'un agent. Il en résulte une notion importante appelée la "dépendance sociale", qui permet d'introduire dans les connaissances d'un agent le fait qu'il dépende d'un autre agent dans la résolution d'un but donné.

Dans les sociétés composées d'un très grand nombre d'agents, le concept de lois sociales complète celui de dépendances sociales, pour gérer le comportement des agents.

4.2 Organisation sociale

Une société d'agents est constituée de trois éléments : un ensemble d'agents, un ensemble de tâches à réaliser et un ensemble d'objets associés à l'environnement [10]. L'organisation sociale d'un système multi-agents est la manière dont le groupe est constitué, à un instant donné, pour pouvoir fonctionner. Cette organisation peut être statique ou dynamique. Son dynamisme est le fait que le groupe se réorganise à chaque fois en fonction de la tâche à accomplir.

Les structures organisationnelles en IAD se différencient selon trois facteurs : le type de décentralisation (lié au type de communication), le type des agents appartenant au groupe et le mode de coopération entre les agents. Les chercheurs en IAD essayent d'adapter les modèles sociologiques aux systèmes multi-agents.

Le choix d'une structure organisationnelle dépend à la fois des caractéristiques du groupe et de la tâche à accomplir. Une tâche étant caractérisée par la complexité du raisonnement, la quantité de connaissances et la capacité d'action. Quelques travaux s'intéressent aux mesures de l'efficacité d'un système en fonction de la complexité de son organisation sociale.

Cas d'agents réactifs Il n'y a pas vraiment d'organisation dans les systèmes d'agents réactifs. En effet, la structure du système émerge des comportements de ses membres et non d'une volonté d'organisation.

Cas d'agents cognitifs Les organisations dans ce cas sont variables. Elles sont essentiellement liées à chaque cas et à chaque type de fonction (voir par exemple Contract Net et SYNCHRO[5] [17]).

4.3 Contrôle et prise de décision

En fonction de l'organisation sociale du système, le contrôle global peut être le résultat des mécanismes de contrôle locaux aux agents ou - dans les sociétés hiérarchisées - régit par certains 'agents' (superviseur) qui peuvent prendre en main le contrôle de quelques éléments de la société, en entretenant soit des rapports maître-esclave avec les autres agents, soit en guidant la mise en action ou en attente des agents.

Les mécanismes de décision font référence à tout ce qui est allocation de tâches ou de ressources, en relation avec la résolution de conflit. Ils regroupent, entre autres, toute la partie décisionnelle des protocoles de négociation. La répartition des tâches pour la résolution d'un problème nécessite d'être la plus optimale possible, c'est d'ailleurs l'un des facteurs importants de la convergence vers la résolution. Il est nécessaire que chaque agent adopte un rôle adapté, en fonction de l'organisation et de ses compétences. On parle alors d'architecture d'agents et de découpage de connaissances. Le problème ne se résout plus en partant d'un état initial (l'état de départ ou l'état but comme le suggère l'approche de l'IA classique) pour arriver à un état final,

[5] SYstème Naturel de Communication Homme-Robot-Ordinateur est un système multi-robots, destiné à améliorer la flexibilité des transports dans une unité de production industrielle et peut s'étendre à d'autres applications de la robotique mobile.

mais en construisant au fur et à mesure des solutions partielles qui se trouvent sur le chemin. Chaque agent cherche à apporter des éléments de solution au fur et à mesure de l'exploration. On distingue trois approches différentes dans la répartition de la connaissance entre les agents :

4.3.1 Approche modulaire

Elle consiste à découper un système en un certain nombre de modules qui doivent travailler ensemble. Chaque module est conçu comme un système expert appelé *spécialiste*, disposant de sa propre base de connaissances et communiquant avec les autres modules pour accomplir sa fonction.

4.3.2 Expertise en concurrence

Chaque agent dispose d'une base de connaissance qui reflète un point de vue spécifique au problème à résoudre. Par exemple, un système de diagnostic peut comprendre deux agents : l'un spécialisé dans l'analyse profonde et l'autre dans l'analyse de surface où il faut gérer les conflits.

4.3.3 Simulation d'univers

Il ne s'agit pas de résoudre un problème à proprement parler, mais de définir une population d'agents dont les règles de comportement vont amener le système à résoudre le problème

4.4 Coopération

La coopération est une caractéristique très importante dans les systèmes multi-agents. En effet, une résolution distribuée d'un problème est le résultat de l'interaction coopérative entre les différents agents.

Dans le cadre d'une telle dynamique collective, un agent doit disposer - en plus de la connaissance reflétant son degré d'implication (croyances, buts, intentions, engagements, modèle de soi et d'autrui) - d'un certain nombre de compétences nécessaires pour la coopération. Il doit pouvoir [17] :

- Mettre à jour le modèle du monde environnant,
- Intégrer des informations venant d'autres agents,
- Interrompre un plan pour aider d'autres agents,
- Déléguer la tâche qu'il ne sait pas résoudre à un autre agent dont il connaît les compétences.

Ces caractéristiques forment les qualités essentielles d'un agent coopératif.

4.4.1 Coopération et structure d'organisation

Selon Ferber [18], il existe deux architectures d'organisation pour les sociétés d'agents :

- *Structure horizontale :* Dans de telles sociétés, tous les agents sont au même niveau, il n'y a pas d'agents maîtres et d'agents esclaves. C'est le cas, par exemple, d'un groupe d'agents ayant des spécialités différentes travaillant pour la résolution d'un même problème.
- *Structure verticale* : Dans une architecture verticale, les agents sont structurés par niveaux[6]. Dans un même niveau on retrouve localement une structure horizontale. Le fonctionnement des agents

[6] Agents traitant différents niveaux d'abstraction du problème.

dans de telles sociétés est le suivant : l'agent reçoit le problème à résoudre d'un autre agent qui lui est supérieur dans la hiérarchie, il le décompose-en :

- des sous-problèmes auxquels il peut répondre localement,
- des sous-problèmes qu'il pourrait résoudre en coopérant avec les autres agents du même niveau que lui,
- Des sous-problèmes qu'il fait suivre aux agents du niveau inférieur dans la hiérarchie.

4.4.2 Modèles de coopération

Quelle que soit l'organisation d'une société d'agents, un agent peut coopérer suivant les modèles suivants :

- *Coopération par partage de tâches et de résultats*, avec la possibilité de prendre en compte localement les plans des autres.
- *Commande* : un agent supérieur A décompose le problème en sous-problèmes qu'il répartit entre les autres agents Xi. Ceux-ci le résolvent et renvoient les solutions partielles à A.
- *Appel d'offre* : A décompose le problème en des sous-problèmes dont il diffuse la liste. Chaque agent Xi qui le souhaite envoie une offre ; A choisit parmi celles-ci et distribue les sous-problèmes. Le système travaille ensuite en mode commande.
- *Compétition* : Dans le mode compétition, A décompose et diffuse la liste des sous-problèmes comme dans le mode appel d'offre, chaque agent Xi résout un ou plusieurs sous-problèmes et envoie les résultats correspondants à A qui à son tour fait le tri.

4.5 Résolution de conflits

Les agents coopératifs ont besoin de coordonner leurs activités et négocier leurs actions afin d'éviter les situations conflictuelles pour résoudre un problème .

4.5.1 Coordination

La coordination des actions permet aux agents de considérer toutes les tâches et de ne pas dupliquer le travail. Elle est liée à la planification et à la résolution des conflits, car c'est à ce niveau qu'on tient compte des actions (plans) des différents agents. Elle est également nécessaire lors de l'accès à certaines ressources.

Dans les systèmes d'IAD, la coordination des actions des agents peut s'organiser suivant deux schémas différents en fonction de l'organisation sociale : une coordination au moyen d'un système capable de déterminer et de planifier (globalement) les actions des différents agents ou à l'inverse, on décide de donner une totale autonomie aux agents qui, à leur tour, identifient les conflits pour les résoudre localement. L'inconvénient de l'utilisation d'un coordinateur centralisé est de pouvoir engendrer un goulot d'étranglement par une

surabondance de demandes. L'inconvénient de l'autonomie des agents réside dans le temps passé à envoyer et à écouter des messages.

On peut distinguer deux types de coordination : la coordination due à la gêne (les agents doivent par exemple, coordonner leurs plans de navigation pour s'éviter mutuellement) et la coordination due à l'aide (dans un environnement multi-robots, les agents doivent par exemple, synchroniser leurs actions pour pouvoir agir efficacement et transporter un objet).

4.5.2 Négociation

Les activités des agents dans un système distribué sont souvent interdépendantes et entraînent des conflits. Pour les résoudre, il faut considérer les points de vue des agents, les négocier et utiliser des mécanismes de décision concernant les buts sur lesquels le système doit se focaliser. La négociation est caractérisée par :

- Un nombre faible d'agents impliqués dans le processus ;
- Un protocole minimal d'actions : proposer, évaluer, modifier et accepter ou refuser une solution.

Le processus de négociation ne consiste pas forcément à trouver un compromis mais peut s'étendre à la modification des croyances d'autres agents pour faire prévaloir un point de vue. Pour le mener à bien, il est nécessaire d'utiliser un protocole qui facilite la convergence vers une solution.

4.6 Communication

4.6.1 Protocoles de communication

Dans un système distribué, il faut que les entités disposent d'un langage commun pour pouvoir échanger des informations. Ce langage, appelé mécanisme de communication inter-processus, est un ensemble de primitives connues par chaque entité et dont l'ensemble constitue un protocole de communication qui permet de structurer et d'uniformiser les interactions inter-agents.

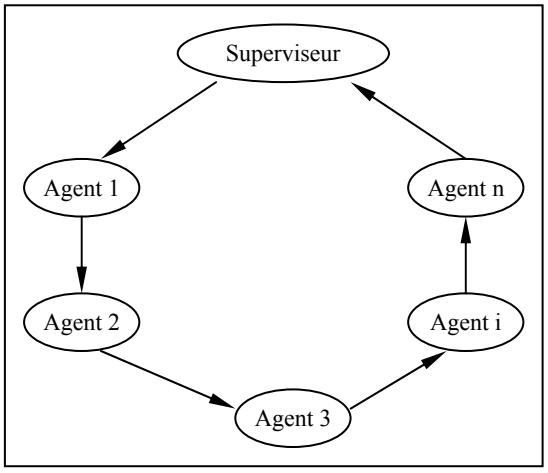
Les primitives de base d'un protocole de communication sont les suivantes :

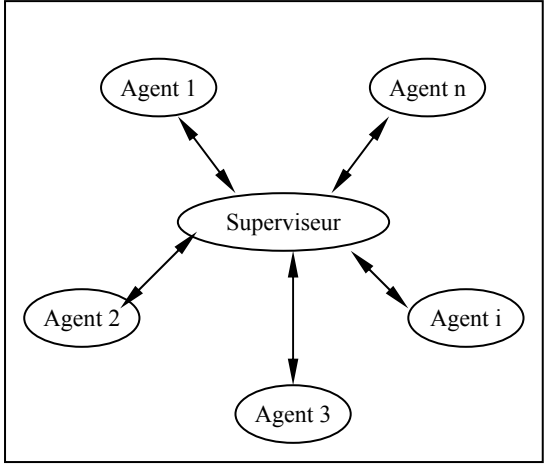
- Etablissement d'une connexion entre deux entités ;
- Identification du nœud destinataire dans un réseau de communication ;
- Envoi de données ;
- Réception de données ;
- Définition d'un type de communication : synchrone ou asynchrone.

Un protocole de communication peut être basé sur une architecture standard de communication, en l'occurrence, Internet ou OSI. Un exemple de système de création et d'implantation de protocole de communication est présenté dans [19]. En revanche, la communication directe n'est pas toujours possible. Dans ce cas, il convient de faire appel à un intermédiaire appelé *médiateur*.

4.6.2 Architectures de communication

La communication entre les agents peut être organisée suivant trois schémas différents :





Réseaux en anneau (figure 3.4) Les interactions dans ce genre d'organisation sont trop lentes, un message destiné à un agent doit transiter par (au plus) n-1 agents

.

Fig 2.4 *- réseaux en anneau* ***Fig 2.5*** *- réseaux en étoile*

Réseaux en étoile (fig 3.5) Ils présentent l'avantage de l'accès rapide entre le superviseur et les autres agents. La nature bidirectionnelle des connexions rend complexe la gestion des interactions inter-agents.

Réseaux hybrides (figure 3.6) Ce type d'organisation est une solution hybride reliant deux types d'organisation : un réseau en bus et un réseau en étoile.

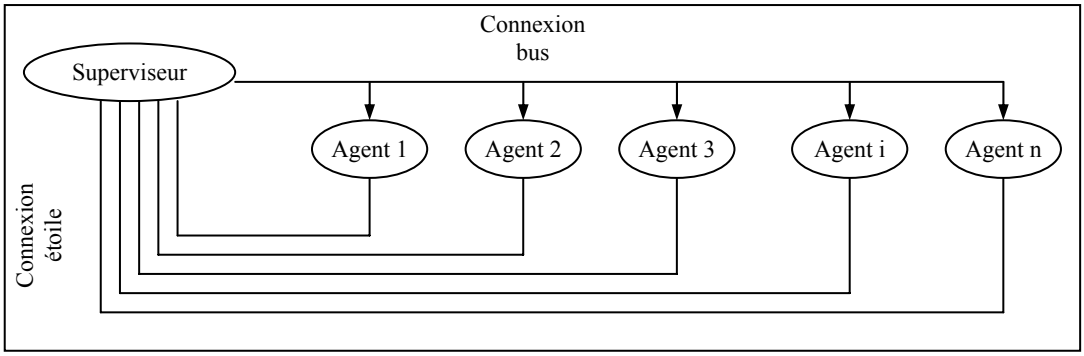


Fig 2.6 *- réseaux hybrides*

4.6.3 Actes de communication

Une communication dans les systèmes multi-agents, est un acte pouvant modifier l'état interne de l'agent récepteur. L'agent peut être amené à remettre en cause son plan d'action ou à modifier ses croyances.

Une communication peut correspondre à une information, une requête ou une interrogation ; elle doit avoir une sémantique reconnue par l'agent récepteur[7]. La réception d'un message par un agent doit aider à une convergence vers la solution. Tout cela demande une bonne étude (par l'agent émetteur) du contenu du message lors de sa composition.

On présente une distinction entre les différents types de communication :

[7] Par exemple, comment décrire un objet de l'environnement à un agent ? Il y a un problème de référence : les objets n'ont pas nécessairement un nom unique ou les mêmes caractéristiques significatives pour tous les agents.

- ***Communication sélective ou diffusée*** : par opposition à la diffusion, une communication sélective s'adresse à un nombre restreint d'agents, elle suppose donc un critère de sélection ;

- ***Communication sollicitée ou non sollicitée*** : une communication peut être demandée par un autre agent ;

- ***Communication avec ou sans accusé de réception*** : dans le cas d'une communication avec accusé de réception, l'émetteur attend une confirmation de la bonne réception du message ;

- ***Communication à transmission simple ou répétée*** : le message peut être répété sur plusieurs envois.

4.6.4 Modes de communication

Communication par envoi de messages Les agents sont en liaison directe et envoient leurs messages directement et explicitement au destinataire. La seule contrainte est la connaissance de l'agent destinataire. Les systèmes fondés sur la communication par envoi de messages relèvent d'une distribution totale à la fois de la connaissance, des résultats et des méthodes utilisées pour la résolution du problème. Les systèmes d'acteurs en sont l'illustration parfaite.

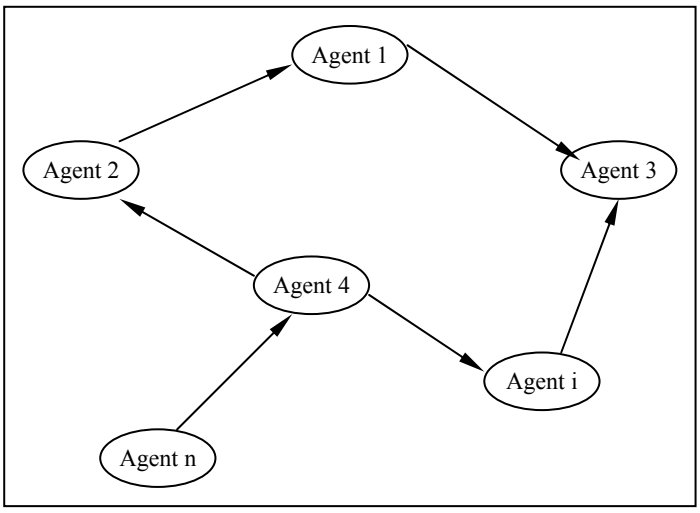


Fig 2.7 - communication par envoi de messages

Communication par partage d'informations Les composants ne sont pas en liaison directe mais communiquent via une structure de données partagée où ils viennent lire et écrire et où l'on trouve les connaissances relatives à la résolution (état courant du problème) qui évolue durant le processus d'exécution. Cette manière de communiquer est l'une des plus utilisées dans la conception des systèmes multi-experts. L'exemple parfait d'utilisation de ce mode de communication est l'architecture de blackboard.

4.6.5 Actes de langages

La théorie des 'actes de langages' ou '*Speech Acts*', est introduite par le philosophe anglais J.-L. Austin dans son livre 'How to do things with words'. Elle constitue un fondement théorique pour la spécification de

modules de communication. Elle est basée sur la constatation suivante : lorsqu'on parle, on effectue des actes. Cela apparaît clairement dans le cas des verbes appelés ‘performatifs’ (comme promettre, ordonner, demander, etc.) où l'acte est effectué par l'énoncé même : ‘je t'ordonne de sortir’. Mais on peut généraliser aux autres verbes appelés ‘constatatifs’ (par exemple : ‘il pleut’ devient ‘je déclare qu'il pleut’).

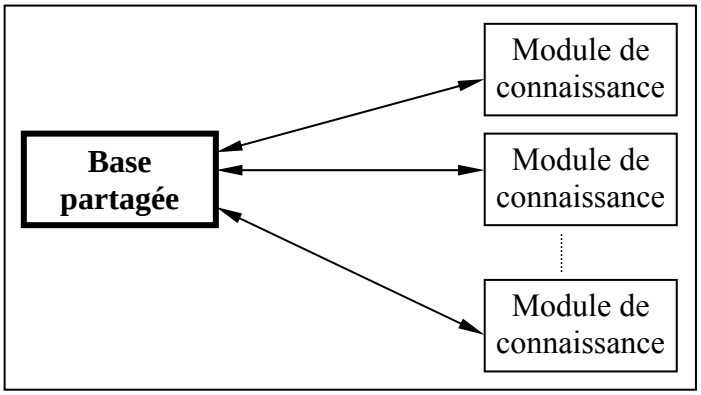


***Fig 2.8** - communication par partage d'informations*

Austin distingue trois types d'actes dans un énoncé :

- ***Actes locutoires*** : activité mentale et physique de formulation et d'articulation de l'énoncé. L'acte réussit si l'énoncé est formulé et articulé correctement ;

- ***Actes illocutoires*** : actes effectués par l'interlocuteur lors de la formulation de l'énoncé. En énonçant ‘il est dix heures ?’, on ne peut comprendre s'il s'agit d'une affirmation ou d'une interrogation. Faire comprendre qu'il s'agit bien d'une interrogation est un acte illocutoire qui réussit. L'acte illocutoire est donc codé dans le message. Dans cet exemple, si l'auditeur comprend que c'est une affirmation, alors l'acte illocutoire échoue ;

- ***Actes perlocutoires*** : c'est la conséquence indirecte mais pouvant être intentionnelle des actes locutoires et illocutoires sur l'auditeur. Ces actes dépendent du contexte, et contrairement aux actes illocutoires, ils ne sont pas codés dans l'énoncé. Par exemple à l'énoncé ‘il fait froid’ est associé à l'acte perlocutoire de mise en marche du chauffage.

Ce sont surtout les actes illocutoires qui ont fait l'objet de nombreux travaux. Une bonne synthèse sur tous les aspects des actes de langages, sur les différents points de vues et sur la ‘programmation orientée agent’ basée sur ces actes peuvent être consultés dans [27]

Dans les travaux sur les systèmes multi-agents, plusieurs chercheurs sur la communication commencent à s'intéresser à la théorie des actes de langages. Cette théorie permet une bonne modélisation de la communication entre les agents. Les actes de langages permettent de bien illustrer la théorie des intentions, en effet, à une intention correspond un certain type d'actions (actes illocutoires). L'intention, et par conséquent la

communication, seront donc reconnues à travers le type de l'action (déclaration, interrogation, requête, offre, etc.). Cela impose une mise en correspondance entre les intentions et les modèles internes de l'émetteur et du récepteur des messages. Une bonne synthèse est présentée dans [27]

Toutefois, la théorie des actes de langages reste insuffisante pour la modélisation de la coordination et de la négociation et donc de la coopération entre les agents car elle ne permet pas de décrire les informations échangées : quand ont-elles été échangées, quel agent fait quoi, la décomposition et l'allocation des tâches, etc.

4.7 Emergence

L'émergence est un concept relatif aux systèmes réactifs. Une fonctionnalité qui émerge est une fonctionnalité qui n'a pas été explicitement programmée, elle est le résultat d'interaction entre deux ou plusieurs comportements. Cette notion ressemble beaucoup à l'éco-résolution[8] .

Par exemple, si on veut qu'un robot (en marche) suive un mur (comportement suivre le mur), on définit deux comportements (antagonistes) : être attiré par le mur et être repoussé par le mur. De l'interaction de ces deux comportements va émerger un nouveau comportement : suivre le mur.

Dans ce cas, on parle d'émergence de comportement, mais il existe aussi d'autres formes d'émergence telles que l'émergence de rôles, l'émergence de stratégies, l'émergence de contrôle et l'émergence des structures de coordination. L'émergence est un phénomène compliqué et très peu compris, beaucoup de chercheurs s'y intéressent.

6 Conclusion

Dans ce chapitre, nous avons présenté une étude des univers multi-agents et une architecture d'agents intelligents, et essayé de clarifier la terminologie du domaine : IAD, SMA et systèmes ouverts.

L'IAD et les SMA sont des thèmes de recherche en cours d'exploration. Ils font intervenir plusieurs domaines de recherche tels que les systèmes répartis, la biologie, l'IA, la psychologie cognitive et la sociologie. De ce fait, les termes utilisés varient selon les différents chercheurs.

Pour la conception d'applications d'IAD, un besoin s'exprime pour une architecture de base pour la construction de systèmes multi-agents avec un langage de spécification, des protocoles de communication et de coopération ainsi que des moyens de mise en œuvre des différentes lois sociales dans une société d'agents.

Nous avons proposé une synthèse des travaux en IAD et SMA. Les principales conclusions que nous pouvons en tirer sont les suivantes :

- L'approche distribuée de l'IAD s'adapte mieux à la réalité des systèmes complexes que l'IA classique, en effet les comportements que l'on vise à modéliser sont le plus souvent le résultat d'une coopération entre plusieurs entités ; L'IAD enrichit le processus de résolution de problèmes en le partageant entre plusieurs agents ;

[8] L'éco-résolution est une technique de résolution de problème par coopération entre plusieurs entités. La résolution s'effectue par la dynamique d'évolution des différentes entités qui adaptent leurs comportements locaux aux contraintes de l'environnement jusqu'à ce que les buts soient atteints. L'éco-résolution conduit à l'éco-programmation (computational ecology).

- L’interaction de plusieurs expertises incomplètes ou peu fiables peut mener à une expertise plus sûre et plus robuste ;
- L’évolution des machines parallèles est un atout favorable à cette approche ;
- L'IAD permet de respecter les normes du génie logiciel dans l'élaboration des systèmes, à savoir : la modularité, la fiabilité et la réutilisabilité ;
- L’IAD est pluridisciplinaire, un système multi-agents doit intégrer certains mécanismes régissant la dynamique de participation d'un individu à la vie du groupe (ce qui nécessite des notions sociologiques, etc.).

Beaucoup de systèmes existent mais la plupart présentent une centralisation du contrôle et n’intègrent pas des concepts comme l'intentionnalité ou la rationalité. On est donc loin des vrais systèmes multi-agents tels qu'ils sont théoriquement définis.

Chapitre III

Etat de l'art des Plates Formes Multi-Agents

Résumé : ***Malgré les résultats satisfaisants de la simulation des systèmes complexes et hétérogènes en utilisant le formalisme orientée agent, néanmoins les modèles obtenus après implémentation en réalité et en pratique n'ont rien avoir avec le formalisme agent, pourquoi ? La réponse réside dans le fait que les conception sont implémentées au moyens de formalisme qui n'est pas orientée agent, tel l'approche objet et la programmation orientée objet. Autrement dit les agents eux mêmes sont simulés par des objets, par exemples, c'est à dire qu'il n y a pas d'outil de conception et d'implémentation où l'agent est implémenter en tant qu'agent et non pas simulé. Ceci est bien entendu l'objectif des plates formes multi agents qui visent à offrir aux modélisateurs un support où tout représenté sous forme d'agent.***

1 Introduction

dans ce chapitre nous allons essayé de mettre l'accent sur les difficultés que rencontrent les concepteurs de plate formes multi-agents à travers la présentation et la discussion de deux plates formes multi-agents, à savoir MadKit (Multi-Agent Development Kit) et la plate forme OpenCybel. Il existe cependant sur le terrain plusieurs genres de plate formes MA et sont toutes spécifiques à un ou plusieurs domaines d'application.

2 Les plates formes Multi-Agents

Depuis plusieurs années, on a vu se multiplier les plate-formes de développement agents. Sous ce terme illusoirement générique se cachent en fait plusieurs grandes catégories de plate-formes. Même s'il n'est pas question de voir ici en détail l'ensemble de ces plate-formes (ou même de ces catégories), nous allons en parcourir quelques exemples et noter les traits marquants et leurs justifications.

2.1 Catégorisation

2.1.1 Les plate-formes d'agents mobiles

Elles se situent souvent dans la filiation du précurseur TELESCRIPT [20], et présentent parfois une confusion entre "agent mobile" et "code mobile".

Un exemple récent de plate-forme d'agents mobiles est la plate-forme Aglets, développée par IBM Japon. On y retrouve les mécanismes de base de la migration : un identifiant unique, un itinéraire qui décrit les différents noeuds à visiter et les actions à prendre à chaque étape. Cela impose évidemment la mise en place d'une plate-forme Aglet sur les noeuds désirés du réseau, car la migration n'est possible que si l'infrastructure est présente. Un outil de conception permet de spécifier les contraintes de sécurité à appliquer à chaque aglet sur chaque site. La communication entre aglets sur les sites se fait soit par un tableau noir, soit par passage de messages synchrones ou asynchrones.

Il faut bien dire que les fonctionnalités que propose Aglets sont plus à rattacher au domaine des objets mobiles que véritablement des agents. En effet, les aspects d'autonomie d'action ou d'interaction sont réduits à leur plus simple expression.

2.1.2 Les outils de la simulation Multi-Agents

Ce sont souvent des outils construits de façon adhoc pour un domaine de simulation, ou même une seule application [14] [21].

SWARM [22] est une exception notable de plate-forme à vocation générique. Dans ce cas, l'outil propose un modèle suffisamment générique de simulation (basé sur une décomposition récursive des systèmes) sur lequel on peut bâtir éventuellement des modèles spécifiques. En fait, la force de cette plate-forme réside plus dans l'importance des outils annexes : obtention des données par des sondes que l'on peut positionner sur quasiment tous les points de l'application, traitement et première analyse par des outils statistiques, visualisation

en temps réel des paramètres, lancement de jeux d'expériences. Le modèle d'agent est peut-être paradoxalement la partie la plus faible de l'outil.

2.1.3 Les plate-formes orientées modèle

Sous ce qualificatif se rangent des plate-formes à vocation généraliste du point de vue applicatif, mais développées par rapport à un modèle très spécifique d'agent dont elles permettent en fait la validation pratique.

AgentBuilder [23] appartient à cette catégorie, et réunit à la fois un outil de conception et une plate-forme d'exécution. Le modèle des agents AgentBuilder dérive du modèle Agent0 [24], la communication s'effectue par messages KQML. L'architecture laisse néanmoins le concepteur libre de rajouter de nouveaux actes de langages pour s'adapter à un domaine particulier. L'outil de conception est extrêmement ciblé sur le type de modèle défini et couvre l'intégralité du processus de développement. Le concepteur a la possibilité de définir ses schémas d'interaction, ses structures sociales, ses mécanismes de tâches et de réactions aux messages, etc. Néanmoins, toute construction est strictement limitée aux modèles définis et des variations sont très difficiles à mettre en œuvre. L'autre partie de l'outil est un moteur d'exécution, qui est en fait un interprète pour les définitions d'agents générées par l'outil de conception.

On retrouve ici un des traits caractéristiques des systèmes construits sur un modèle d'interaction KQML : on se sert de la spécification KQML comme d'une base pratique pour avoir un consensus suffisant sur la sémantique des différents actes, mais on s'autorise des libertés avec l'usage et l'extension des essages. Il s'agit bien de faciliter la vie du concepteur de système plutôt que d'essayer de faire inter opérer des agents d'applications différentes, sans parler d'intégration entre plate-formes.

Les extensions d'architectures classiques Certaines infrastructures ont aussi étés désignées comme des technologies de choix pour l'implémentation d'applications multi-agents (tels que CORBA [25] ou Jini [26]). Il est vrai que dans certains cas, pratiquement rien ne manque, hormis une sémantique appropriée. Voyager est un exemple d'infrastructure "mixte" : c'est à la base un bus objet (ORB) écrit en Java, mais étendu par quelques fonctionnalités intéressantes de mobilité d'agent. Voyager permet au concepteur d'écrire des agents, en sachant que la mobilité est fournie au niveau de l'infrastructure, avec quelques autres services. Par exemple, Voyager peut mettre en place automatiquement des adresses permettant de faire suivre les messages au fur et à mesure du déplacement de l'agent. La communication peut être synchrone (et c'est une simple invocation de méthode) ou asynchrone, via des boîtes aux lettres éventuellement distribuées.

Par contre, tout comme Aglets, Voyager repose essentiellement sur les mécanismes du langage d'implémentation (Java) en particulier au niveau sérialisation et gestion fine de la sécurité, mais n'utilise pas vraiment un modèle de comportement riche ou tout au moins un niveau de description authentiquement agent.

2.2 La standardisation FIPA

Une approche très différente, par rapport aux grandes catégories de plate-formes que l'on a parcourues, est proposée par un consortium universitaire et industriel, la FIPA (Foundation for Intelligent Physical Agents).

Il s'agit d'un effort d'intégration et d'interopérabilité entre applications multi-agents passant par l'écriture de spécifications, pour normaliser rapidement les technologies agents. Comme ce souci de généricité et d'intégration est aussi le nôtre, nous allons explorer plus en détail cette proposition.

Un ensemble de spécifications

On peut dégager deux grands ensembles de spécifications "normatives" dans FIPA : celles ayant traits aux plate-formes qui vont héberger les agents, et celles décrivant le langage d'interaction ACL (Agent Communication Language).

Le but est bien de mettre en place un "middleware" agent qui fournira des services génériques d'accueil, d'identification, et de communication entre agents FIPA. Cet ensemble forme la plate-forme de référence de FIPA (voir figure 4.1). Dans FIPA, ces services de base peuvent être implémentés eux-mêmes sous la forme d'agents conformes à FIPA.

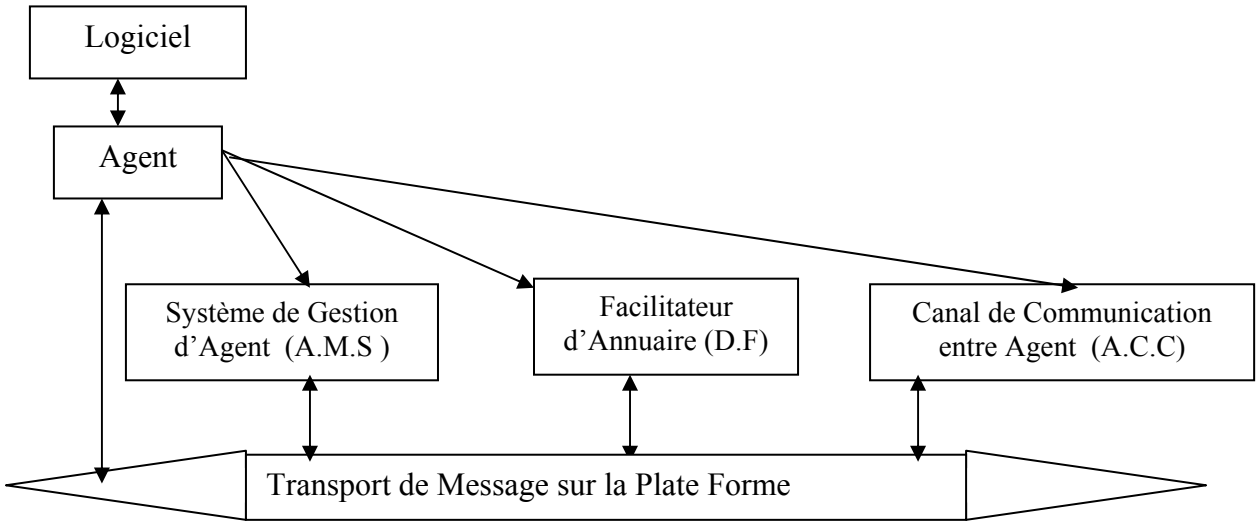


Fig 3.1 *Le modèle de référence pour une plate-forme Multi-Agents FIPA System (AMS)*

Dans la figure 4.1, on voit qu'il y a trois rôles principaux dans une plate-forme multi-agents FIPA :

- Le **Système de Gestion d'Agents** (Agent Management System - AMS) est l'agent qui exerce le contrôle de supervision sur l'accès à et l'usage de la plate-forme ; il est responsable de l'authentification des agents résidents et du contrôle d'enregistrements.

- Le **Canal de Communication entre Agents** (Agent Communication Channel - ACC) est l'agent qui fournit la route pour les interactions de base entre les agents dans et hors de la plate-forme ; c'est la méthode de communication implicite qui offre un service fiable et précis pour le routage des messages ; il doit aussi être compatible avec le protocole IIOP pour l'interopérabilité entre les différentes plates-formes multi-agents.

- Le **Facilitateur d'Annuaire** (Directory Facilitator - DF) est l'agent qui fournit un service de pages jaunes à la plate-forme multi-agents. Il faut remarquer qu'il n'y a aucune restriction sur la technologie utilisée pour l'implémentation de la plate-forme : émail, basé sur CORBA, applications multi-threads Java, etc.

Le standard spécifie aussi le **Langage de Communication d'Agents** (Agent Communication Language - ACL). La communication des agents est basée sur l'envoi de messages. Le langage FIPA ACL est le langage standard des messages et impose le codage, la sémantique et la pragmatique des messages. La norme n'impose pas de mécanisme spécifique pour le transport interne de messages. Plutôt, puisque les agents différents pourraient s'exécuter sur des plates-formes différentes et utiliser technologies différentes d'interconnexion, FIPA spécifie que les messages transportés entre les plates-formes devraient être codés sous forme textuelle. On suppose que l'agent est en mesure de transmettre cette forme textuelle. La norme FIPA préconise des formes communes pour les conversations entre agents par la spécification de protocoles d'interaction, qui incluent des protocoles simples de type requête-réponse, mais aussi des protocoles spécifiques aux agents comme le réseau contractuel et les enchères anglaises et hollandaises.

Il faut noter qu'à l'heure actuelle, un débat a lieu au sein de FIPA sur le sujet de "l'agentification" de ces trois services. Initialement, dans les spécifications FIPA'97 [17], ces trois services étaient des agents à part entière, au sens FIPA. Ils communiquaient en FIPAACL et étaient des intermédiaires obligés. Par exemple, pour envoyer un message d'un agent FIPA sur une plate-forme à un autre sur une plate-forme distante, il faut déjà faire suivre le message à l'ACC local. Celui-ci transmettrait alors son message à son homologue.

Ce mécanisme a été critiqué pour son inefficacité et sa lourdeur : une interaction ACL n'étant pas quelque chose d'anodin du point de vue traitement, on double quasiment toutes les interactions par des interactions contraignantes pour des tâches purement administratives. Voici par exemple le message nécessaire pour envoyer un message d'un agent FIPA agent-a à un autre agent-b si l'on observe scrupuleusement la norme :

```
(request
    :sender agent-a
    :receiver acc
    :content (action acc
        (forward
            (request
                :sender agent-a
                :receiver agent-b
```

:content

.....

)))

:protocol fipa-request

:language sl0)

L'approche actuelle est donc de considérer l'ACC comme un service non-nécessairement agent, qui peut être décrit en termes d'objet à l'usage de l'AMS ou le DF local et qui puisse éviter le déploiement d'une architecture de traitement lourde pour ce service de base. On y perd un mécanisme d'interaction unifié et extrêmement générique - FIPA-ACL - mais on facilite l'intégration potentielle avec des infrastructures de communication pré-existantes et on allège un peu les plate-formes. La mise en oeuvre est bien plus directe, mais l'interaction avec ces services se fait toujours via un protocole d'interaction ACL.

On peut quand même s'inquiéter de ces dérives par rapport à l'idée originelle de FIPA, qui avait le mérite de la cohérence du tout-agent.

L'autre élément clé des implémentations FIPA est ACL. ACL a été dès le départ vanté comme étant un "meilleur KQML", car formellement défini, plus facilement validable, et doté de protocoles d'interactions [19]. Cela a été remis en question par la communauté KQML assez rapidement [26], qui a ensuite proposé une sémantique formelle pour KQML. Après les premières expériences, il semblerait que les espoirs mis dans ACL ne sont pas tous vérifiés [16].

Pour terminer sur ce bref aperçu de la communauté FIPA, il faut signaler les changements de mode opératoire et de cible que connaît cet organisme. Dans les premiers temps (de 1996 à 1998), des spécifications étaient établies chaque année, puis éventuellement amendées l'année suivante. Depuis 2000, le processus est plus itératif, et se rapproche du fonctionnement d'autres organismes comme l'IETF [27] : des spécifications préliminaires sont établies, testées en vraie grandeur, puis seulement adoptées comme standards, ou bien automatiquement considérées obsolètes si rien n'arrive dans un certain délai. Cela ouvre la voie à l'expérimentation et laisse un plus grand champ d'action à la FIPA. On pourra comparer les soixante-dix spécifications en cours actuellement aux sept de 1997 et cinq de 1998.

De façon comparable, l'architecture FIPA a été affinée récemment pour aller dans le sens d'une plus grande modularisation. Nous l'avons vu avec la levée de la contrainte d' "agentification" de l'ACC. De même, CORBA/IIOP n'est plus un pré requis absolu, en ces temps de téléphones mobiles et protocoles WAP. L'encodage des messages peut être fait en s'appuyant sur d'autres formats que l'ASCII pur, pour ouvrir la voie à XML, Unicode ou à des formats compacts. Tout ceci va dans le sens d'une plus grande variabilité des

domaines de déploiement des applications FIPA, même si le modèle de base de l'agent reste inchangé et conditionne en fait fortement le style d'application.

La plate-forme Emorphia FIPA-OS

Pour illustrer comment ces spécifications peuvent se concrétiser, nous prenons l'exemple d'une des implémentations existantes : FIPA-OS, une plate-forme de validation des spécifications FIPA élaborée sous la houlette des laboratoires de Nortel Networks. En particulier, elle a été le premier outil à mettre correctement en œuvre les parties obligatoires des spécifications. Elle a été clairement positionnée comme un moyen d'accroître la visibilité et l'utilisation de FIPA.

Son architecture est un modèle en trois couches : transport, modèle et agent. Elle permet d'accueillir plusieurs modèles d'agents à partir du moment où le modèle de base de FIPA est respecté. Ces agents utiliseront implicitement les services FIPA classiques.

Des éléments de découplage sont néanmoins présents dans les couches de transport, qui ne sont pas forcément restreintes à CORBA/IIOP et aux mécanismes de persistance d'agents.

Les modèles d'agents de bases sont conçus pour accueillir directement des interactions conformes à ce qu'attend FIPA. C'est en particulier le respect d'ACL et des protocoles d'interaction qui y sont associés, mais également l'utilisation des langages de contenus définis par FIPA : SL et CCL[9].

2.3 Analyse et discussion

Le plus souvent, la plate-forme se confond avec son application : recherche d'information pour Infosleuth [28], équilibrage de charge sur Challenger [29], même si la volonté de généricité est affirmée, il est souvent difficile d'évaluer ce point hors d'un corpus d'applications conséquent. D'autre part, les environnements de conception, développement, et exécution sont définis dans la majorité des cas de façon rigide, alors qu'il serait plus intéressant d'avoir un découplage et une modularisation de ces différents aspects pour permettre de spécialiser les outils en fonction du domaine ou des conditions de déploiement (environnement de test, serveur aveugle, ...). De plus, ces plate-formes posent souvent un problème d'outillage des applications : on ne dispose que rarement d'outils d'analyse du fonctionnement des agents ou de la plate forme, et lorsque c'est le cas, il n'est pas réellement possible de construire des mécanismes d'observation spécifique sans avoir à intervenir sur le cœur de la plate-forme.

Mais le principal reproche que l'on pourrait formuler à l'égard des plate formes multi agents est leur spécialisation : la conception d'agent est centrée sur un modèle particulier, ce qui induit forcément une homogénéité sur les applications qui peuvent y être hébergées. Certaines plate-formes échappent néanmoins à cet écueil, comme DIMA[30], Geamas [31] ou Swarm [22] qui reposent volontairement sur des modèles plus génériques.

---

[9] L'originalité de CCL [33], par rapport à des spécifications comme KIF, est son expression sous la forme d'un CSP.

La contrainte d'un modèle rigidifié d'agent entraîne souvent un écart par rapport aux choix initiaux du concepteur. Cela pourrait être anodin dans le cas d'un domaine établi, mais cela s'avère particulièrement gênant dans le cas des systèmes multi-agents où les architectures, les modèles, ou même les définitions de base ne sont toujours pas véritablement affirmées.

Même dans le cas de FIPA, qui essaie pourtant d'aboutir à une interopérabilité entre systèmes (via des spécifications), on se rend compte que l'architecture des systèmes est en fait contrainte par deux fois. D'une part le modèle de référence d'un agent et de son langage de communication oriente très fortement le concepteur vers un style particulier d'architecture (par exemple, exprimer un système en vue d'une résolution par émergence est très malaisé). D'autre part, l'implémentation particulière de la plate-forme en elle-même va rajouter un modèle de comportement spécifique. Le développeur d'agent n'a d'autre choix que de s'y plier, et d'évaluer soigneusement l'adéquation entre son problème et l'architecture qu'il ne pourra guère adapter à ses besoins.

Pour finir, on notera qu'un des problèmes majeurs du monde des plate formes agents n'est peut-être ni technique, ni conceptuel, mais plutôt lié au contexte. L'atomisation du travail sur les plate formes et la très grande spécialisation des modèles n'encouragent guère l'émergence d'une communauté d'utilisateurs. Comme l'a bien mis en évidence Gasser, les faibles possibilités de partage de modèles, d'agents, ou d'expérience entre concepteurs freinent la généralisation des applications multi-agents, une pédagogie de ces systèmes, et l'apparition d'infrastructures communes.

3 La plate forme multi-agents MadKit

3.1 Introduction

Dans ce cette première partie du chapitre, nous présentons l'architecture de MadKit (pour "Multi-Agent Development Kit"), une plate-forme générique de conception et d'exécution de systèmes multi-agents. Cette plate-forme est basée sur un modèle organisationnel plutôt qu'une architecture d'agent ou un modèle d'interaction spécifique.

L'utilisation de groupes et de rôles associés à des agents est mis en œuvre tant en tant qu'outil de modélisation et de conception pour les développeurs de systèmes multi-agents, que de principe d'architecture de la plate-forme elle-même.

Cette architecture est basée sur un noyau agent minimal découplé de tout modèle individuel d'agent. Dans cette plate-forme, les services classiques de passage de message distribués, de migration ou de surveillance sont fournis au meta-niveau par des agents spécialisés afin d'obtenir un maximum de flexibilité. Une interface graphique componentielle et découplée du noyau et des agents permets de supporter différentes modes d'utilisation et d'exploitation de la plate-forme.

3.2 Modèle organisationnel

Nous présentons ici très rapidement le modèle de description organisationnel qui structure la plate-forme MadKit. Ce modèle a été décrit en détail dans le chapitre 4. Sa sémantique opérationnelle basée sur le π calcul.

3.2.1 Agent

Quasiment aucune contrainte n'est posée sur l'architecture interne ou le modèle de comportement de l'agent. L'*agent* est simplement décrit comme une entité autonome communicante qui joue des *rôles* au sein de différents *groupe*s. La très faible sémantique associée à l'agent dans ce modèle est tout à fait volontaire. L'optique est bien de laisser la définition d'agent en retrait, non seulement pour ne pas prendre part dans le débat classique de "qu'est-ce qu'un agent", mais surtout pour laisser au concepteur toute liberté pour choisir l'architecture interne appropriée à son domaine applicatif. Cette liberté se retrouvera dans les principes de MadKit

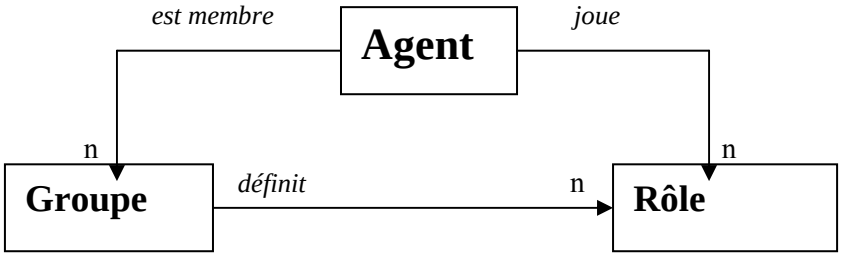


***Fig 3.2** – Modèle organisationnel Rôle-Groupe-Agent*

3.2.2 Groupe

Le *groupe* est la notion primitive de regroupement d'agents. Chaque agent peut être membre d'un ou plusieurs groupes. D'une façon un peu simpliste, un groupe peut être vu comme un moyen d'identifier par regroupement un ensemble d'agents. Plus classiquement, associé au rôle, il définira la structuration organisationnelle d'un SMA usuel. Les différents groupes peuvent se recouper librement.

3.2.3 Rôle

Le rôle est une représentation abstraite d'une fonction, d'un service ou d'une identification d'un agent au sein d'un groupe particulier. Chaque agent peut avoir plusieurs rôles, un même rôle peut être tenu par plusieurs agents, et les rôles sont locaux aux groupes. L'hétérogénéité des situations d'interaction est rendue possible par le fait qu'un agent peut avoir plusieurs rôles distincts au sein de plusieurs groupes, les communications étant clôturées par les groupes.

3.3 Architecture Générale

3.3.1 Principes

La structure organisationnelle que nous venons de décrire est implémentée au cœur de la plate-forme MADKIT, tant pour fournir un modèle organisationnel aux SMA exécutés que pour le fonctionnement interne du système. L'utilisation de groupes d'agents pour la structuration de plate-formes a été proposée dans d'autres

architectures, comme par exemple [1], bien que ces mécanismes sont limités aux problèmes de migration et ne possèdent pas ce trait distinctif de prise en compte de groupes et rôles simultanés pour un agent donné.

En plus de ce modèle d'organisation, MadKit est basé sur trois principes :

- Architecture à micro-noyau
- Agentification systématique des services
- Découplage applicatif entre noyau, agents et application d'accueil.

Concrètement, MadKit est un ensemble des packages Java qui implémentent le noyau agent, diverses bibliothèques de base de messages, d'agents et de sondes. Le choix de Java a été fait très tôt pour des raisons de portabilité, mais également pour la richesse des bibliothèques de bases, les possibilités de sérialisation d'objets et de présence de réflexivité, même sommaire.

Le principe essentiel de MADKIT est d'utiliser partout où cela est possible la plate-forme pour sa propre gestion et exécution. Tous les services hors ceux assurés par le micro-noyau sont implémentés par des agents à part entière. Ceci vient à la fois d'un souci d'unicité de modèle, mais également de mise à l'épreuve des SMA comme modèle de programmation. Cela le rapproche également fortement des approches meta en architectures objets [20] à ceci près que nous plaçons la relation au meta entre agents applications et "systèmes" (communication, ordonnancement, surveillance...).

MadKit n'est pas une plate-forme agent dans le sens classique, centrée autour d'un modèle particulier d'agent ou d'interaction. La taille réduite et les fonctionnalités du noyau de base, la neutralité des types agents, associée à ce principe de services agentifiés et de découplage par rapport aux applications "hôtes" permet en fait d'obtenir toute une gamme d'environnements d'exécutions aux finalités parfois complètement opposées.

3.3.2 Le micro-noyau agent

Le "micro-noyau" de MADKIT est un environnement d'exécution d'agents de taille réduite (moins de 40 Ko). Le terme "micro-noyau" est utilisé ici intentionnellement comme référence au modèle des systèmes d'exploitation à micro-noyau. On pourrait directement plagier leur principe fondateur [17] *« incorporating in micro-kernels a small number of key facilities that allow the efficient deployment of system services."* en remplaçant "system" par "agents".

3.3.2.1 Fonctionnalités

Le noyau ne prend en charge que les fonctions suivantes :

Gestion des groupes et rôles locaux Comme l'interopérabilité et les mécanismes d'ex-tension de MadKit se basent sur le modèle agent-groupe-rôle, il est essentiel que cette information organisationnelle soit gérée au plus bas niveau afin que tous les agents, quels que soient leurs modèles individuels, y aient accès. Le

noyau a la responsabilité de maintenir une information correcte sur les membres des groupes et les rôles tenus. Il vérifie également si les requêtes faites sur le système de groupes et rôles sont acceptables (c'est à dire en évaluant ou délégant les fonctions de rôles).

Gestion du cycle de vie des agents Le noyau gère également le lancement (et éventuellement l'arrêt) des agents et maintient les tables de références sur les objets d'implémentation. Il est également le gestionnaire des informations administratives sur les agents (possesseur, modalités de créations, ...) et donne un identifiant garanti unique à chaque agent. Cet identifiant, l'AgentAddress, est forgé à partir de l'addresse du noyau MadKit et l'identification de l'agent sur le noyau ; il peut être rapproché de la notion de GUID de FIPA. La forme de cet identifiant peut également être redéfinie pour faciliter l'intégration avec d'autres plate forme agent.

Passage de message local Le noyau a la responsabilité de l'aiguillage et des distributions de messages entre agents **locaux** (s'exécutant sur le noyau). Le passage de message au plus bas niveau revient à de simples échanges de références pour pouvoir facilement implémenter différentes sémantiques de passage de message à un niveau supérieur.

Le noyau lui-même est en fait encapsulé dans un agent particulier, le Kerne-lAgent qui est créé automatiquement à l'amorçage de la plate-forme. Il agit comme représentant du noyau dans le monde agent : toutes les demande de surveillance ou de contrôle sont alors écrites comme une interaction inter-agent et préservent l'unicité du modèle.

3.3.2.2 Mécanismes de passage au meta-niveau

Le noyau lui-même est extensible par des "hooks", des points d'interceptions sur ses opérations. Un agent autorisé (par exemple en ayant rempli les conditions d'accès au groupe "système") a la possibilité de demander un accès à l'un de ces hook. La demande se fait par une interaction agent avec le KernelAgent mentionné plus haut. Ces hooks sont un mécanisme de publication/abonnement qui permet soit la surveillance, soit l'interception des opérations. Quasiment toutes les fonctionnalités de base du noyau (lancement d'un agent, accès et modifications à la structure organisationnelle, passage de message) sont concernées. Cela permet donc d'écrire facilement des agents d'analyse d'un SMA en fonctionnement (dans le style de [19]), ou bien des agents étendant les fonctionnalités de base de la plate-forme. Le nombre réduit de fonctionnalités bien définies favorise la modification du comportement des opérations de base tout en préservant au maximum leur sémantique.

Pour toute opération noyau, il existe deux types de hooks :

Outils La classe de base des agents permet également de manipuler une éventuelle interface graphique associée à l'agent, les flots d'entrée/sortie, etc.

3.3.3 Structure et fonctions d'un agents

La classe de base d'un agent MadKit (AbstractAgent), définit quelques fonctionnalités de base pouvant être nécessaires dans les modèles classiques.

3.3.3.1 Fonctionnalités

Les fonctions associées à tout agent sont :

- **Cycle de vie** L'agent dispose de quatre états (création, activation, exécution, et destruction), et a la possibilité de démarrer d'autres agents sur le noyau local (et les désactiver par la suite). Par contre, aucun mécanisme concret d'exécution n'est défini à ce niveau.

- **Communication** La communication est implémentée sous forme de passage de message asynchrone, soit d'agent à agent identifié par leur AgentAddress ou leur groupe et rôle, soit sous la forme d'une diffusion à tous les teneurs d'un rôle dans un groupe donné.

- **Organisation** Tout agent dispose de primitives permettant d'observer son organisation locale (connaître les groupes et rôles courants) et d'y agir (prise de rôle, entrée et retraits de groupes).

- **Outils** La classe de base des agents permet également de manipuler une éventuelle interface graphique associée à l'agent, les flots d'entrée/sortie, etc.

3.3.3.2 Messages

Les messages sont définis par héritage à partir d'une classe de base qui ne définit que la notion d'émetteur et destinataires. Certain messages de base sont néanmoins fournis (encapsulation d'une chaîne ou d'un objet arbitraire, messages KQML et FIPA-ACL, messages XML), mais restent extensibles pour s'adapter à tout protocole d'interaction.

3.3.3.3 Threads et moteur synchrone

Pour faciliter les implémentations de modèles classiques d'agents autonomes, cette classe de base est déclinée vers une implémentation associant un thread d'exécution à un agent.

Néanmoins, pour certains types de systèmes, en particulier les SMA réactifs, il est nécessaire de gérer un grand nombre d'agents (jusqu'à centaines de milliers) de granularité assez fine, et de contrôler leur ordonnancement. Cela est quasi-impossible si l'on se repose sur un mécanisme de thread à cause de la surcharge induite. Dans ce cas, on utilise ces agents "synchrones" via des agents externes qui vont définir leur politique d'exécution synchrone. Des agents d'observation peuvent être ajoutés pour avoir une vue globale sur certaines propriétés, à l'instar des plates-formes classiques de simulation comme Swarm [22].

3.4 Spécifités de la plate forme MadKit

3.4.1 Agentification des services

A l'opposé de plate-forme plus monolithiques, MadKit utilise donc des agents pour implémenter au niveau meta des services comme le passage de message distribué, la migration d'agent, la sécurité d'organisations d'agents et divers aspects de gestion du système ; et ce éventuellement à l'aide des mécanismes

d'extension du noyau. Comme ces services sont mis en oeuvre par des agents, et décrits par la structure organisationnelle, les interactions entre ces agents systèmes, le noyau, et les agents de l'applications sont décrits dans le modèle agent-groupe-role. Ceci implique qu'un agent offrant une fonctionnalité donnée peut être remplacé par un autre de façon transparente (et éventuellement durant le fonctionnement d'une application), à partir du moment où les groupes, rôles et interactions sont respectées.

Par exemple, des développeurs extérieurs ont échangé nos agents de synchronisation de l'information organisationnelle par d'autres qui fournissaient une interface avec des annuaires distribués.

Les agents du SMA applicatif et les autres agents "système" n'ont pas eu à être modifiés et n'ont en fait pas remarqué l'échange.

Baser les services sur des agents décrits en terme de rôles a également l'effet de faciliter une montée en puissance. Un agent tenant plusieurs rôles peut, au fur et à mesure que les besoins de l'application grandissent, déléguer dynamiquement à de nouveaux agents certains de ses rôles pour réduire sa charge.

De plus, la structuration en groupe agissant souvent en "masquage" et délégation d'un SMA complexe, un rôle ne correspond pas forcément à une mise en oeuvre par un et un seul agent. Par exemple, un agent fournissant un rôle de communication système peut être un simple *représentant* d'un groupe d'agents spécialisés prenant en charge chacun un protocole.

Avoir découplé les services du noyau permet aussi de ne pas alourdir les applications : deux plate-forme exécutant un SMA localement ou en distribué ne diffèrent que par le lancement ou non des agents de communication.

3.4.2 Applications hôtes

Le noyau de la plate forme ne fournit aucune interface graphique pour l'utilisateur. Il est par contre possible de lui associer une application hôte qui la mettra en oeuvre.

De plus, chaque agent peut définir un objet graphique pour sa représentation, d'un simple bouton à une application classique encapsulée. L'hôte aura alors la charge, au lancement d'un agent, de décider de la mise en place de l'objet graphique défini par l'agent (dans des fenêtres séparées, combiné avec l'interface d'un autre agent, etc..) et de les gérer au niveau global.

Associé au principe d'agentification des services, cela permet de modulariser complètement la plateforme. Par exemple, les variétés suivantes de MadKit ont été mises en oeuvre :

- La G-Box, un environnement graphique complet de développement et de test de systèmes multi-agents. Elle permet de contrôler le cycle de vie des agents, charger des SMA dynamiquement, et de manipuler graphiquement les accointances ou bien les tables de groupes et de rôles.

- Un simple “booter” pour démarrer uniquement une liste d’agents donnés, qui peut être utilisé pour la distribution finale d’une application agent ou le mise en place d’agents (facilitateurs, annuaires, interprètes) sur un serveur.

- Une applet qui empaquète le noyau agent et un SMA applicatif, pour l’exécuter dans un navigateur distant.

- Une application qui embarquerait un noyau MadKit est des agents applicatifs pour prendre en charge les aspects dynamiques ou coopératifs, indistinguables d’une application “classique”

- Pour valider notre approche, nous également avons développé une version de MadKit utilisant le même noyau que la plate-forme complète, et capable de tourner sur des assistants personnels de type Palm. Un SMA applicatif réduit (gestionnaire de rendez-vous) a été implémenté de façon classique et mis en place avec un agent de communication spécifique (par infra-rouge). Le prototype complet fonctionne malgré les limitations fortes en puissance de calcul et occupation mémoire (moins de 64 Ko disponible pour l’application).

3.5 Discussion

Dans ce chapitre, nous avons présenté une plate-forme multi-agent basée sur un modèle organisationnel, l’idée qu’il est possible d’intégrer au sein d’un même outil une grande variété d’architectures d’agents et de modèles de communication.

Cependant cette plate forme est plutôt un environnement d’exécution d’applications hétérogènes que d’être une plate forme multi-agents pour la simple raison est que l’agent, concept central dans les systèmes multi-agent, n’est guère définit de manière explicite

4 La plate forme OpenCybel

4.1 Introduction

Nous allons voir dans cette deuxième partie de l’état de l’art des plates formes multi-agents, de manière sommaire en quoi consiste cette plate forme

OpenCybele est un environnement construit au-dessus de la plate-forme Java™ 2 pour le contrôle et l’exécution des agents. L’exécution des agents est dirigée par évènements et contrôlée par OpenCybele.

OpenCybele adopte une architecture sur plusieurs couches promouvant la possibilité d'ajouter des services plug-and-play, tels que les gestions des erreurs, des threads, des évènements (internes ou provenant de l'interface), des horloges, la communication, la concurrence, … On accède aux services par le biais des GSI (Genereric Service Layer).

4.2 Le modèle d'agent

4.2.1 Qu'est-ce qu'un agent

Les définitions donnés pour un agent et une activité sont les suivantes :

Un Agent est un groupe d'activités dirigées par des évènements, ces activités partagent des données, des threads et une structure de concurrence.

Une Activité est un objet actif qui opère sur des données internes. Les données internes et les méthodes sont encapsulées, c'est à dire qu'elles ne sont pas accessibles directement par les autres objets.

Schématiquement un agent dans OpenCybel est un ensemble d'objet comme suit :

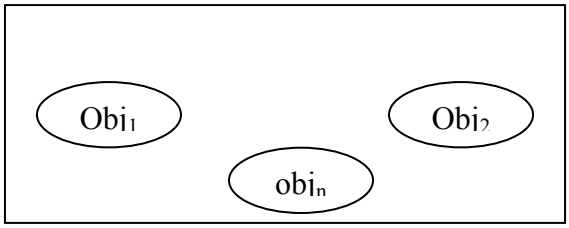


Fig 3.3 ***Modèle d'agent dans OpenCybel***

4.2.2 La Programmation Centrée sur les Activités (ACP)

Une décomposition du domaine d'application conduit à la modélisation de fonctionnalités ou d'entités physiques sous forme d'Agents. Les Activités (objets) sont les composants de base des agents. De la même manière qu'un système est composé d'agents autonomes, un agent est composé d'activités autonomes. Aucune activité ne peut appeler directement une méthode d'une autre activité.

4.3 La notion de Programmation Centrée sur les Activités

4.3.1 Le Modèle de Concurrence

Dans OpenCybel les activités soient dirigées par les évènements, l'exécution d'une activité dépend de l'état des autres activités avec lesquelles elle a une relation de concurrence ou une dépendance d'exécution.

4.3.2 Les différents états d'exécution

Une Activité peut être dans un des six états suivants :

- Runnable : C'est l'état initial de toutes les activités, cette activité est en attente d'évènements
- Active : l'activité exécute une méthode agissant sur son état interne
- Hold : L'activité attend une autre activité pour retourner à son état Runnable. (qu'il y aient ou non des évènements en attente)
- Event-Blocked : Le thread dans lequel l'activité s'exécute est bloqué si l'activité a envoyé un message et attend une réponse
- Activity-Blocked : Lorsqu'une activité attend qu'une autre activité fille se termine
- Done : Lorsque l'activité n'attend plus aucun événement

4.3.3 Les Relations de Concurrence entre activités

Entre deux activités, il peut exister une des relations suivantes :

- Concurrent : Deux activités sont dans dites concurrentes si l'état d'une activité n'affecte pas l'état de l'autre
- Event Serialized : Une activité est dans l'état Active, Event-Blocked ou Activity-Blocked ce qui place l'autre dans l'état Hold (relation commutative)
- Block and Wait : Lorsqu'une des activités attend que l'autre se termine

4.3.4 Transition d'Etat d'Activités et Modèle de Concurrence

Lorsqu'une activité parent lance une activité fille :

- le parent doit indiquer quelle relation de concurrence doit exister
- le parent doit indiquer si l'activité fille doit hériter de l'environnement (ensemble de relations de concurrence héritées) du parent

4.4 Discussion

dans cette plate forme on a vu qu'un agent est constitué d'un ensemble d'activités où chaque activité possède un ensemble de méthodes et un ensemble d'attributs exactement comme un objet. Donc un agent ici est un regroupement d'objet. Il se peut aussi qu'on trouve un agent n'ayant qu'une seule activité et dans ce cas on est beaucoup plus dans le formalisme objet que celui d'agent.

5 Conclusion

Nous espérons à travers ces deux exemples de plate forme multi-agents MadKit et OpenCybel avoir mis l'accent sur les difficultés et obstacles que rencontre généralement les concepteurs de plate forme orientée agents.

Ces difficultés sont dues essentiellement pour le manque de clartés des concepts propre au formalisme agent; comment se fait il qu'après plusieurs années de recherches dans ce domaine on arrive toujours pas à donner une définition unificatrice du terme agent et tous les concepts liés reste par conséquent ambigus et flous.

Chapitre IV

Les Modèles Organisationnels des Systèmes Multi-Agents

Résumé : *Lorsqu'on assemble plusieurs agents la question se pose de savoir comment les organiser. Par conséquent, la notion d'organisation joue un rôle essentiel dans les SMA. L'organisation d'un système peut être dynamiqu; Dans ce cas, l'étude de sa dynamique organisationnelle s'impose. La notion d'organisation est intensivement étudiée dans le domaine de l'IA. Elle constitue l'une des quatre dimensions retenues dans le domaine de recherche sur les SMAs. Ces dimensions sont, d'après Demezeau [34], l'agent (A), l'environnement (E), l'interaction (I) et l'organisation (O).*

1 Introduction

Lorsqu'on assemble plusieurs agents la question se pose de savoir comment les organiser. Par conséquent, la notion d'organisation joue un rôle essentiel dans les SMA. L'organisation d'un système peut être dynamique. Dans ce cas, l'étude de sa dynamique organisationnelle s'impose.

La notion d'organisation est intensivement étudiée dans le domaine. Elle constitue l'une des quatre dimensions retenues dans le domaine de recherche sur les SMAs. Ces dimensions sont d'après [34] : l'agent (A), l'environnement (E), l'interaction (I) et l'organisation (O).

Jacques Ferber [18] indique que «L'organisation est, avec l'interaction, l'un des concepts de base des systèmes multi-agents. »

Toutefois, il s'agit d'une notion complexe et difficile à définir. Le mot « organisation » est utilisé pour indiquer l'agencement de relations entre agents, un concept abstrait.

Notons qu'il existe une relation intime entre un système et son organisation. La disposition des relations entre agents détermine l'évolution future du système. Puis, le système même détermine l'évolution de son organisation. Par exemple, les relations entre agents, suivant le modèle d'une hiérarchie, déterminent l'évolution d'une unité militaire.

Toutefois, c'est l'unité même qui génère (soutient) la hiérarchie : les deux composants ne peuvent pas exister indépendamment. Cette boucle entre système et organisation peut également exister à un niveau supérieur. Par exemple, une bonne unité militaire respectera toujours le modèle d'une hiérarchie, indépendamment de l'évolution des agents et des relations entre agents. Le type d'organisation détermine comment l'organisation concrète peut évoluer.

La figure 4.1 montre les boucles entre les niveaux d'organisation différents. Les sens différents du terme « organisation » peuvent provoquer des malentendus, surtout parce que la terminologie développée pour les distinguer n'est pas normalisée.

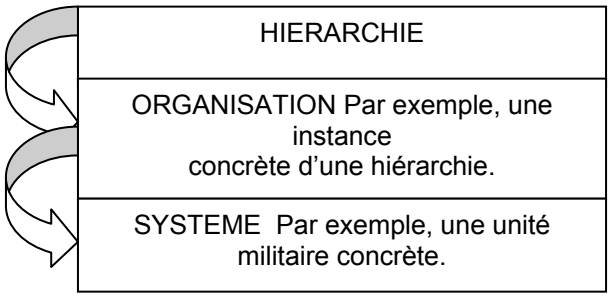


***Fig 4.1**: Les niveaux différents d'organisation.*

Dans le domaine des SMA, les notions de *rôle* et de *tâche* sont directement liées à celle d'organisation. En essence, l'organisation est vue comme un mécanisme pour gérer la distribution de **rôles** et/ou de **tâches** sur

les différents agents, afin de garantir une **fonctionnalité globale**. Par exemple, la notion de rôle est essentielle dans les méthodologies récemment développées telles que Cassiopée [35] ou Aalaadin [18].

La manipulation de rôles exige des capacités cognitives, impliquant que soit l'organisation soit déterminée par le concepteur, *soit* qu'elle soit déterminée par des agents cognitifs. Si on veut construire des systèmes dans lesquels les agents mêmes changent leur organisation, on est obligé d'utiliser des agents cognitifs. Si on veut utiliser uniquement des agents réactifs, on est obligé d'imposer une organisation statique sur les agents. Notons que – en général - il est très difficile de réaliser la traduction automatique, dans les deux sens, entre les rôles et les propriétés des agents au niveau réactif.

2 l'organisation des systèmes classiques : objets et Modèles de rôles

En fait, ces approches sociales ne sont pas l'apanage des systèmes multi-agents. Identifier fonctionnellement, structurer des systèmes sont des besoins qui en dépassent largement le cadre : assez logiquement, on en retrouve les principes dans d'autres modèles, principalement les bases de données et les modèles objets.

Mais la perspective n'est pas exactement la même. Dans un contexte objet, une distinction est à faire entre la notion de classe, définissant les capacités d'objets individuels et isolables, et le rôle, rattaché à des caractéristiques partagées de position ou de responsabilité d'objets dans un système donné. Contrairement à ce que l'on a pu voir dans les approches agents, l'interprétation d'un rôle est très peu ambiguë : on veut avant tout conserver les bonnes propriétés de typage dans un système.

Le niveau agent correspond à une modélisation organisationnelle au-dessus des objets, où ceux-ci sont agrégés pour former des entités à la dynamique variable.

Mais pourquoi faire apparaître ce genre de structuration ? Si l'on part du principe qu'établir une architecture, c'est à la fois concevoir et construire une structure (à partir d'entités et de modèles élémentaires) et répartir les fonctionnalités du système entre les différents composants, on se rend compte que l'on retrouve nos préoccupations initiales : exprimer aspects fonctionnels et structurels. Comment faire le lien entre les modèles sociaux et les architectures plus classiques ? Revenons à la base, aux besoins qu'expriment [18] au sujet des frameworks de taille conséquente :

Structure **:** l'interface d'un ensemble d'entités de base ainsi que les structures statiques ou dynamiques qui les mettent en relation dans une composition.

Fonction **:** l'association fonctionnelle entre le problème et la structure, par identification pure ou raffinement de définitions initiales.

Abstraction **:** l'identification (et le nommage) d'une composition d'éléments ayant une certaine structure et une certaine fonctionalité.

Réutilisation **:** la capacité à être appliqué plusieurs fois en des contextes différents.

Le problème vient en fait de la liaison à établir entre la conceptualisation (par l'étude du domaine) et l'implémentation (par la construction de l'architecture). Les deux aspects sont la plupart du temps exprimables simultanément d'un point de vue opératoire. Imaginons naïvement que l'on puisse simplement hériter d'une classe "de domaine" pour former une classe "d'architecture". L'héritage aura pour effet que deux instances de différentes sous-classes ne sont pas identiques même si elles le sont du point de vue conceptuel. Une solution consiste à appeler le concept de rôle à la rescousse pour permettre à un objet de s'intégrer dans différentes couches applicatives. Le rôle est alors assimilable à un point de vue (spécifique au client) sur un objet jouant ce rôle.

On trouvera une bonne synthèse des motivations justifiant des modèles à base de rôle dans [18] . Ce travail a également l'intérêt d'être à mi-chemin entre approches objet et agents. Nous en reprendrons ici les points principaux.

On peut finalement voir un modèle de rôle comme l'identification d'un archétype, une structure récurrente d'objets à décrire comme un ensemble de rôles en interaction. En général, les modèles de rôles cherchent avant tout à capturer la manière d'interagir que vont avoir des objets concrets. A ce titre, ils complètent élégamment la vue abstraite et centrée sur l'élément, de la vue en classes. Les rôles vont alors être dynamiquement assignés à des objets dans une application. On peut ramener la notion de rôle à un "service" spécialisé que remplira l'entité (qu'on pourrait ensuite associer à l'idée de comportement pour un agent ou de type, pour un objet). Le rôle permet donc de voir le système d'une façon plus globale : un

Rôle a des collaborateurs, qui seront les autres rôles avec lesquels il interagit.

En résumé, quels sont les points caractéristiques des modèles de rôles ?

- les modèles de rôle mettent l'action sur l'interaction. Les classes donnent les capacités des objets individuels, mais le rôle s'occupe plus de la position et des responsabilités d'un objet dans une structure globale ou dans le système. Les modèle de rôles sont orthogonaux et complémentaires aux classes
- Les rôles permettent de décrire les systèmes ou sous-systèmes en termes de patterns d'interaction, ce qui offre un nouveau niveau d'abstraction à mi-chemin entre classe et instance.
- Les modèles de rôles peuvent être dynamiques (prise et abandon de rôle, évolution, transfert, ...)
- Le rôle lui même peut être, selon les modèles, traité au même niveau que les classes et objets, avec la possibilité de les instancier, généraliser, spécialiser ou agréger.

A noter que tout cela permet également de penser à la réutilisation en terme de rôles, par exemple pour élaborer de nouvelles définitions à partir de l'existant : par dérivation (avec une entité jouant un rôle dérivé devant pouvoir jouer aussi les rôles de base), par aggrégation (où un rôle peut être l'aggrégation d'autre rôles, masqués ou non), par relations (en posant des dépendances ou des équivalences entre différents rôles).

3 Modèles organisationnels des systèmes multi-agents

Dans cette section nous présenterons deux modèles organisationnels du domaines des systèmes Multi-agents. Il s'agit d'un modèle basé sur les concepts d'agent, groupe et de rôle, baptisé par leur concepteur ALAADIN, utiliser dans plusieurs plate forme multi-agents tel que MadKit. Le deuxième Modèle est basé sur les concepts d'Agent, rôle, compétence et tâche dén omé YAMAM qui se veut une alternative au premier, ce modèle est utiliser dans la plate forme multi-agents Phoenix.

3.1 ALAADIN : Agent-Groupe-Rôle

On a tendance à qualifier d'organisation des structures sociales stables, ou bien présentant des normes sociales explicites. Nous utiliserons ce terme d'une manière plus générale, pour définir des sociétés d'agents analysables en ces termes, quitte à préciser le domaine d'étude et y raffiner les définitions selon le cas. Notre modèle est basé sur l'association de trois concepts clés : l'agent, le groupe et le rôle, utilisés simultanément pour décrire des organisations concrètes d'agents. La figure 4.1 présente un diagramme UML de ce modèle de base.

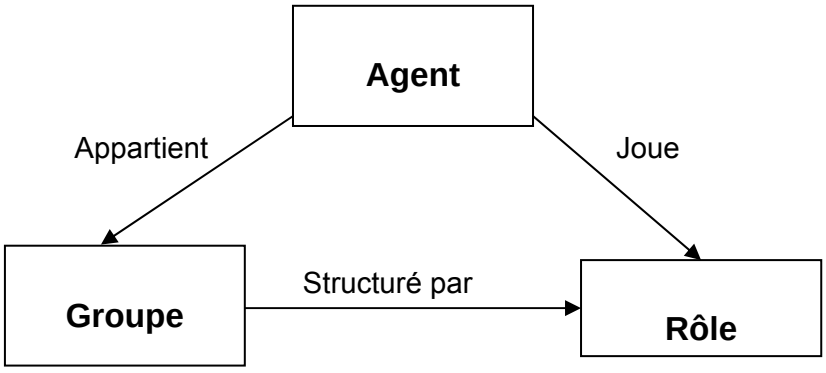


Fig 4.2 ***– Les trois concepts centraux***

On le résumera par la formule suivante :

Un agent peut intervenir dans plusieurs communautés (que nous appellerons 'groupes' dans notre modèle) en parallèle. Il peut jouer dans chacun de ces groupes un ou plusieurs rôles correspondant à ses activités ou

interactions. Ces rôles peuvent être portés par un nombre d'agents arbitraire, dépendant de la situation et des normes de l'organisation.

Voyons maintenant plus en détail ce que nous entendons ici par ces termes.

3.1.1 L'Agent

Dans ce modèle, aucune contrainte n'est posée ou pré requis sur l'architecture interne de l'agent et aucune supposition est faite quand au formalisme ou modèle particulier pour en décrire le comportement. Nous pourrions en donner une définition minimale, telle que :

Une entité agissante, autonome, et dont l'action s'exerce dans un contexte social.

Dans le modèle, l'agent est simplement décrit comme une entité autonome communicante qui joue des rôles au sein de différents groupes.

On pourrait reprocher a cette définition son flou, mais nous insistons sur le fait qu'elle est intentionnellement générale pour permettre à chaque concepteur de système de choisir, parmi les modèles classiques, le plus adapté à son application.

Précisons aussi dès maintenant que la nature de l'action sociale de l'agent n'est pas imposée. Ce peut être des mécanismes de coordination reposant sur des actes de langage ou des mécanismes de coopération non symboliques, comme par exemple le marquage d'un environnement. De la même manière, on n'explicite pas à ce stade le niveau de représentation de ce contexte social : il peut être internalisé par des connaissances de l'agent sur la société dans laquelle il s'insère, ou bien explicité uniquement au niveau du concepteur.

Ce sera par le biais de l'agent que nous ferons le lien avec les politiques d'interaction et de coopération, la dynamique des sociétés, et, à l'opposé, les architectures individuelles des entités.

Notons que même si nous évoquons la notion d'agent dans un contexte opérationnel, cela n'implique pas forcément que la définition de son environnement social soit uniquement faite à ce niveau. Ceci dépend avant tout des choix de représentation et d'architecture interne que fera un concepteur ou un analyste. L'agent est à ce stade une notion "en creux" faite pour être complétée par un modèle plus classique (d'inspiration cognitive, BDI, éthologique, économique, ...).

3.1.2 Groupe

Le groupe est définit comme la notion primitive de regroupement d'agents. Chaque agent peut être membre d'un ou plusieurs groupes. Dans sa forme la plus simple, un groupe peut être vu comme un moyen d'identifier par regroupement un ensemble d'agents.

Le groupe est avant tout un terme générique pour qualifier une communauté d'agents en relation (par interaction, par partage d'un environnement, par un but ou une ontologie commune, ...).

D'une manière plus évoluée, le groupe peut être vu comme un SMA usuel. Un point majeur de cette définition est que les différents groupes peuvent se recouper librement, c'est à dire que certains agents peuvent apparaître dans plusieurs groupes sans que l'on doive observer un strict modèle hiérarchique.

Bien que l'expression de la dynamique de ces groupes soit détaillée plus loin, nous pouvons préciser dès à présent que la structuration en groupe d'un ensemble d'agent peut varier avec le temps et que la genèse de l'organisation est vue avant tout comme le résultat de l'action des agents.

Prenons l'exemple d'un agent (humain) : il peut participer à diverses communautés. Il sera responsable dans une équipe de recherche, dans le même temps simple délégué dans le groupe qui se réactive tous les trimestres pour discuter d'orientations pédagogiques, ou encore disputer dans son temps libre des matchs de volley. Ce que l'on dégagera dans ces situations est la variété des situations d'interactions possibles, toutes très différentes dans leurs manifestations : par exemple, la reconnaissance d'intentions dans le match, la discussion libre et le vote dans une réunion, ou bien la structure assez codifiée de passivité ou de questions/réponses d'un séminaire. Cela n'empêche pas que ces différentes situations ont toutes leur cohérence, leur vocabulaire propre et leur modèle d'action et d'interaction.

3.1.3 Rôle

Le rôle est la représentation abstraite d'une fonction, d'un point d'interaction ou d'une identification d'un agent au sein d'un groupe particulier. Chaque agent peut avoir plusieurs rôles, un même rôle peut être tenu par plusieurs agents, et les rôles sont locaux aux groupes.

> *On appelle rôle la représentation abstraite d'une fonction du groupe pouvant contraindre le comportement de l'agent, et incarnée dans un ou des comportements spécifiques par l'entité.*

Le rôle va représenter ce que l'on attend comme comportement de l'agent dans l'organisation : travailler en coopération avec d'autres agents, et trouver son positionnement par rapport à l'organisation elle-même. Cela peut être une simple tâche à remplir vis à vis de l'application globale, mais également une relation à un statut ou une fonction dans l'organisation.

Cette représentation peut se traduire par une stratégie d'implémentation particulière pour les agents mettant en oeuvre potentiellement ces rôles.

Pour illustrer cela, prenons l'exemple maintenant classique des matchs de robots footballeurs [36]. On peut associer à un agent des rôles correspondant à sa position dans l'organisation : être capitaine de l'équipe, ce qui évoque plusieurs schémas d'interaction possibles (des indications plutôt impératives sur l'action à mener données aux autres joueurs). Cela n'empêche pas d'avoir d'autres rôles dans cette structure, dont celle de simple joueur, qui implique d'autres situations possibles (simplement dialogues ou information de démarquage, par exemple). D'autres dénotent la spécialisation par rapport à une tâche et ce que les autres agents peuvent attendre de lui, comme les capacités et devoirs d'un avant-centre. Ce qui n'empêche nullement, plus tard dans la partie,

qu'un autre rôle soit dynamiquement acquis pour faire temporairement le défenseur lors d'une offensive dangereuse.

3.2 YAMAM : Agent-rôle-tâche-Compétance

Dans cette section nous allons voir un autre modèle organisationnel, baptisé par ces concepteurs Yamam (Yet Another Multi-agent Modèl). Dans ce dernier l'agent n'est pas vu comme l'élément central du système. L'essence du modèle tient dans la notion de compétence qui permet la tenue d'un rôle.

3.2.1 L'Agent

Il n'y a pas d'a priori sur ce qui est un agent. Il reste un niveau de description conceptuelle, une entité plus ou moins autonome et communicante plongée dans un environnement dynamique. La généricité de l'agent est assurée suivant le principe suivant : un agent a des caractéristiques originellement réactives. Un agent réactif n'est créé que lorsque des compétences ne pourront êtres ajoutées dynamiquement. Par contre, si on veut que l'agent évolue dans le temps il faut instancier un agent cognitif héritant ses caractéristiques d'un agent réactif. Un agent cognitif saura utilisé des compétences réactives et/ou des compétences cognitives et pourra apprendre de lui-même de nouvelles compétences.

3.2.2 Rôle

Un rôle représente généralement un service, une fonction ou une forme d'identification d'un agent. Un agent prend en charge un ou plusieurs rôles. Un rôle est toujours local à un environnement et repose sur une liste de tâches à accomplir séquentiellement, parallèlement, avec ou sans répitition. Un rôle implique donc la tenue d'une ou de plusieurs tâches. Dans cette optique un agent peut tenir un rôle à condition qu'il sait exécuter toutes les tâches impliquées dans le rôle et donc s'il possède les compétences nécessaires pour cela. Un rôle implique souvent la planification de plusieurs tâches et pour cela l'agent doit posséder une compétence de planification de tâches.

3.2.3 Tâches

Une tâche peut être l'exploitation d'une compétence (savoir) ou peut être une action qui nécessite une ou plusieurs compétences pour sa réalisation (savoir faire). Un agent devra donc posséder sinon acquérir les compétences nécessaires à la réalisation d'une tâche pour pouvoir tenir un rôle. Toutes fois la gestion des tâches est locale à l'agent. Une tâche pouvant dépendre du résultat d'une autre tâche, un agent doit donc organiser lui-même la concrétisation de ses tâches. Cette compétence sera nécessaire au niveau de la tenue d'un rôle.

3.2.3 Compétence

Une compétence est une unité de connaissance nécessaire à la réalisation d'une tâche donnée. Le concept de compétence est vu ici au m^me niveau organisationnel que le rôle ou les tâches car on ne peut représenter le rôle sans la notion de compétence. C'est à la fois un savoir et un savoir faire. Une compétence peut être

réactive ou cognitive. Un agent réactif ne peut recevoir que des compétences réactives, alors un agent cognitif peut utiliser des compétences réactives et cognitives.

Un agent peut agréger plusieurs compétences pour réaliser l'ensemble des tâches prévu à la tenue d'un rôle. Il y a évidement plusieurs niveau d'abstraction car une compétence d'un niveau i peut être l'agrégation de plusieurs compétences de niveau i-1. Une compétence dite « atomique » pourrait, à ce titre, être la faculté l'addition de 1+1 ou la faculté de bouger un membre de son corps (dans le cas des agent robots). Une compétence de plus haut niveau pourrait être da savoir réaliser un addition quelconque ou de déplacer tout le corps en même temps en actionnant les membres de façons synchronisée.

L'intrication entre rôle et compétence est non trivial car l'enrichissement de la compétence à une répercutions sur le rôle. Un agent qui ne sait que se déplacer ne va pas tenir le même rôle qu'un agent capable de se déplacer de façons coordonnée et harmonieuse dans le cadre d'une chorégraphie.

4 Conclusion

De ce qui précède, nous avons fait le point sur les caractéristiques communes que nous avons relevé dans la plupart des modèles organisationnels. Nous y relevons comme points clés :

Une structuration de l'espace social, pour faciliter la coopération, l'interaction entre ses membres. L'organisation est avant tout affaire de support à l'activité collective, facilitant l'action collective des agents dans leurs espaces d'action.

Une dynamique associée a cette structure. L'organisation est capable d'évoluer, de faire migrer certains de ses membres d'un point à l'autre de la structure pour améliorer ses paramètres propres, voire d'envisager une réorganisation pour répondre à de nouvelles exigences.

Un point de rencontre obligé entre structure sociale abstraite et entités l'incarnant. On sait maintenant bien qu'une étude de la structure de l'organisation en tant que telle et faisant abstraction de ses agents ne peut rendre compte de la richesse des comportements possibles. L'entité individuelle est contrainte par l'organisation mais garde une liberté de décision dans ce cadre.

Une attribution fonctionnelle à l'ensemble de l'organisation. Ceci peut être aussi simple qu'un désir de survie de l'organisation, mais est plus classiquement un comportement que l'on attend de l'ensemble du système : prospérer, remplir un objectif, accomplir des tâches ...

Une capacité d'adaptation supposée. L'organisation est une entité organique, qui sait réagir à son environnement direct, absorber les chocs, survivre aux aléas. Cela concerne également la capacité à résister aux conflits internes : les situations bloquantes doivent être évitables et gérables rapidement pour ne pas pénaliser l'ensemble.

Une identification manipulable. L'organisation reste une entité désignable en tant que telle et l'on peut imaginer et observer plusieurs organisations distinctes, mais basées sur la même définition abstraite. Cela nous amène à définir un cadre d'action plus précis, qui nous guidera dans l'élaboration d'un modèle organisationnel générique. Que peux-t'on effectivement supposer sur une organisation, à part qu'elle servira de base à la coopération ?

Le respect de l'autonomie individuelle. Même si l'analyste - ou le concepteur, dans le contexte de construction de systèmes computationnels - veut pouvoir envisager l'organisation comme une entité unique et identifiable, il n'en demeure pas moins que l'activité est distribuée au niveau de chaque entité agissante du système. Il n'y a pas de locus unique identifiable dans l'action. Il faut garder à l'esprit la clé de la compréhension de la notion d'organisation qu'est ce couplage entre structuration abstraite et incarnation de l'activité. On pourra après mettre un accent plus ou moins prononcé sur l'entité individuelle ou le contexte social, mais cette dualité demeure.

La capacité de séparation. L'organisation suppose une activité structurée (sinon on ne serait pas en présence d'un système organisé !). Cela implique de pouvoir considérer, en tant que regard distinct sur le système, une sous-partie de cette organisation. Cette propriété est essentielle : c'est précisément ce qui nous permet de maîtriser un système arbitrairement complexe par une modularisation et une séparation des problématiques. Dans le cadre d'une organisation, nous voulons un minimum d'indépendance dans l'analyse ou la conception des situations d'interaction, des ontologies manipulés, ou même des objectifs des différents groupes d'acteurs.

*La possibilité de communication***.** Si l'organisation permet de structurer l'espace social, c'est précisément pour fournir le support d'une mise en relation entre entités. Le travail collectif implique nécessairement une inter-action entre ses membres, qui n'est pas possible sans une communication, quelle qu'elle soit. Notons que l'on ne se restreint pas forcément ici aux communications d'un haut niveau d'abstraction : nous pouvons tout à fait envisager des mécanismes non symboliques comme le marquage d'un environnement partagé. Le point essentiel est que les entités aient un processus d'interprétation commun de l'activité de communication.

Chapitre V

La Plate Forme NewObject

Résumé : *l'avènement de l'approche Mutlti-Agents à bien entendu laisser la porte grande ouverte sur les tentatives et essais de mettre au point un outil de conception et d'implémentation orientée agent. Malgré les efforts énormes consentis dans ce domaine par les différentes équipes de recherche à travers le monde, n' a pas encore atteint le but principale qui est la conception d'une plate forme Mutli-Agent générique ; autrement dit applicable à tous les domaines comme il en est avec l'approche objet. A la différence de l'objet, qui est un paradigme de programmation, l'agent tel qu'il est présenté dans la littérature, est un paradigme de construction d'application. L'objectif de notre travail, présenté dans ce chapitre, consiste en un essai dans ce sens, c'est à dire nous essayons de montrer qu'il est possible de mettre au point une plate forme Multi-Agent plus ou moins générique en prenant simplement comme point de départ l'objet et de voir comment on peut le transformer en agent.*

1 Introduction

Dans ce chapitre nous allons présenter notre propre travail qui consiste en l'étude et la conception d'une plate forme multi-agents baptisée NewObject, qui se veut une extension naturel du paradigme objet pour en faire un paradigme agent.

Mais avant d'entamer la construction proprement dite de la plate forme cible, on va regarder en premier lieu en quoi consiste le paradigme objet, en deuxième lieu l'accent est mis sur les principales dissemblances existantes entre l'objet et l'agent. C'est justement ces différences qui vont nous guidés le long de la construction de la plate forme NewObject.

2 Paradigme et modèle objet

La programmation par objet est née du constat des faiblesses de la programmation structurée et de la conception par hiérarchie fonctionnelle descendante. Le problème majeur de ces approches est la dichotomie très nette entre structure des données et structure des traitements qu'elles induisent : le processus de conception s'attachant uniquement aux fonctions et à leur raffinage progressif, les données sont en quelque sorte diffuses dans toute la conception.

Or, au cours d'un processus de conception, on se rend compte que la structure que l'on envisage pour les traitements est bien plus souvent remise en cause que la structure des données. Il est donc plus judicieux de centrer le processus de conception sur les données (ou les «objets») manipulés. D'autre part, en centrant le processus de conception sur les données, la probabilité d'obtenir une certaine forme de réutilisation augmente, car dans de nombreux cas, un logiciel est une collection de structures de données plus ou moins «universelles» agencées pour obtenir une fonctionnalité «spécifique».

L'approche objet vise donc à permettre un processus de conception guidé par les données, mais s'attache à cacher la représentation interne de ces données en n'autorisant leur accès qu'à travers un ensemble d'opérateurs, qui masquent aux utilisateurs de l'objet les détails de son implantation physique. Ainsi la dichotomie entre données et traitement disparaît, et l'objet est une entité qui rassemble ses données propres et les opérateurs susceptibles de les manipuler.

Rumbaugh [37] caractérise une approche orientée-objet par quatre critères :

- Le premier est **l'identité** : tout objet à une identité (sa référence) et les références d'objets sont uniformes et indépendantes du contenu, ce qui permet de constituer des collections d'objets hétérogènes ;
- Le second est la **classification** : tout objet est une instance d'une classe, qui définit sa structure et son comportement ;
- Le troisième est le **polymorphisme**, c'est à dire la propriété qu'un même traitement ait des interprétations différentes selon la classe de l'objet auquel on l'applique ;

- Le quatrième est **l'héritage**, qui permet de relier les différentes classes en une hiérarchie taxonomique, et qui est essentiel pour promouvoir l'extensibilité et la réutilisabilité des classes.

- L'architecture d'un système d'objets est structurée suivant une organisation de type client / serveur. Un objet du système, qui communique avec les autres objets par envoi de messages, réclame l'exécution d'un service à un autre objet, et en récupère le résultat éventuel. Nous utiliserons le terme d'invocation pour désigner l'action d'un objet client qui réclame l'exécution d'un service à un objet serveur.

- Avant de poursuivre, prenons une définition du concept d'objet. Cette définition ne prétend pas à l'universalité.

Bien que plus ou moins récente (fin des années 70), la méthode d'analyse, de conception et de développement orienté objet n'a jamais connu d'heures aussi glorieuses que depuis la fin des années 90. Cette méthode nous fournit avec ses concepts fondamentaux une nouvelle technique de conception et de développement des systèmes d'information. Parmi ces concepts fondamentaux, notons, entre autres, les objets (évidemment), les classes, l'encapsulation, l'héritage, le polymorphisme et l'agrégation. Sans être tous des concepts inédits, les concepts à la base de la méthode orientée objet sont utilisés de façon telle qu'ils créent un nouveau paradigme qui se détache de façon marquée des méthodes de conception structurée (orientés traitements - développement traditionnel) et des méthodes de conception orientées vers les données. Voyons donc de plus près quels sont ces différents concepts.

2.1 Objet

Définition

Un ***objet*** est une entité qui possède un état, et qui offre à son environnement un ensemble d'opérateurs destinés à faire évoluer cet état.

On peut qualifier un objet de la façon suivante : "un objet est quelque chose de réel ou d'abstrait, de tangible ou d'intangible, à propos duquel on emmagasine des données et les méthodes (procédures, fonctions) permettant de les manipuler" . Il ne s'agit là que d'une définition parmi tant d'autres mais elle contient 3 des éléments fondamentaux qui définissent un objet soit, le caractère réel ou abstrait d'un objet (ex. un avion ou la représentation d'un avion), le caractère intangible ou tangible (ex. avion ou vol) et ce qu'il contient (les données et les méthodes de manipulation). Un quatrième élément fondamental de l'objet est son unicité. Chaque objet est une occurrence et peut donc être identifié de façon unique, être distingué des autres objets (l'avion 1 n'est pas l'avion 2). La structure de données d'un objet est définie en terme d'attributs. Ces attributs sont des valeurs qui sont associées et étroitement connectées aux objets (ex. le modèle, l'année de fabrication, la taille, etc... d'un avion).

2.2 Classe

Définition

Une ***Classe d'Objets*** définit "en intention" l'espace d'états des objets qui sont ses instances en décrivant pour ces objets une structure de données, un ensemble d'opérateurs de changement d'état et une structure de contrôle.

Une classe d'objets permet donc de regrouper des objets possédant les mêmes caractéristiques (même structure et même ensemble de méthodes) afin d'en faciliter la gestion. Les classes, qui possèdent aussi leurs méthodes, sont utilisées pour créer de nouveaux objets (de nouvelles instances). Les classes agissent un peu comme des "moules à objets".

2.3 Les Types Abstraits de Données

Les Types Abstraits de Données (TAD) [38] sont une formalisation du concept d'objet dans une approche algébrique. Une description par Type Abstrait vise à obtenir une description complète, précise et sans ambiguïté d'une structure de données, sans pour autant se fonder sur la représentation physique de cette structure de données. Un TAD est une abstraction d'un concept pour lequel une implémentation particulière n'est qu'une instance.

Un Type Abstrait de Donnée décrit une classe de structures de données en fournissant un ensemble de Services et en précisant les propriétés formelles de ces services.

Les services sont décrits par leur signature, c'est à dire leur espace de départ et d'arrivée. Ils donnent en quelque sorte la syntaxe du type ;

- Les propriétés formelles sont de deux ordres : les Pré-condition et les axiomes. Pré-condition et axiomes donnent à eux deux la sémantique du type.
    - Les préconditions définissent quand un service est applicable, en fonction de l'état de l'instance ;
    - Les axiomes décrivent les propriétés sémantiques invariantes des instances du type.

L'exécutabilité des Types Abstraits est assurée par des techniques de réécriture. L'aspect formel de cette approche de spécification permet d'effectuer des preuves de cohérence et de complétude des modèles produits.

La notion de type abstrait s'apparente à celle d'objet si l'on accepte d'associer à chaque champ un domaine de définition. C'est le choix qui est fait dans les langages à objets fortement typés tels que C++.

2.4 Encapsulation

L'encapsulation signifie, en termes simples, que les données, de même que les méthodes permettant de les manipuler, sont contenues dans un même emballage (objet) et qu'un utilisateur (humain ou un autre objet) ne peut directement atteindre les données associées à cet objet. L'utilisateur peut accéder aux données d'un objet

uniquement en effectuant une requête sur les méthodes de manipulation des données de cet objet. Ces méthodes de manipulation sont appelées des accesseurs (accesseurs en consultation et en modification). L'utilisateur est donc uniquement concerné par la requête de services à effectuer et non par les détails et la façon dont ces services sont effectués.

2.5 Héritage

Dans une démarche de modélisation, la définition d'une hiérarchie d'héritage entre classes a plusieurs significations :

- Tout d'abord, l'héritage permet de favoriser la réutilisation : en pratique, un système ne se construit pas "ex nihilo", mais au contraire s'appuie sur l'expérience acquise, et profite de développements antérieurs. En termes de conception orientée-objet, cela ce traduit par le fait qu'on souhaite récupérer la conception d'une classe précédemment développée et simplement l'adapter à de nouveaux besoins, sans avoir à recommencer sa construction depuis le début. L'héritage est alors considéré comme un procédé essentiellement pratique permettant l'extension aisée des classes existantes. Dans cette approche, une classe est considérée comme un module, dont on souhaite récupérer et étendre les fonctionnalités ;

- L'héritage a également un sens plus fondamental, lié aux théories de représentation de la connaissance [39]. Ici on cherche à structurer l'ensemble des classes en une taxonomie, permettant de grouper des entités conceptuellement proches. Lorsque l'on considère l'extension d'une classe, l'héritage porte alors une sémantique proche de l'inclusion ensembliste : Si une classe B hérite d'une classe A, alors l'ensemble des instances de B est inclus dans l'ensemble des instances de A. L'héritage décrit alors la spécialisation d'une classe par une autre; On dira alors qu'un B "est un" A (is-a) ou "est une sorte de" A (a kind of ou AKO). La classe est alors considérée davantage comme un Type Abstrait de Données, et l'héritage formalise la notion de sous type.

Héritage simple

Les notions d'extension et de spécialisation sont souvent confondues dans les langages de programmation Orientés-Objet, et ne sont d'ailleurs pas totalement indépendantes. Toutefois, certains auteurs [40] proposent de trancher plus nettement entre l'héritage, considéré comme un concept lié à l'implémentation, et le sous-typage, qui construit une hiérarchie conceptuelle de classes proposant une interface de messages compatible.

America définit, pour une classe, une interface d'utilisation et une interface d'héritage, cette dernière constituant un ensemble cohérent d'attributs et de services. Dans le même esprit, on introduit la notion de conformité (conformance) entre classes, plus large que l'héritage, où deux classes peuvent être conformes l'une à l'autre sans être explicitement reliées dans le graphe d'héritage, c'est à dire sans forcément partager le même ensemble d'attributs.

La vision conceptuelle de l'héritage (qui identifie classe et type) et sa vision pratique (qui identifie classe et module) peuvent parfois entrer en conflit : par exemple, la vision pratique ferait de *Ellipse*

une sous classe de *Cercle*, puisqu'une ellipse a les même caractéristiques qu'un cercle, avec l'adjonction d'un centre supplémentaire ; ceci est bien entendu en contradiction avec la vision conceptuelle, où *Cercle* est une sous-classe de *Ellipse*, avec la contrainte que les deux centres soient confondus.

Héritage multiple

On parle d'héritage multiple lorsque une classe peut avoir plusieurs antécédents immédiats ; le graphe d'héritage n'est alors plus un arbre, mais un treillis. Dans ce cadre, les deux aspects de l'héritage (vision d'une classe comme un module ou comme un type) sont présents :

- La puissance de représentation des connaissances est augmentée, car on peut dès lors regrouper au sein d'une même classe des caractéristiques de provenance diverse. Le danger potentiel de cette approche est de surcharger une classe de manière à ce qu'elle représente en définitive plus d'un type d'objet;

- La réutilisation est facilitée : ainsi en Eiffel, l'héritage multiple est souvent utilisé pour émuler le mécanisme d'inclusion (#include) des langages modulaires tels que C ; on peut ainsi définir une classe comme une bibliothèque de fonctions utilitaires (par exemple des routines mathématiques). Si une classe a besoin d'une de ces fonction, il lui suffit d'ajouter la classe-bibliothèque à la liste de ses ancêtres.

Le seul problème posé par l'héritage multiple est la possibilité de conflit de nom entre des caractéristiques (services ou attributs) héritées de différents ancêtres. Les langages ou les formalismes supportant l'héritage multiple proposent divers mécanismes pour lever ces conflits : les techniques les plus fréquemment présentées sont la spécification d'un mode de parcours du graphe d'héritage, qui conserve la première définition rencontrée de la caractéristique, ou le renommage explicite au niveau de la classe des caractéristiques en conflit [41].

En résumé l'héritage est un concept qui découle directement des processus cognitifs naturels que sont la classification, la généralisation et la spécialisation. Plus spécifiquement, "l'héritage est une relation entre les classes qui permet la définition et la mise en oeuvre d'une classe basée sur d'autres classes existantes". Ainsi, la relation d'héritage permet à une classe d'objets (sous-classe) de réutiliser les attributs et les opérations (méthodes) définis pour une classe d'objets plus générale (super-classe).

L'héritage peut être simple ou multiple. L'héritage simple permet à une sous-classe d'avoir uniquement une seule super-classe, alors que l'héritage multiple permet à une sous-classe d'avoir plusieurs super-classes.

2.6 Agrégation

L'agrégation est une relation qui permet de décrire un objet composite en terme d'objets qui le constituent (ex. vehicule = 1 chassis + 2 trains avant et arrière + … etc).

2.7 Polymorphisme

Le polymorphisme permet à une sous-classe d'outrepasser l'implantation d'une méthode héritée d'une super-classe. Il consiste, tout en gardant le même nom pour une méthode héritée, à associer un code spécifique qui vient ainsi se substituer à celui de la méthode héritée. Lorsque la méthode est appelée, l'utilisateur (humain ou autre objet) n'a pas besoin de connaître quelle est l'implantation précise à sélectionner puisque le système OO se charge de sélectionner le code qui est approprié pour la classe en question.

2.8 La réutilisation

A travers son mécanisme d'héritage, la méthode orientée objet permet la réutilisation des composantes (classes, objets) à l'intérieur d'un même système de même qu'à l'intérieur de plusieurs systèmes. Le concept de réutilisation n'est pas unique à la méthode objet mais, contrairement aux autres méthodes (conception structurée et conception orientée vers les données), son application y est quasi implicite. La réutilisation est considérée par plusieurs experts comme une réduction du temps et du coût du développement puisqu'elle permet d'accélérer la vitesse de développement et ainsi de réduire les coûts qui y sont liés.

3 Les principales différences entre Objet et Agent

Dans ce paragraphe nous allons mettre l'accent sur les principales différences existantes entre un objet et un agent tel qu'ils sont définis dans la littérature.

3.1 Sur le plan autonomie

La première différence réside dans la façon dont un agent et un objet sont autonomes. L'objet encapsule son état qui sera accessible par d'autres objets via les méthodes publiques de son interface. Il contrôle ainsi son état mais pas son comportement. Dès lors qu'une méthode est accessible (visible ou publique) elle peut être déclenchée à tout moment sans d'ailleurs aucune connaissance pour l'objet cible de l'objet "déclencheur".

Dans un SMA un agent demande à un autre agent de réaliser une action ou d'atteindre un but, voir un objectif. L'agent, contrairement à l'objet, a le pouvoir de refuser ou de négocier cette demande. Enfin rien n'interdit de construire des SMA en utilisant des langages à objets. C'est le cas aujourd'hui pour la plupart des SMA opérationnels. En revanche cela demande justement de construire les mécanismes offrant cette autonomie indispensable.

3.2 Sur le plan conceptuel

La conception "objet" d'un système impose le partage de buts communs, ce qui ne peut être envisagé dans le cas des SMA, pour lesquels des systèmes conçus indépendamment les uns des autres sont amenés à inter opérer. En effet le paradigme objet repose sur la notion de client et de serveur. On décompose une application en fonctions qui seront implantées par des objets (composites ou non suivant la granularité choisie). L'apport est strictement méthodologique car elle favorise la réutilisation de code, la modularité ...etc.

3.3 Sur le plan comportement

Du faite que la conception objet de systèmes impose un partage de buts communs et que l'objet est démuni de comportement social (l'objet est vu comme étant une boite noir ayant une entrée et une sortie) rend ces systèmes non flexible aux éventuels changements possible comme l'ajout de nouveaux objets par exemple, dans les SMA c'est tout à fait le contraire, il se trouve que les agents ont un comportement (réactivité, pro-activite et comportement social). Le paradigme objet ne propose rien concernant cet aspect.

3.4 Sur le plan environnement

Dans les systèmes à base d'objets généralement l'environnement est statique et il n'y a pas de changements qui peuvent mettre en cause l'activité des objets, dans les systèmes à base d 'agents la situation est tout à fait différentes. Dans les SMA l'environnement est dynamique et les modifications peuvent survenir à n'importe quel moment, voir même au cours de l'exécution de l'agent de ses tâches. le dynamisme de l'environnement dans les SMA résulte de l'autonomie des agents

3.5 Sur le plan adaptabilité

Un objet est définit comme étant une entité statique, une boite noir qui accepte des informations en entrées et les transformes en information en sortie, et ne change guerre durant tout le cycle de son mise en exploitation. Par contre un agent s'adapte à son environnement et peut changer de mission au cours de son existence.

L'adaptabilité de l'agent à l'environnement est due essentiellement au concept d'apprentissage qui permet à l'agent d'acquérir de nouvelles compétences et d'avoir un autre rôle dans la société d'agents à laquelle il appartient

4 Les Types d'Architectures concrètes pour agents

Le but de cette partie est de faire le tour de quatre classes d'agents différentes et de voir comment on représente l'état interne d'un agent d'une manière formelle et comment on implante une action; Ces classes d'agents sont :

- ***Les agents logiques*** : le processus décisionnel est basé sur des déductions logiques
- ***Les agents réactifs*** : ici le processus décisionnel est basé sur une simple correspondance entre les situations *(*s) et les actions *(*a).
- ***Les agents "BDI"*** : ici l'agent décide des actions à entreprendre à partir de ses états internes *(*I) qui sont exprimés sous la forme de croyances (Belief), de désirs (Desire) et d'intentions (Intention).
- ***Les agents "multi-niveaux" :*** les connaissances internes de ce type d'agents sont organisées en différents niveaux d'abstractions, permettant ainsi différents niveaux de traitements.

4.1 Architectures d'agents logiques

Dans cette architecture les connaissances des agents reposent sur des symboles et sont décrites sous la forme d'expressions logiques. L'agent utilise la déduction logique dans la résolution des problèmes pour piloter son comportement.

Une théorie logique capable de décrire le comportement d'un agent et en particulier son processus de décision (observation et action) face à son environnement, doit expliciter comment l'agent génère ses buts pour atteindre ses objectifs, et comment il combine alternativement des comportements réactifs et dirigés par les buts (délibératifs).

Traditionnellement on part des spécifications de haut niveau du comportement d'une application que l'on raffine par strates successives afin d'arriver à des entités que l'on peut implanter en machine. Dans le cas des approches logiques, les spécifications de haut niveau sont directement implantées en machine. On parle alors de spécifications exécutables, qui produisent directement le comportement des composants de l'application. Cela a le mérite d'être très élégant dans la mesure où toute la sémantique de l'application est exprimée sous une même représentation assez simple.

Pour comprendre les principaux problèmes de cette approche, regardons en détails sur plusieurs exemples comment elle fonctionne.

Soit un exemple d'agent dont la base de faits est constituée de formules logiques comme celles ci :

Ouvert(valve221)

Température(réacteur4726, 321)

Pression(réservoir776, 28)

Ce type de formules représente assez bien l'environnement de l'agent. La base de faits de l'agent correspond à ce qu'il croit sur l'état de son environnement. Si l'agent pense que la *valve 221* est ouverte, alors il possède le fait *ouvert(valve221)* dans sa base. Inversement la présence de ce fait dans sa base n'implique pas que la réalité de son environnement soit en accord avec ce fait. Pour cela, il suffit que le capteur fonctionne mal, ou que le raisonnement qui a conduit à la production de ce fait soit faux ou encore que l'interprétation de la formule *ouvert(valve221)* soit complètement différente selon le concepteur de l'agent et l'agent lui même (en terme d'action induite par sa présence).

Soit L l'ensemble des formules logiques du premier ordre et $D = \wp(L)$ l'ensemble des bases de faits possibles contenant des éléments de L. Chaque état de l'agent correspond à un élément de D. Soient les différents membres (états de l'agent) de D. Le processus de décision d'un agent est modélisé par un ensemble ρ de règles de déduction (inférence).

On écrit $\Delta |---_{\rho} \Phi$ si la formule logique Φ peut être prouvée en appliquant ρ (règles déductives) sur Δ. La fonction de perception $capter : S \rightarrow P$ ne change pas. En revanche la fonction $compiler : D \times P \rightarrow D$

associe l'ancienne base de faits à la perception en cours de l'environnement pour produire la nouvelle base de faits (état interne) de l'agent.

Ce nouvel état conduit l'agent à produire (règles de déduction) une action par l'intermédiaire de la fonction *agir : D* $\rightarrow A$. Cette fonction prend donc en entrée un état de l'agent parmi l'ensemble (D) de ses états possibles et renvoie en sortie une action dans l'ensemble (A) des actions possibles. Son pseudo-code est le suivant :

1. fonction *agir (*Δ *: D) : A*
2. début
3. pour chaque $a \in A$ faire
4. si Δ |--- $_{\rho}$ *Faire(a)* alors
5. renvoyer *a*
6. fin si
7. fin. pour
8. pour chaque $a \in A$ faire
9. si Δ $\chi \rightarrow _{\rho}$ $\neg$ *Faire(a)* alors
10. renvoyer *a*
11. fin.si
12. fin. pour
13. renvoyer *null*
14. fin fonction *agir*

Le principe de cette fonction est que l'agent recherche parmi les actions qu'il connaît, celle *(*a) pour laquelle il peut déduire la formule *Faire (a)* de son état courant (base D) et en utilisant ses règles d'inférence ρ . La première action qui vérifie ces conditions sera exécutée par l'agent (cf. lignes 3 à 7), car étant considérée comme la meilleure. Si l'agent ne trouve pas cette meilleure action, il tentera de trouver une action *(a'*) consistante avec son état interne, c'est à .dire dont la formule *non Faire(a')* ne peut être déduite de sa base Δ (cf. ligne 9). En cas de nouvel échec, l'agent ne déclenchera aucune action.

C'est ainsi que le comportement de l'agent est déterminé par ρ (son programme) et son état interne courant (base de faits sur ce qu'il connaît de son environnement).

4.2 Architectures réactives

Les approches "réactives" sont issues des problèmes et limites rencontrés par l'approche logique (voir la section 4.5 Discussion). Dans le milieu des années 80, des chercheurs décidèrent de trouver une alternative à cette approche afin de mieux tenir compte des très fortes contraintes temporelles liées au caractère changeant des environnements réels.

Cette approche est basée principalement sur un comportement *réactif* de l'agent à son environnement et sur l'émergence de comportements intelligents à partir de l'interaction de nombreux comportements

élémentaires (simples). Les agents réactifs sont des agents qui interagissent avec leur environnement plutôt que de raisonner dessus.

De plus en plus de travaux s'intéresse à des approches mixtes dans lesquelles les agents ont une composante réactive et une ou plusieurs composantes cognitives (incluant du raisonnement) ; nous verrons ces approches avec les architectures multi-niveaux.

Pour bien mesurer l'apport des approches purement réactives, nous allons examiner l'architecture de "subsomption" (de subsumer, mettre sous) de Rodney Brooks dans [42], qui est considérée comme la plus représentative et dont l'auteur est l'un des plus fervents opposants à l'approche symbolique.

Dans cette architecture chaque comportement de l'agent est vu à travers une fonction "agir" qui lui est propre. Cette fonction reçoit en permanence des informations sur l'environnement de l'agent et décide de l'action à accomplir. L'agent possède donc autant de modules comportementaux (fonctions) que de tâches spécifiques à accomplir, qui sont réalisés sous la forme d'automates à états finis et qui fonctionnent en parallèle. Ces modules manipulent une représentation symbolique simplifiée et ne raisonnent pas directement avec cette représentation.

En général les règles de décision sont de la forme $situation \rightarrow action$; elles font correspondre une action à un ensemble de perceptions.

Une autre caractéristique de ce type d'architecture est que plusieurs comportements peuvent être potentiellement lancés simultanément. L'idée est de pouvoir capter simultanément différents niveaux d'information de l'environnement. Cela nécessite de mettre en place dans l'agent un mécanisme de choix du comportement à privilégier, en cas de sélection de plusieurs actions contradictoires.

D'où le principe de niveaux comportementaux allant du plus bas (le plus prioritaire et le moins abstrait) au plus haut (le moins prioritaire et le plus abstrait). Un niveau inférieur (subsomption) peut "cacher" les niveaux supérieurs, dans la mesure où il conduira au déclenchement d'une action inhibitrice pour d'autres actions des niveaux supérieurs, évitant ainsi tout risque de conflit.

Dans le cas des robots mobiles, à l'origine de ces travaux de Brooks, un comportement très prioritaire est par exemple d'éviter un obstacle, avant même de savoir quelle en est sa nature.

Formellement un comportement est une paire *(c, a)*, où $c \subseteq P$ est un ensemble de perceptions appelé *condition*, et $a \in A$ est une action. Un comportement *(c, a)* sera potentiellement jouable par l'agent si et seulement si l'environnement est dans un état $s \in S$ avec *capter (s)* $\in c$. L'ensemble des comportements possibles de l'agent sera défini par la règle suivante :

$$Comp = \{ (c, a) \mid c \subseteq P \text{ et } a \in A \}$$

On associe à l'ensemble des règles comportementales $R \subseteq Comp$ de l'agent, une relation binaire d'inhibition $\Im$ sur R : $\Im \subseteq R \times R$.

C'est une relation d'ordre total sur R (transitive, non .réflexive et antisymétrique).
On écrit $b_1 \Im b_2$ *si* $(b_1 , b_2) \in \Im$, et on prononce b_1 *inhibe* b_2, ce qui signifie que b_1 est de plus bas niveau que b_2 et sera prioritaire sur b_2.

La fonction *agir* prend en entrée une perception *(p)* de l'ensemble *(P)* des perceptions possibles de l'agent et renvoie en sortie une ou plusieurs actions parmi l'ensemble *(A)* de ses actions possibles. Elle est définie de la façon suivante :

1. fonction *agir(p : P) : A*
2. var *activable* : $\wp (R)$
3. début
4. *activable :={ (c, a)|(c, a) ∈ R et p ∈ c }*
5. pour chaque (*c, a*) ∈ *activable* faire
6. si ¬ (∃ (c',a') ∈ *activable tel que (c',a') ℑ (c,a))* alors
7. renvoyer *a*
8. fin si
9. fin pour
10. renvoyer *null*
11. fin fonction *agir*

L'agent commence par sélectionner (étape 4), grâce à ses capteurs, l'ensemble de ses comportements possibles (variable *activabl*e). Pour chacun d'eux l'agent vérifie (étape 6) si parmi les comportements trouvés l'un d'entre eux inhibe ce comportement courant. Dans ce cas l'agent ne retient pas ce comportement ; sinon (étape 7) il renvoie l'action correspondant à ce comportement, pour être placée dans la liste des actions retenues.

Cette architecture conduit au pire à un temps de calcul pour le processus de décision qui est en $O(n^2)$ où n est le plus grand nombre entre le nombre de comportements et celui des perceptions. Ce qui est tout a fait accessible par des machines récentes, d'autant plus que ce mécanisme peut être codé dans le hardware (entraînant un temps de décision constant).

4.3 Architectures BDI (Belief, desire, Intention)

Cette approche s'intéresse aux différentes étapes du processus de raisonnement qui conduit à choisir une action pour atteindre un but fixé. Il faut d'abord fixer les buts à atteindre *(délibérati*on) c'est à dire se demander

quoi faire, puis regarder comment faire pour les atteindre *(means end reasoning*). Cette approche doit permettre aux agents qui l'utilisent d'avoir des comportements stables et cohérents même si l'environnement dans lequel ils évoluent est instable.

Le concept d'intention est au centre de cette approche car il permet de relier les buts aux croyances et aux engagements en disposant d'une théorie du passage à l'acte des agents.

Si l'on prend l'exemple de l'étudiant qui vient de terminer un premier cycle universitaire, il est devant un choix à faire pour la suite de sa vie active. Avant de choisir telle ou telle option, il faut d'abord voir qu'elles sont celles qui lui sont possibles. Suivant le niveau de l'étudiant il pourra ou non continuer un deuxième cycle universitaire ou entrer dans une école d'ingénieur. Une autre possibilité sera de trouver un travail dans l'industrie.

L'étudiant devra choisir parmi l'une de ces options potentielles, celle qui deviendra son intention (engagement pris par l'étudiant pour son comportement à venir). Ce choix guidera ses actions futures. On peut dire que ses intentions influenceront son raisonnement futur.

Si vous savez que l'étudiant à l'intention de devenir universitaire, alors vous vous attendrez à ce qu'il agisse en fonction de cette intention, par exemple en rattrapant des cours qu'il n'a pas beaucoup vu ou encore en essayant différents concours après avoir subit des échecs.

L'étudiant va éliminer des options qui seraient contradictoires avec ses intentions, comme par exemple prendre un travail très prenant alors qu'il ait décidé de continuer ses études.

Adopter une intention c'est avoir de la suite dans les idées, et se donner les moyens d'y arriver (rationalité), car le chemin peut être long et difficile. En revanche si pendant ce parcours on s'aperçoit que cette intention était soit sur dimensionnée par rapport à un potentiel requis ou ne s'avérait pas conduire à ce que l'on imaginait au départ, notre rationalité nous conduit naturellement à renoncer à cette intention.

Enfin les intentions sont très étroitement liées à notre analyse raisonnée de la situation courante et à notre croyance en l'avenir. Ce qui se traduit par une estimation des chances que l'on ait d'aboutir à la situation correspondant à nos intentions.

On peut résumer les rôles importants que jouent les intentions dans le raisonnement pragmatique :

- elles pilotent les moyens que l'on se donne pour atteindre les objectifs correspondants ;
- elles contraignent les réflexions et les actions futures ;
- elles persistent dans la durée ;
- elles influencent les croyances à la base des raisonnements à venir.

David Kinny et Michael Georgeff ont étudié, dans ce contexte d'agents BDI [42], les performances des agents "imprudents" (qui ne reconsidèrent jamais leurs intentions) et celles des agents "précautionneux" (qui reconsidèrent sans cesse). Le paramètre déterminant est en fait le taux de changement γ du monde dans lequel évoluent ces agents. Un γ faible favorise les agents *imprudent*s, alors qu'un γ élevé favorise les agents *précautionneux* qui tireront parti opportunément de nouvelles situations. Suivant la dynamicité de l'environnement l'une ou l'autre approche sera pertinente. D'où la décomposition de chaque agent BDI en au moins sept niveaux fonctionnels. Formellement *Bel* est l'ensemble des croyances possibles, *Des* est l'ensemble des désirs possibles et *Inten* est l'ensemble des intentions possibles de l'agent. Son état est alors décrit à tout moment par le triplet *(B, D, I*), où $B \subseteq Bel, D \subseteq Des, et\ I \subseteq Inten$. Ces niveaux sont les suivants :

- un ensemble de croyances courantes *(*B) sur son environnement ;
- une fonction de révision de ses croyances *(br*f) qui calcule ses nouvelles croyances à partir des courantes et de ses nouvelles perceptions de son environnement ; $brf : \wp(Bel) \times P \rightarrow \wp(Bel)$.
- une fonction de génération de ses options pertinentes *(option*s) qui représentent en fait ses désirs ou choix possibles conformément à ses intentions ; elle se base sur les croyances qu'il a de son environnement et sur ses intentions ; cette fonction est responsable des actions mises en oeuvre ; elle doit produire des options consistantes (vis-à-vis des croyances et intentions courantes) ; enfin elle doit être opportuniste, c'est à dire être capable de détecter des changements de l'environnement qui sont favorables à la conduite de nouvelles intentions ou d'intentions laissées car irréalistes : $options : \wp(Bel) \times \wp(Inten) \rightarrow \wp(Des)$.
- une fonction de filtre *(filtr*e) qui représente la phase initiale (quoi faire) de son processus de raisonnement ; elle active ses nouvelles intentions en fonction de ses croyances, options et intentions courantes ; c'est donc un processus continu ; cette fonction doit éliminer (rendre inactives) les intentions devenues irréalistes (trop coûteuses par rapport aux gains escomptés) ou incohérentes (entre elles) : $filtre : \wp(Bel) \times \wp(Des) \times \wp(Inten) \rightarrow \wp(Inten)$. Cette fonction ne crée pas de nouvelles intentions mais active/désactive des intentions existantes.
- un ensemble d'intentions courantes *(I*), représentant ses centres d'intérêts actuels ;
- une fonction de sélection *(exécute*r) de l'action à exécuter. Cette fonction renvoie une intention exécutable, qui correspond donc à une action : $exécuter : \wp(Inten) \rightarrow A$.
- la fonction de décision de l'agent est définie par $agir : P \rightarrow A$; son pseudo code est le suivant :

```
1.    fonction agir( p : P ) : A
2.    début
3.        B := brf ( B, p )
```

4. *D := options (B, I)*
5. *I := filtre (B, D, I)*
6. renvoyer *exécuter(I)*
7. fin fonction *agir.*

Il existe deux types d'intentions ; celles qui sont immédiates et qui correspondent à des actions à faire, et celles qui engagent un comportement dans le futur. Cette deuxième forme d'intentionnalité renvoie à un mécanisme de planification.

Le principe de raisonnement des agents BDI est donc de raffiner progressivement les options en intentions de plus en plus concrètes (planification), qui au final correspondront à des actions exécutables. L'agent planifie ses actions pour satisfaire ses intentions.

La modélisation de ces fonctions par des formules logiques permettra entre autres choses de vérifier la consistance de l'état d'un agent BDI. Cela revient à répondre à des questions comme la suivante : n'est il pas contradictoire d'avoir l'intention d'aboutir à *x* tout en ayant la croyance en *y* ?

En pratique les intentions ne sont pas toutes de même niveau, c'est pourquoi il peut être intéressant d'y associer des priorités, leur donnant ainsi des importances relatives. De même on peut mettre les intentions dans une pile des plus abstraites aux plus concrètes.

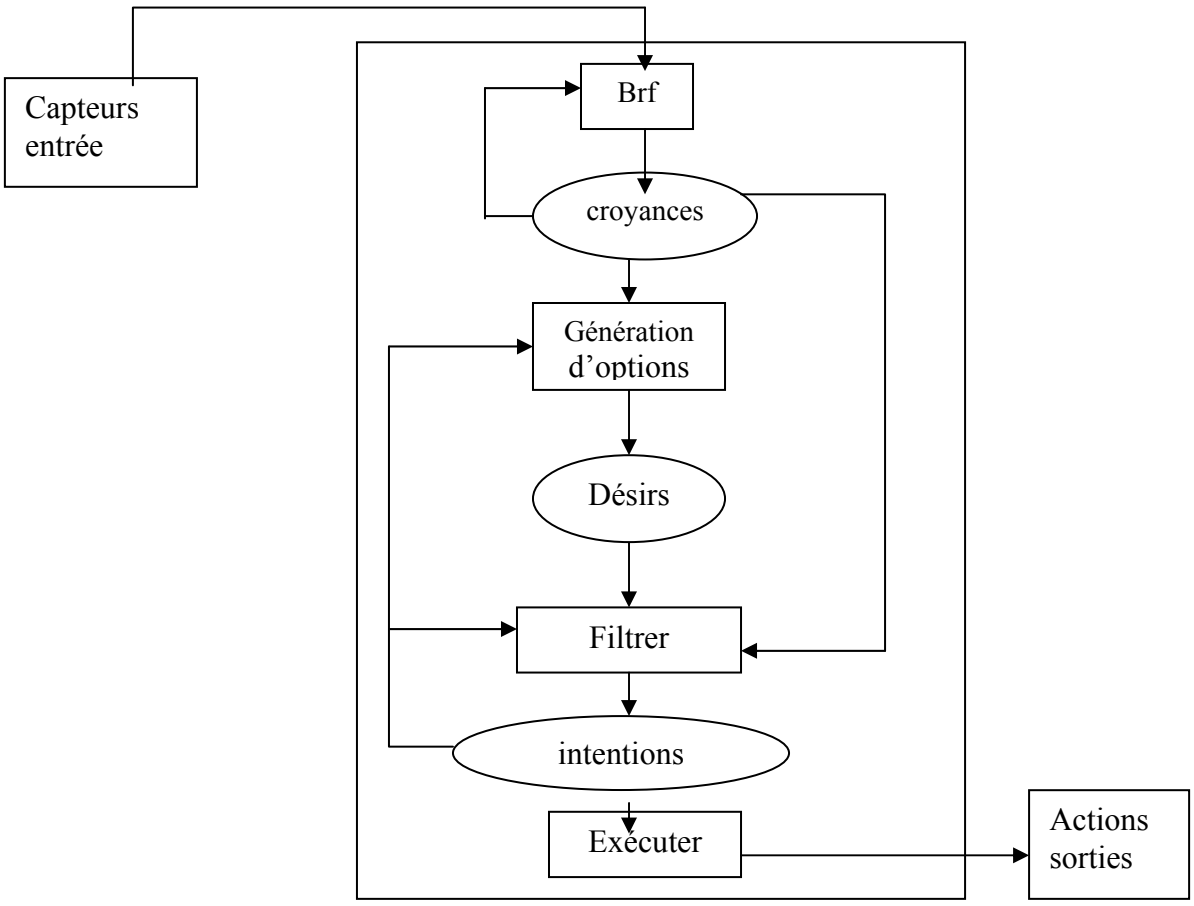


Fig 5.1 *Architecture d'un agent BDI*

Dans cette approche le concept d'intention est central car il permet de stabiliser le processus de décision de l'agent. En revanche la difficulté de cette approche est de régler le dispositif de révision des intentions en fonction des caractéristiques de l'environnement. Ce problème s'amplifie dans le cas des systèmes multi.-agents, dans la mesure où à tout instant les agents eux mêmes peuvent modifier l'environnement sans synchronisation entre eux. Dans ce contexte et pour rendre l'environnement le plus stable (c'est à dire prévisible) possible, il convient que chaque agent se tienne le plus souvent possible à ses engagements et communique ces derniers à ses collègues (socialisation). Une solution à ce problème revient à planifier des actions conjointes entre agents, pour lesquelles la notion d'engagement devient fondamentale. Il ne suffit plus de dire que l'on va faire quelque chose, encore faut il pouvoir en apporter des garanties.

En outre cette approche nécessite une bonne définition de ce qu'est une action, un événement, un état du monde et comment ces événements et états du monde sont liés dans le temps ; d'où le recours à des opérateurs temporels associés à une description multiformes de la notion d'action.

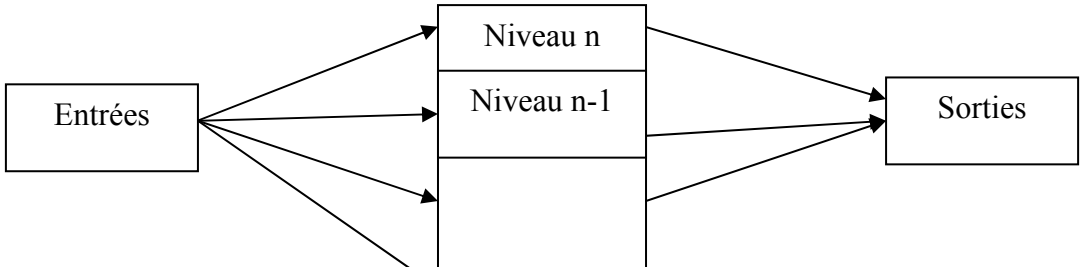
4.4 Architectures Hybrides ou Multi-niveaux

Comme nous l'avons vu dans les architectures précédentes, les approches *réactive* et *pro-active* se complètent plus que ne s'opposent. En effet chacune d'elles répond à des besoins précis, mais manque de ce que l'autre peut apporter. C'est justement l'objet des architectures multi niveaux que de faire une synthèse constructive de ces deux approches.

Pour cela elles se basent sur une hiérarchie de niveaux qui interagissent entre eux. Dans ces architectures il existe au moins deux modes de contrôle des échanges d'information entre les niveaux :

- ***contrôle horizontal*** : tous les modules (un par niveaux) sont directement connectés aux capteurs externes et à la sortie qui déclenche des actions ; chaque module interne à l'agent se comporte comme un agent en proposant des actions à faire ;
- ***contrôle vertical*** : seul un module gère les entrées (capteurs) et un autre les sorties (actions à faire).

Les architectures *horizontales* (à contrôle horizontal) sont plus simples à concevoir, mais peuvent conduire à des comportements incohérents (compétition entre les niveaux). Pour pallier à cela on peut utiliser une fonction de médiation (contrôle centralisé), qui impose à son concepteur de connaître toutes les interactions (conflits) entre les niveaux. Avec n niveaux et m actions possibles par niveau, on a m_n interactions possibles. De plus cela peut entraîner une chute de performances dans le processus de décision de l'agent (arbitrage de conflits internes).



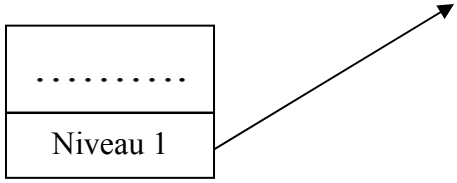


Fig 5.2 *Schéma du contrôle horizontal*

Les architectures *verticales* (à contrôle vertical) évitent partiellement ce problème en utilisant un contrôle des flux d'informations entre les niveaux, basé soit sur le mode à *une passe* soit sur le mode à *deux passe*s. Dans le mode à *une pass*e, les informations arrivent des capteurs sur un module spécialisé, puis traversent en séquence les autres modules jusqu'au dernier qui pilote la sortie (action à prendre).

Pour l'autre mode (deux passes) c'est le même module qui assure l'interface d'entrée et de sortie de l'agent. Les informations suivent le même chemin que dans le mode précédent puis redescend l'architecture en sens inverse pour revenir à l'unique module frontal (interface) de l'agent.

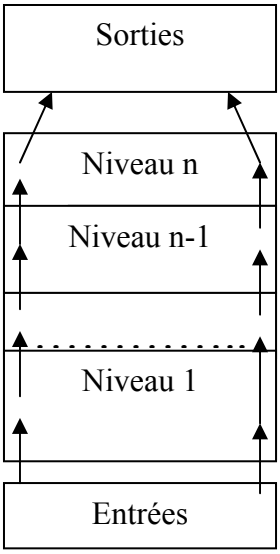


Fig 5.3 *Schéma du contrôle vertical*

4.5 Discussion :

Avant de dire quelle architecture nous préconisons pour les agents de notre plate forme , discutons d'abord les quatre type d'architectures présenter plus haut.

L'approche logique pour la réalisation d'agents est en pratique irréalisable dans des environnements réels, c'est a dire complexes, et fortement dynamiques (changeants). Cela vient justement du choix de l'action la plus pertinente à faire. En effet il suffit que l'agent utilise plus de temps pour inférer son choix (action choisie) que ne prendra l'environnement qu'il observe pour évoluer de telle façon que ce choix ne soit plus pertinent au moment où il prendra effet. Un exemple simple consiste à décider de déplacer un objet dans l'environnement alors que l'objet n'existe plus dans cet environnement. La solution impose que l'agent infère

son choix dans un temps instantané, ou en tout cas inférieur à la périodicité d'évolution de l'objet observé dans l'environnement. Cette contrainte est quasiment improbable dans un environnement réel, d'autant plus que dans l'approche logique le processus de choix peut être long voire très long.

Pour améliorer ce problème ont peut éviter de coder l'ensemble du comportement de l'agent sous forme de règles et faits logiques. On perd alors le bénéfice et l'élégance d'une description unique et simple de la sémantique comportementale de l'agent dans son environnement.

Il existe d'autres problèmes inhérents à cette approche comme par exemple :

- *Les environnements dynamiques et complexes* : Dans de tels environnements, il est très difficile de représenter symboliquement les propriétés de l'environnement liées à sa dynamique ou à sa complexité. C'est le cas par exemple pour représenter comment l'environnement change en fonction du temps.

Ceci a contraint les chercheurs dans ce domaine de trouver d'autres approches qui permette de pallier à ces difficultés rencontrées et c'est justement le rôle des architecture réactives.

Malgré les résultats encourageants, les architectures réactives imposent quelques contraintes spécifiques comme :

- Les agents ne disposent pas de modélisation de leur environnement ; cependant ils doivent avoir localement suffisamment d'informations sur ce dernier pour faire le choix de la bonne action à prendre.
- Cette localité de l'information interdit le recours à des états antérieurs de l'agent.
- Il devient alors difficile pour ce type d'agents de s'améliorer en apprenant à partir d'expériences.

Des solutions à ces problèmes ont été avancées comme celle des agents évolutifs et que l'on trouve dans le champ d'étude de la vie artificielle.

Ces contraintes ont un impacte seulement sur les domaines d'application, autrement dit ce type d'architecture n'est pas conseiller dans des application où l'apprentissage est à prendre en charge.

L'un des principaux problèmes rencontrés avec l'approche BDI lorsque l'on conçoit ce type d'agents est qu'ils aient un comportement qui traduise un juste milieu entre ces différentes propriétés. En effet un agent peut être amené, au cours de son fonctionnement, à laisser tomber certaines intentions qu'il juge ne plus être pertinentes, ou devenues irréalistes, ... Cela l'oblige de temps à autre à remettre en question ses propres intentions, afin de modifier éventuellement ses actions à venir. Cette reconsidération des intentions a un coût en temps et en ressources nécessaires.

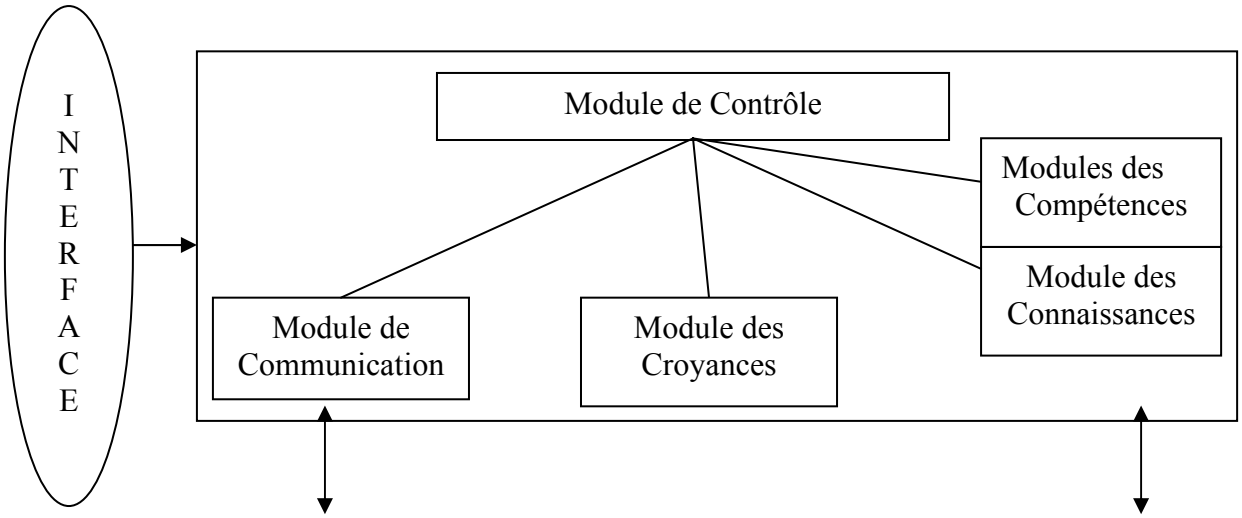
D'où le dilemme entre l'adoption alternative de comportements pro-actifs (dirigés par les buts) et de comportements réactifs (dirigés par les événements) :

- Soit l'agent ne reconsidère pas assez souvent ses intentions au risque de poursuivre des actions inutiles ;
- Soit il reconsidère trop souvent ses intentions au risque de ne jamais atteindre les objectifs qui en découlent.

L'architecture que nous préconisons pour notre plate forme est l'architecture réactive, car c'est la plus proche du paradigme Objets basé sur le concept client-serveur. Nous savons déjas que les objets sont des entités simple, réactifs aux stimulés et de plus dans la littérature les objets sont définit comme étant des agents réactifs (dans un certain sens bien sûr).

5 Modèle d'Agent proposé

Nous présentons dans ce paragraphe le modèle d'agent retenu pour notre plate forme ainsi que la description de l'ensemble de ses constituants.



E N V I R O N N E M E N T

Fig5.4 Structure interne d'un agent dans la plate forme NewObject

- La partie ***Interface*** contient l'ensemble des méthodes ou les compétences activables par les autres agents.

- Le module ***Contrôle*** permet à l'agent, en cas de sollicitation d'une compétence par un autre agent, de répondre favorablement en exécutant la compétence, ou répondre défavorablement en refusant la réalisation de l'action demander pour de multiple raison ; comme par exemple le cas où l'agent est occupé par l'exécution d'autre compétence. ce dernier se base sur les Croyances qu'a l'agent sur lui ainsi que sur autrui. L'introduction de ce module est in dispensable pour permettre à l'agent d'être autonome car ceci nous permet de pallier la première différence existant entre un objet et un agents, à savoir le concept d'autonomie, dans ce cas l'agent (l'objet) n'est plus astreint à répondre ou à accepter tous les message venant de ses semblables. Il a maintenant le contrôle de son état et de son comportement en acceptant ou en refusant la satisfaction des requêtes venants des autres semblables.

- Le Module de ***Communication*** est intégrer pour permettre la l'échange d'information avec les autres agent et l'environnement.

- Le module des ***Croyances*** renferme les connaissances de l'agents qu'il a sur lui, c'est à dire sur son état, et sur les autres agents.

- Module des ***Compétences*** renferme l'ensemble des capacités qu'un agent possède. Ces compétence ou son savoir faire sont généralement public et peuvent êtres invoquées instantanément selon l'état de l'agent.

- Le module ***Connaissances*** (Attributs)peut aller de simple variable jusqu'aux base de données ou base de connaissances que l'agent manipule dans la réalisation de ses actions pour atteindre ces objectifs. La modification apportée ici est qu'une connaissance peut faire référence à des données externe au module Connaissances c'et à dire extérieur à l'agent (objet).

5.1 Le Module de Contrôle

Ce module représente le méta-comportement de l'agent. Il lui offre la possibilité d'observer et de contrôler son comportement. Notre module de Contrôle décrit les séquences d'opérations qui peuvent s'enchaîner et les changements qu'elles induisent. Il décrit l'évolution du comportement de l'agent et a un rôle important à jouer dans la définition des activités concurrentes des différents modules de l'agent. Pour avoir une abstraction du comportement de l'agent, nous avons choisi de fonder la représentation de ce module sur les notions *d'état* et de *transition*. Les états marquent les étapes importantes de la vie d'un agent ; ils représentent une phase temporaire

de la vie d'un agent, dont il sort au bout d'un certain temps. Un agent ne change d'état qu'après l'apparition d'un événement. Les états définissent ainsi le contexte dans lequel prennent place les événements qui marquent la vie de l'agent. Ils sont reliés par des transitions.

Utilisation d'un Réseau de Transitions Augmentés (Augmented Transition Network)

Pour donner au modèle d'agents proposé les caractéristiques citées ci-dessus, nous avons choisi de représenter le module de contrôle par un ATN [Annexe]. Ce dernier est défini par un ensemble fini d'états reliés par un ensemble de transitions. Chaque état de l'ATN décrit un état de l'agent.

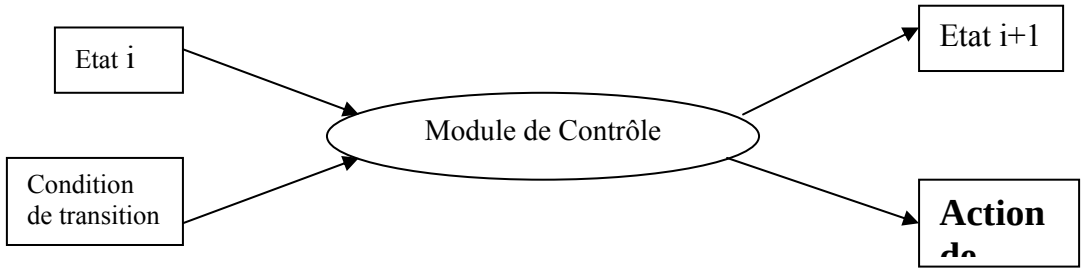


***Fig 5.5** : Transitions du module de Contrôle*

Si à partir de l'état i de l'agent, la condition de transition qui est attachée à l'état i est vérifiée, une action de transition est effectuée. Elle a pour effet de changer le statut des modules donc l'état de l'agent (état i+1).

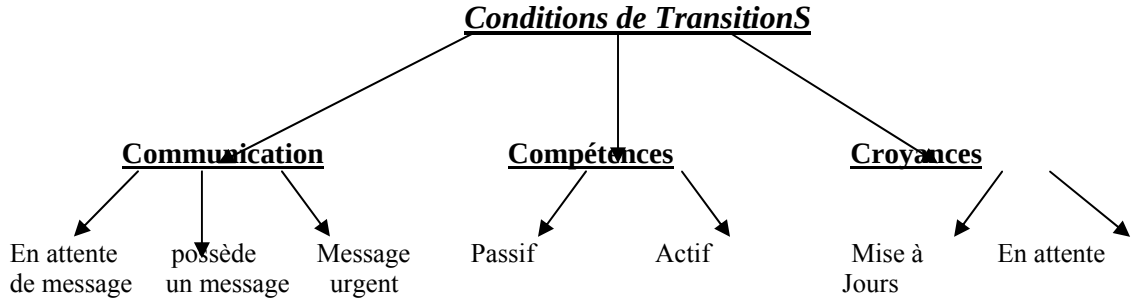


***Fig 5.6** Exemples d'états des modules de comportement*

A chaque module correspond différents états possibles qui définissent par leur combinaison l'état de l'agent.

L'état d'un agent est une combinaison de l'ensemble des états de chacun des modules de la première strate. Le réseau de transition définit a priori l'ensemble des transitions possibles qui relient un ensemble d'états que peut prendre l'agent au cours du temps.

Les conditions de transition sont les différents signaux reçus par ces modules (voir la figure 5.6). Si elles sont valides, elles entraînent alors des actions de transition c'est-à-dire des opérations sur les modules

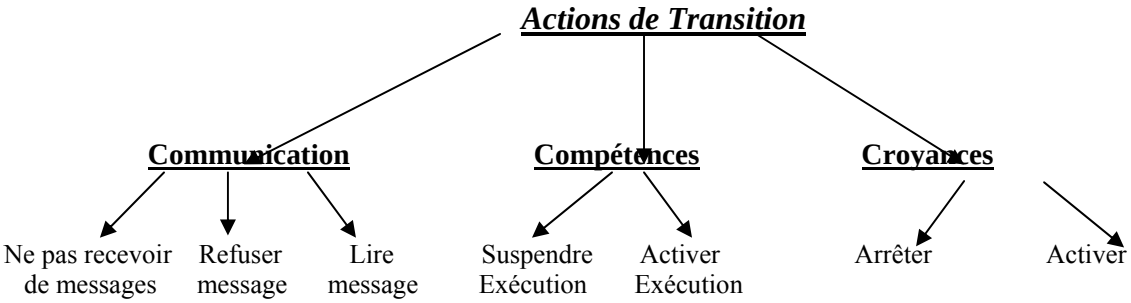


Fig 5.7 *Exemples d'actions*

Chaque module de Contrôle peut exécuter différentes actions qui permettent de changer l'état des modules de la première strate. Les principales caractéristiques des automates notamment les ATN sont les suivants :

- Les dépendances et interdépendances causales d'un certain ensemble d'événements peuvent être représentées explicitement. Les événements qui sont indépendants ne sont pas séquentialisés. Ils permettent de représenter les systèmes (simples ou complexes) à différents niveaux d'abstraction de façon déclarative et concise et d'en contrôler le déroulement.
- Ils permettent de combiner les changements qui coïncident et de déterminer l'état global résultant. Ils sont très simples et ne nécessitent aucune nouvelle théorie.
- Ils sont flexibles, simples, déterministes et efficaces.

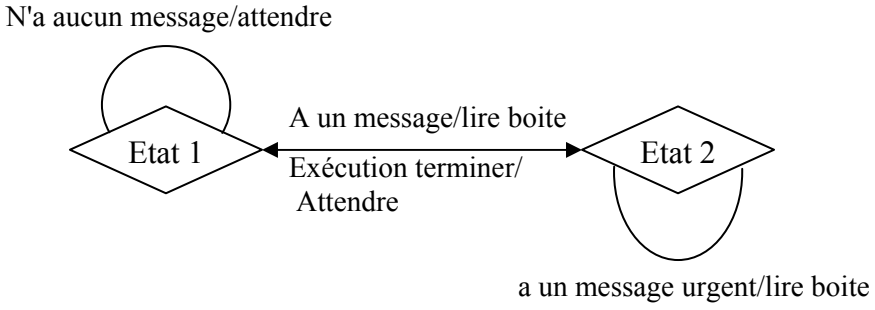


Fig 5.8 Exemple d'un ATN.

La figure 6.8 donne l'exemple d'un ATN qui gère deux modules (communication et compétences). A partir de *état1*, la condition "n'a aucun message" conduit à l'action "attendre" et au maintien au niveau de *état1* alors que la condition "a un message" entraîne l'action "lire la boîte aux lettres" et le passage à *état2*. Quel que

soit l'état courant la condition “a un message urgent” entraîne toujours l'action “lire la boîte aux lettres”. La stratégie de l'agent qui possède cet ATN consiste à privilégier le traitement de messages urgents à toute autre activité.

5.2 module de communication

Le module de communication réalise deux fonctions, valable localement et à distance :

- la réception de messages en provenance d'autres agents.
- l'envoi de messages pour communiquer avec les accointances de l'agent ou agir sur l'environnement.

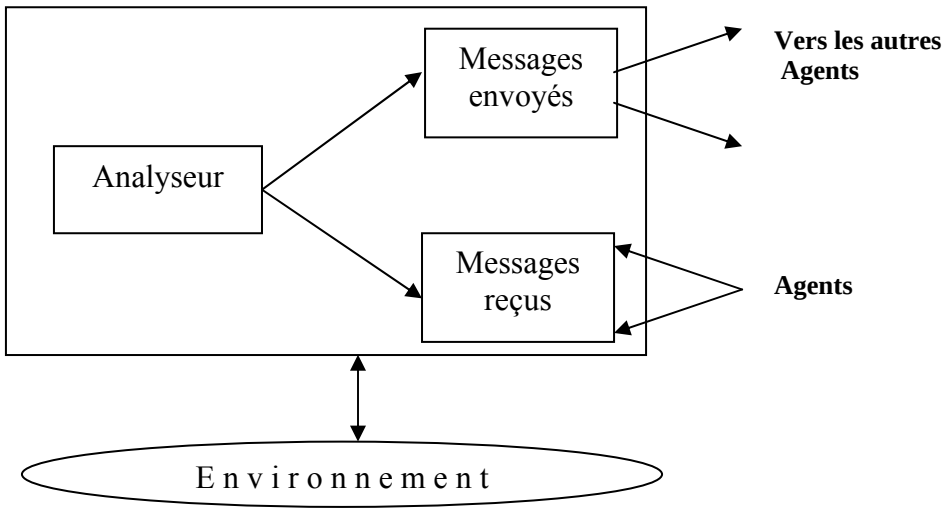


Fig 5.9 *Module de communication*

Pour la réception d'informations il va, en fonction du but qu'il poursuit, filtrer les messages qu'il reçoit, déterminer leur priorité (LIFO, FIFO, ...) et le type de traitement à leur accorder (interruption de l'action en cours pour traiter un message, ...).

Ce module a aussi pour fonction la réalisation d'actions directes (modification de l'environnement via des effecteurs) ou indirectes (transmission d'informations à d'autres agents). Les deux types d'actions sont déterminés par le module de contrôle. Le mode de transmission des informations, qui peut être sélectif, diffusé, avec accusé de réception, synchrone, asynchrone, ..., est fonction des directives 0reçues du module de raisonnement. Ce module intègre des connaissances succinctes sur les autres agents (par exemple : traiter en priorité les messages en provenance de l'agent n°1, refuser de communiquer avec l'agent n°2, les informations de l'agent n°3 sont les plus fiables, ...).

5.2.1 Communication locale

A la différence du paradigme objet où les communications sont de simples envois de message, dans les SMAs les communications sont plus sophistiquées et compliquées en même temps.

Les différents types de messages utilisés par les agents sont :

- l'émission sans retour : (AgentEmetteur, AgentRecepteur,, Message)
- l'émission avec attente de réponse : (AgentEmetteur, AgentRecepteur,, Message, Wait),
- l'émission multiple sans retour : (AgentEmetteur, Liste{AgentRecepteur}, Message)
- l'émission multiple avec attente de réponses : (AgentEmetteur, Liste{AgentRecepteur}, Message,Wait) .

5.2.2 Communication distante

Pour distribuer nos systèmes multi-agents sur un ensemble de machines, nous réutilisons la technique dite appel de procédure à distance, qui implémente la technique RPC (Remote Call Procedure). Pour exploiter cette technique, nous avons défini une classe d'agent (classe Machine) jouant le rôle d'interface. Chacune des machines utilisées possède un agent nommé AgentMachine dont le comportement est décrit par une instance de la classe Machine. AgentMachine est un agent réactif qui détient les adresses de tous les agents de sa machine. Sa principale fonction consiste à leur transmettre les messages en provenance des autres machines et éventuellement retransmettre le résultat à l'agent émetteur. Le format d'une communication distante est le suivant : (AdresseMachine, AdresseAgent, Contenu).

5.3 Module des Croyances (Perception)

Le module des croyances gère les interactions entre l'agent et son environnement. Ce dernier représente l'ensemble des entités qui compose l'environnement de l'agent. Le rôle principal de ce module est d'initialiser et de mettre à jour un ensemble de données qui regroupe les valeurs des variables externes accessibles via l'échange d'informations avec l'environnement (voir la figure 6.10). Cet ensemble de données représente un modèle de l'environnement perçu.

Ce module comprend un analyseur (interpréteur) qui définit les conditions de recueil des informations (filtrage, fréquence d'échantillonnage, ...) et. Il peut également synthétiser les informations reçus à des instants successifs concernant un même phénomène. A chaque module des croyances correspond ainsi une base de données que le module de contrôle peut consulter. Dans certains cas, les nouvelles informations disponibles (dépassement d'un seuil de référence, non fonctionnement d'un capteur, signal d'alarme, ...) imposent un traitement en temps réel par l'agent.

Le module de contrôle peut alors suspendre le raisonnement en cours pour traiter les nouvelles informations en temps réel. Ce module envoie automatiquement les nouvelles informations perçues au module de contrôle lorsqu'il a l'autorisation de celui ci.

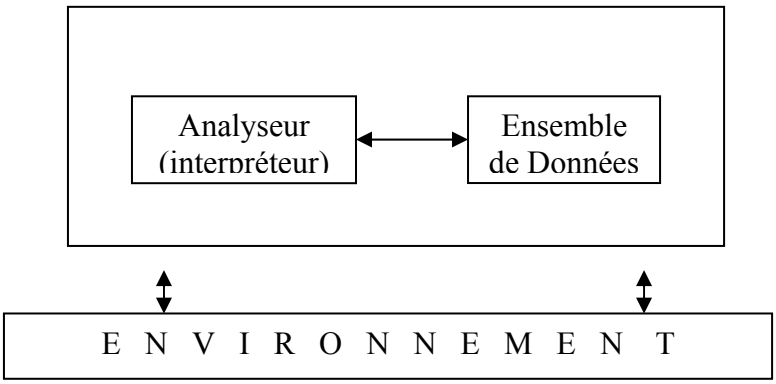


Fig 5.10 Module des Croyances

6 Le mécanisme de contrôle inter agents et de la gestion de la concurrence

Les travaux récents sur les architectures IAD proposent de décomposer chaque agent en différents modules avec un module spécifique qui contrôle l'activation des autres modules. Les agents sont conçus pour allier des capacités réactives à des capacités cognitives, ce qui leur permet d'adapter leur comportement en temps réel à l'évolution de leur environnement. Le problème est alors de définir les mécanismes et les stratégies du module de contrôle interne à l'agent pour gérer les interactions entre ses différents modules [4], c'est-à-dire imposer un séquencement temporel global interne à l'agent.

Notre but dans ce paragraphe est de présenter en détail la façon dont nous avons abordé le problème du contrôle dans notre modèle. Nous plaçons le contrôle à deux niveaux : à la fois *interne* à chaque agent et *inter-agents*. Le contrôle interne est basé sur l'utilisation d'un ATN et sur la représentation déclarative *du* contrôle du raisonnement propre à l'agent. Le contrôle inter-agents est assuré par l'utilisation de mécanismes appropriés pour gérer la concurrence entre agents et éviter les incohérences dues aux accès et partage de ressources.

Pour simuler le parallélisme nous avons choisi un contrôle de l'allocation de processus à deux niveaux : le premier niveau est celui du module de contrôle (simulation du parallélisme entre agents) et le second celui du comportement de l'agent représenter par les modules de communication, des croyances et des compétences (simulation du parallélisme interne à l'agent).

6.1 Contrôle interne

L'autonomie des agents est un objectif qui nécessite un bon mécanisme de contrôle : il doit être d'une part intelligent et d'autre part adaptatif. On définit le contrôle intelligent comme le processus d'une prise de décision autonome dans un environnement structuré ou non structuré. Une entité autonome fournit donc une adaptation à son environnement et à ses objectifs.

Les agents autonomes doivent être capables de planifier dynamiquement la séquence des actions de contrôle nécessaire pour accomplir une tâche complexe. Ils doivent pouvoir gérer les événements externes en temps réel. Ceci nécessite l'intégration des différents types de contrôle.

Le contrôle interne que nous proposons est géré à deux niveaux (voir la figure 9) :

- au niveau du module de Contrôle : un ATN permet de spécifier le comportement de l'agent en fonction des états internes observés,
- au niveau du parallélisme interne : la gestion des processus associés aux différents modules asynchrones de l'agent permet de spécifier l'allocation des ressources disponibles. Ces deux types de contrôles sont détaillés dans les paragraphes suivants.

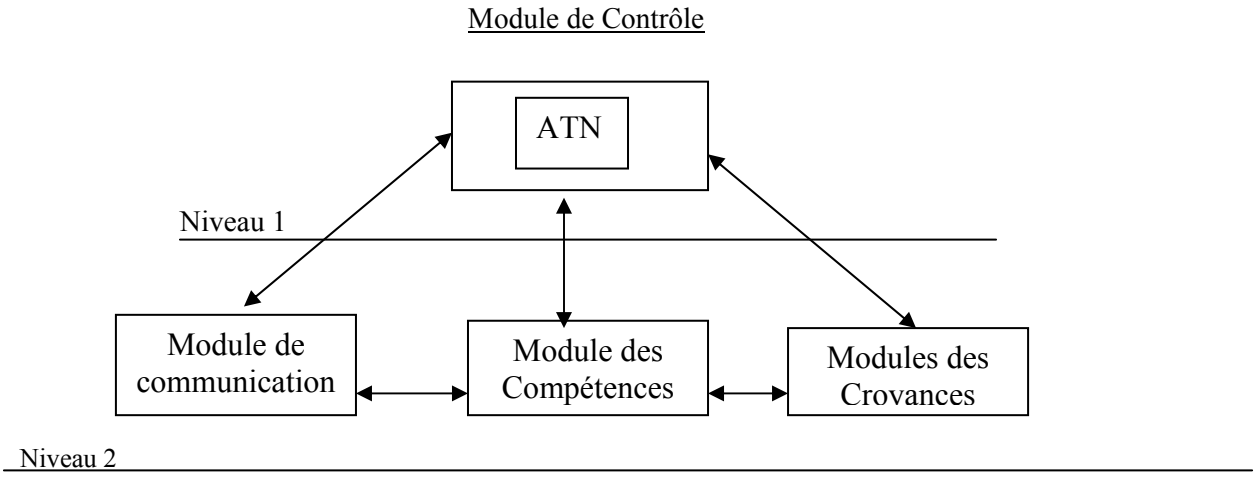


Fig 5.11 *Schéma de Contrôle interne à l'agent*

6.1.1 Niveau Module de contrôle

La complexité des modèles des systèmes dynamiques nécessite l'utilisation de contrôleurs complexes et sophistiqués. Pour pouvoir évoluer dans un environnement dynamique, chaque agent doit disposer d'un mécanisme de contrôle adaptatif. A cet effet, nous avons choisi d'associer à chaque agent, un module de contrôle représenté par un ATN. Ce dernier permet à l'agent d'adapter dynamiquement son comportement aux changements détectés par les modules représentants son comportement. L'adaptation correspond à un nouveau mode d'organisation de l'agent dans ses rapports décisionnels, ses interactions avec son environnement ou encore ses relations avec les autres agents. L'agent est alors capable de contrôler ses aptitudes (à communiquer, comportementales, etc.) selon l'agent avec lequel il interagit et selon l'environnement où il se trouve.

Ce module repose sur les notions *d'état* et de *transition*. Les états définissent le contexte dans lequel prennent place les événements qui marquent la vie de l'agent. Les transitions décrivent les réactions de l'agent à ces événements. Ces états et transitions sont définis par un ATN qui est une représentation synthétique et déterministe du comportement de l'agent.

Cette notion d'événement convient aux besoins d'expression des modèles dynamiques en leur fournissant un mécanisme approprié de contrôle de l'évolution de leur comportement. Les transitions décrivent les réactions de l'agent à ces événements. Elles définissent les contraintes portant sur le séquencement des exécutions et elles permettent de filtrer les événements qu'un agent peut traiter. Elles permettent également d'adapter le comportement de l'agent aux changements que ces événements entraînent.

L'introduction des comportements transitionnels de l'agent est motivée par deux besoins complémentaires :

- l'ordonnancement des actions indépendamment du fait qu'elles puissent ou non s'exécuter de façon concurrente,
- la synchronisation des actions exécutées de façon concurrente afin de préserver la cohérence de leur traitement.

6.1.2 Au niveau des autres modules : une gestion de processus

L'activité de l'agent consiste soit à exécuter une ou plusieurs compétences sollicitées par d'autres agents, soit à mettre à jours sa base de données relative à ses croyances sur l'environnement ou les autres agents. Cependant l'activité de chaque module (communication, croyance et compétences) est vue comme l'activité d'un processus. Le modèle simulé est alors décrit comme un ensemble de processus progressant en quasi-parallèle dans le temps. Le contrôle de processus consiste à décrire le fonctionnement pas à pas de ces processus. Il s'agit alors de définir ces pas, c'est-à-dire la granularité du parallélisme. Cette granularité est inversement proportionnelle au nombre d'interruptions. En effet, une grande granularité permet une bonne simulation du parallélisme et une petite granularité offre une rapidité dans l'exécution.

Les différents modules sont asynchrones et concurrents. En effet, à chaque module est associé un processus. Pour simuler le parallélisme nous avons choisi un contrôle de l'allocation de processus au niveau des modules de communication, croyances et compétences (simulation du parallélisme interne à l'agent).

Au niveau du module de communication, le contrôle de processus est effectué après chaque lecture de la boîte aux lettres à l'aide de la méthode WaitCommunication. Au niveau du module des croyances, le contrôle de processus est effectué à la fin de la méthode d'interprétation des informations en provenance des agents et de l'environnement par l'exécution du message WaitCroyances. Au niveau du module des compétences le traitement n'est suspendu qu'en cas de message prioritaire. Ces différents messages

(WaitCommunication, WaitCroyances et WaitCompétences) permettent au module de Contrôle d'observer et de contrôler l'évolution de ces processus.

6.2 Contrôle inter-agents et gestion de la concurrence

Un système multi-agents est composé donc d'un ensemble d'entités plus ou moins autonomes appelées à coopérées et à coordonnées leurs actions pour atteindre l'objectif décidé par le concepteur du système. Cependant de cette vie artificiellement interactive émerge des conflits entre agents, dues pour la plus part des cas au besoin d'accès aux ressources partageable mis à disposition des agents.

Cette émergence de conflit impose naturellement aux concepteurs de plates formes multi-agents, la prise en compte de la résolution de ces situations conflictuelles, en introduisant des mécanismes ou des outils permettant la résolution des différends.

Dans le souci de respecter le principe d'autonomie des agents, en évitant d'introduire un mécanisme de contrôle centralisé, nous jugerions que le meilleur moyen pour permettre la résolution des conflits est bien entendu l'introduction du concept de Schéduling, qui est l'un des piliers de la conception des plates formes orientées agents.

Dans cette partie on va voir en quoi consiste le schéduling et quels sont les différents types de techniques existant et laquelle choisir.

6.2.1 Le graphe de dépendances

Il est indispensable que les modifications introduites par une phase de traitement de l'agent ne soient pas seulement locales à l'état interne de l'agent, mais diffusées à tous les agents dont les connaissances peuvent en être affectées. Chaque agent doit connaître tous les autres agents qui utilisent les mêmes ressources que celles qu'il modifie. Pour cela, nous distinguons deux types de ressources : les *ressources locales* référencées par un seul agent et les *ressources communes* référencées par plusieurs agents. Le principe du mécanisme de dépendances peut-être défini en deux points :

- On associe à chaque agent un graphe de dépendances donnant pour chaque ressource commune la liste des autres agents qui l'utilisent.

- Le graphe de dépendances est utilisé par l'agent pour informer les autres agents de la modification, de la création ou de la suppression d'une ressource commune. Il est mis à jour progressivement quand les règles de production utilisant des ressources communes sont déclenchées. Par exemple, si une ressource commune est supprimée par les actions de la règle, elle est automatiquement supprimée du graphe de dépendances et un message est diffusé aux autres agents qui l'utilisent pour qu'ils mettent à jour leurs graphes de dépendances.

Les conflits qui peuvent résulter des accès aux ressources partagées sont ainsi évités.

6.2.2 La concurrence entre agents

Comme Axtell l'a encore montré récemment [15], les résultats d'une modélisation multi-agents sont fortement influencés par la façon dont celle-ci est implémentée. Notamment, il est clair que la technique employée pour gérer l'exécution des agents (scheduling) a un impact crucial sur les résultats. Cette observation trouve une explication dans le fait que nous ne disposons actuellement pas de formalisme adéquat pour représenter simplement des interactions collectives. Plus particulièrement, nous éprouvons une difficulté à modéliser la simultanéité des actions des agents [6].

Cependant, peu de travaux portent sur l'analyse des biais engendrés directement par ce problème. Cette problématique, qu'on désignera ici par ***problème du scheduling*** ou autrement dit la ***politique d'exécution des agents***, n'a en effet quasiment aucun support méthodologique alors que, paradoxalement, c'est un passage obligé lors de l'implémentation d'une plate forme multi-agents

En effet, le paradigme multi-agents est fondé implicitement sur la composition de comportements individuels concurrents. Par ailleurs, les architectures multiprocesseurs n'apportent pas de solution évidente : sans synchronisation les agents évoluent au rythme de la complexité de leur architecture interne, ce qui ne permet pas de contrôler la cohérence de la dynamique globale du système. Ainsi toutes les solutions envisagées pour modéliser un système multi-agents (SMA) sont génératrices de biais dans le processus de modélisation. Autrement dit l'implémentation de ce point précis à un impact direct sur l'évolution du modèle multi-agents.

Nous pensons que la conception et l'analyse du scheduling doit prendre une place prépondérante lors de la conception d'une plate forme multi-agents. Il est donc impératif de se donner des moyens méthodologiques et pratiques qui permettent une analyse simple et explicite de ce problème.

6.2.2.1 Le scheduling : au cœur de la conception des plate-forme Multi-Agents

Si de façon classique une modélisation consiste à expérimenter des modèles donnés sous la forme de relations mathématiques entre des variables, au contraire la modélisation multi-agents se propose de représenter directement les individus, leurs comportements et leurs interactions [6].

L'étude de la problématique liée à l'implémentation de ces systèmes complexes est en général centrée autour d'une ou plusieurs des notions suivantes : le type des agents modélisés (communicant, situé, réactif, cognitif, …), l'environnement dans lequel évoluent les agents (discret, continu, déterministe, 2D, 3D, …) et la nature des interactions (coordination, négociation, perceptions et actions sur l'environnement, …).

Une telle approche a l'inconvénient de ne pas faire apparaître de façon explicite les problèmes liés à l'implémentation de la représentation du temps. Alors qu'il est clair que chaque modèle ne nécessitera pas le même niveau de granularité temporelle : du temps réel pour la robotique mobile à la représentation de plusieurs années en écologie. Par ailleurs, modéliser un SMA suppose qu'on a implémenté un mécanisme permettant de synchroniser les actions des agents. Soit Σ définissant l'ensemble des états possibles du système, toute

simulation est basée sur l'hypothèse que l'évolution du monde $\sigma \in \Sigma$ de l'instant *t* à *t+dt* résulte de la composition des actions *A1(t), A2(t), …, An(t)* produites par les agents à l'instant t. Autrement dit, il s'agit de construire une fonction du temps, *Dynamique* D, telle que :

$$\mathbf{D : \Sigma \rightarrow \Sigma}$$

$$\mathbf{s(t+dt) = D (\Pi Ai(t), s(t))}$$

Le symbole Π est, ici, utilisé pour désigner l'opérateur de composition des actions. Il définit de quelle manière les actions produites à un instant *t* doivent être sommées afin de calculer leurs conséquences sur l'état du monde. Sans entrer dans le détail, il est facile de mesurer la difficulté de conceptualiser une telle opération étant donné la multitude et la nature des concepts qui peuvent se cacher derrière le mot action (mouvement, prise de décision, modification de l'environnement). C'est pourquoi il n'existe pas de consensus sur la manière dont le déroulement des interactions doit être modélisé et tous les concepteurs de systèmes multi-agents sont amenés à faire un choix personnel en la matière.

Dans la suite de cette partie, on va présenter les techniques du scheduling les plus connu dans le domaine des systèmes multi-agents puis on fera le point sur la technique retenue pour notre plate forme.

6.2.2.1.1 La Technique à pas de temps constant

Comme le montre la figure 1, cette méthode consiste à activer les agents séquentiellement. L'activation de l'ensemble du système correspond alors à un pas de temps. Cette technique a l'avantage d'éviter la présence de conflits dans l'accès aux variables de l'environnement et de garantir que tous les agents agissent une fois par cycle. Par ailleurs, c'est de loin la technique la plus utilisée du fait de sa simplicité. Par exemple, les plates-formes Multi-Agents STARLOGO et MANTA [35] utilisent cette technique.

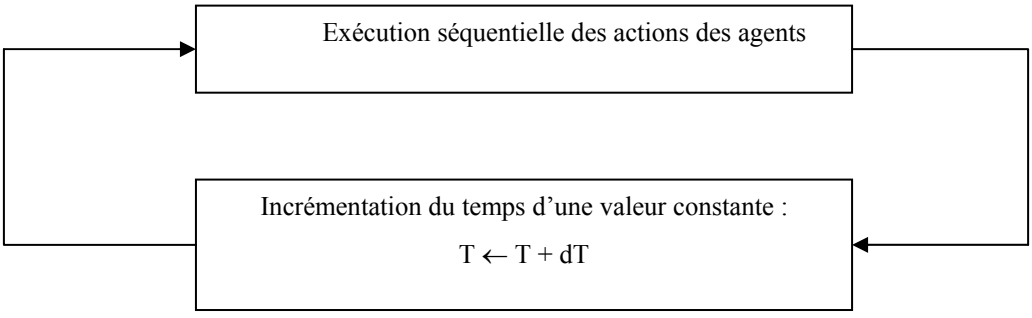


Fig 5.12 *Principe de la technique à pas de temps constant*

6.2.2.1.2 La technique dite avec état tampon ou « double buffer »

Extension de la précédente, cette technique vise à apporter une solution au problème de la simultanéité. L'objectif est de faire en sorte qu'à un instant *t* tous les agents aient la même perception de l'état du monde s(t). Pour cela les actions des agents se font sur des variables tampons et le monde n'est pas directement modifié. Une fois tous les agents exécutés, on fait la synthèse de l'ensemble des actions pour calculer le nouvel état du monde.

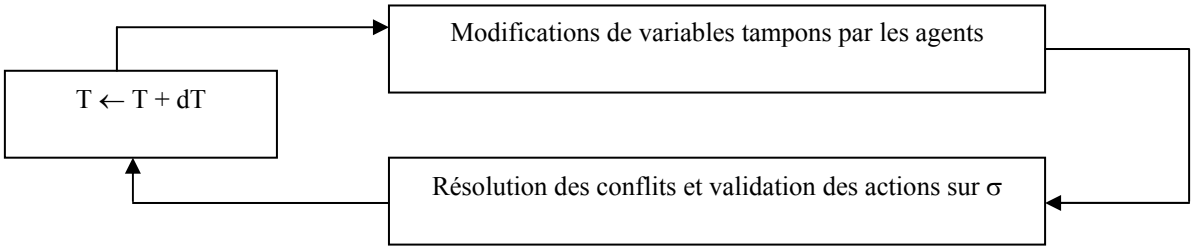


Fig 5.13 *Un modèle avec état tampon*

Les simulations de type *jeu de la vie* sont basées sur ce principe : l'état du monde est mis à jour une fois que toutes les cellules ont calculé leur état suivant. La plate-forme Multi-Agents LIVEWORLD [6.13], propose ce mode de fonctionnement. Cette méthode entraîne cependant directement un nouveau problème. Un conflit apparaît lorsque deux agents ou plus spécifient de nouvelles valeurs différentes pour une même variable. Il faut donc éventuellement résoudre les conflits aboutissant à une incohérence de l'état du monde.

6.2.2.1.3 La Technique basée événements

Plutôt que de synchroniser tous les agents à un instant t, l'idée de cette méthode est d'exhiber explicitement un ordre chronologique entre les actions des agents. Le principe consiste à déterminer a priori ou en cours de conception les événements futurs, leur date et leur nature.

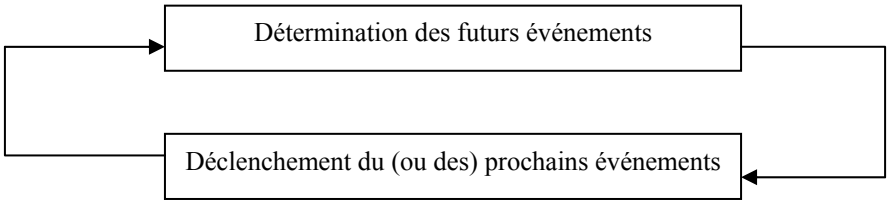


Fig 5.14 *Principe d'une modélisation par événements*

Magnin [11] utilise par exemple dans son simulateur SIEME un ensemble de règles environnementales permettant de déterminer au fur et à mesure des interactions les événements à déclencher :

Si <condition événement> alors <déclencher événement>

Ainsi un principe de causalité entre événements est au cœur de ses préoccupations. Au contraire dans la plate-forme Multi-Agents SWARM [22] la chronologie des interactions est déterminée a priori sous la forme d'une collection structurée d'objets à laquelle est associé un ordonnancement. Celui-ci définit alors la logique de la dynamique du système.

Bien que ce type de technique semble loin des précédents, il n'en reste pas moins qu'on retrouve les mêmes genres de problèmes. En effet, lorsque plusieurs événements sont concurrents les difficultés liées à la modélisation restent les mêmes.

6.2.2.2 Discussion

Notre apport ici est de présenter une technique de schéduling qui peut être vue comme une extension de la technique dite « double buffer » à deux niveaux. En premier lieu tous les agents du système, et à l'instant t, possèdent la même perception de l'environnement et leurs actions se déroule, bien sûr, sur des variables tampons. Une fois les agents ont terminés leurs travail , on passe à la phase de synthèse et de validation de l'environnement avec cette fois ci l'introduction d'un ordre chronologique sur les actions exécutées par les agents ; autrement dit si une variable à subie plusieurs spécification de sa valeur, la valeur qui sera retenu par l'environnement est la première valeur spécifier, et le noyau de la plate forme envoi un compte rendu aux autres agents concernés par cette variable pour une éventuel prise en compte de la nouvelle valeur.

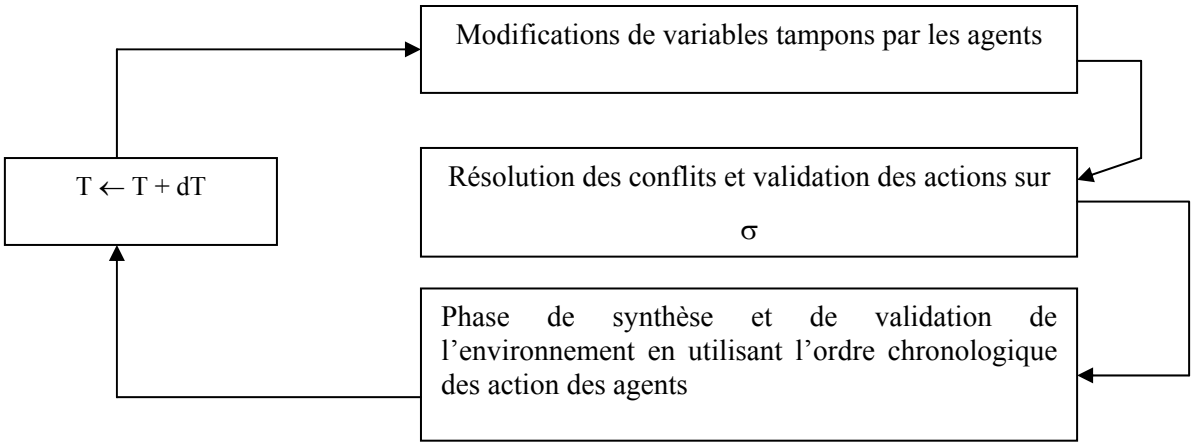


Fig 5.15 *principe d'une modélisation par double buffer avec un ordre chronologique*

7 l'Organisation Hiérarchique

La notion d'organisation, comme on la vu dans le cinquième chapitre, est intensivement étudiée dans le domaine. Elle constitue l'une des quatre dimensions retenues dans le domaine de recherche sur les SMAs. Ces

dimensions sont d'après Demazeau [34] : l'agent (A), l'environnement (E), l'interaction (I) et l'organisation (O).

Jacques Ferber [6] indique que «L'organisation est, avec l'interaction, l'un des concepts de base des systèmes multi-agents. »

Toutefois, il s'agit d'une notion complexe et difficile à définir. Le mot « organisation » est utilisé pour indiquer l'agencement de relations entre agents, un concept abstrait.

Ainsi le nombre d'agents qui diffère d'un système à un autre, allant d'un agent à des centaines d'agents pour les grandes applications comme la gestion d'un aéroport, par exemple, impose la mise en œuvre d'une organisation des agents afin d'éviter la tendance du système vers l'anarchie. Dans un système de grande taille surgit naturellement les problèmes des goulots d'étranglement, dus essentiellement au nombre de communications croissant entre agents et qui risquent de mettre en péril toute l'organisation du système entier. Notre contribution ici porte sur la troisième disparité entre objet et agent, c'est-à-dire sur le plan comportement ou la flexibilité du système, en introduisant l'hiérarchisation des agents, tel qu'on la connaît dans la réalité, comme moyen d'organisation du système.

Il s'agit de répartir les compétences parmi les agents du SMA puis d'organiser ces agents dans une hiérarchie. On dispose donc d'un SMA hiérarchique correspondant à la structure présentée à la figure ci-dessous. Rappelons que l'existence de cette structure hiérarchique permet la gestion automatique de la gestion des invocations des compétences. Ainsi lorsque une compétence comp1 de l'agent1 requiert la compétence comp2 de l'agent2, l'agent1 n'a pas besoin de connaître explicitement l'agent2, le système gère l'acheminement de la requête via un autre agent qu'on peut appeler pour la circonstance SuperAgent. En fait la seule chose qui importe est de savoir que la compétence comp2 est présente dans le SMA ce qui garanti une flexibilité du système face aux éventuels changements possible comme l'ajout de nouveaux agents ou compétence.

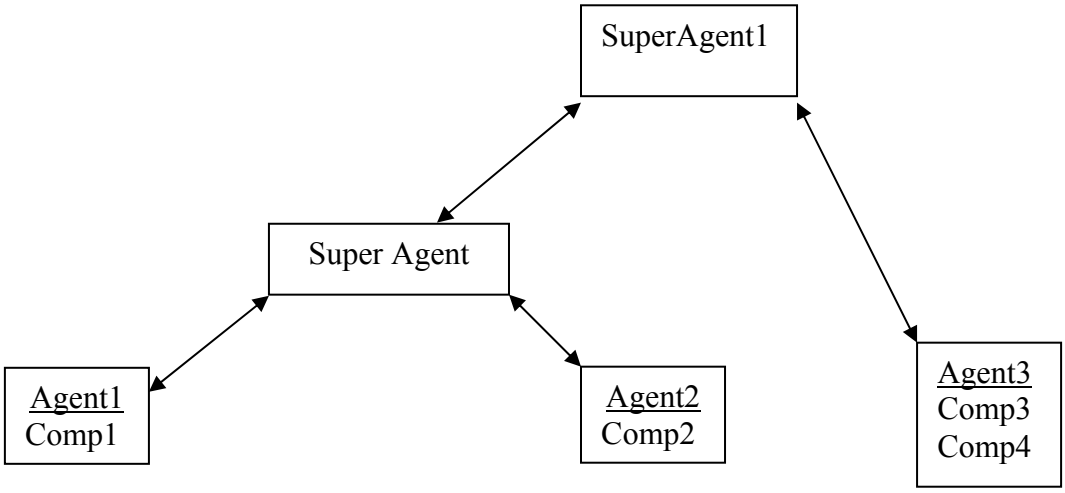


Fig 5.16 *Exemple d'une hiérarchie entre agent*

Dans l'exemple schématisé par la figure ci-dessus l'agent1 n'est pas censé connaître l'existence de l'agent3 ou de ces compétence, en cas de besoin il fait appel à l'agent SuperAgent2 qui se charge du reste.

8 Conclusion

Dans la plate forme Multi-Agent NewObject tout est vu en tant qu'agent. On peut aller des agents les plus simple, qui calcul par exemple une valeur, aux agents les plus compliqués, qui contrôle par exemple l'atterrissage d'un avion.

Toutes les classes héritent d'une seul super classe, OneClasse qui contient la structure de base, voir minimale, de chaque agent du systèmes modélisé. On trouve dans cette super classe les modèles des modules de contrôle, de communication et de croyances.

Pour terminer ce chapitre, on signale que les notions d'héritage et la notion de classe sont utilisées dans notre plate forme de la même manière que dans le paradigme objet.

Chapitre VI

Les Méthodologies de Coneptions des SMAs

Résumé : *Les systèmes multi-agents (SMA) sont actuellement très largement utilisés, particulièrement pour les applications complexes nécessitant l'interaction entre plusieurs entités. Plusieurs méthodologies ont été proposées pour le développement des SMA. Ces méthodologies, constituant soit une extension des méthodologies orientées-objet, soit une extension des méthodologies à base de connaissances, demeurent incomplètes. Par ailleurs, peu d'efforts ont été faits en matière de standardisation des plates-formes SMA. Il apparaît donc évident que le développement des SMA reste encore un domaine ouvert. Le succès du paradigme orienté-agent exige des méthodologies systématiques pour la spécification et la conception des applications SMA.*

1 Introduction

Notre travail de recherche se place naturellement dans le contexte du génie logiciel et de l'intelligence artificielle distribuée où nous nous occupons de plus pré de l'aspect génie logiciel dans le développement des SMA ; dès lors, il est devenu impératif de s'investir dans les méthodologies orientées-agent.

Les systèmes multi-agents (SMA) sont actuellement très largement utilisés, particulièrement pour les applications complexes nécessitant l'interaction entre plusieurs entités. Plusieurs méthodologies ont été proposées pour le développement des SMA. Ces méthodologies, constituant soit une extension des méthodologies orientées-objet, soit une extension des méthodologies à base de connaissances, demeurent incomplètes. Par ailleurs, peu d'efforts ont été faits en matière de standardisation des plates-formes SMA. Il apparaît donc évident que le développement des SMA reste encore un domaine ouvert. Le succès du paradigme orienté-agent exige des méthodologies systématiques pour la spécification et la conception des applications SMA.

Dans ce chapitre, nous allons décrire brièvement les principales méthodologies SMA que nous avons recensées dans la littérature. Il y a, parmi celles-ci, certaines sur lesquelles nous ne possédons pas assez d'informations nous permettant de faire une étude détaillée. Il nous semble cependant que les méthodologies que nous avons étudiées, sont les principales et sont les plus représentatives des méthodologies SMA existantes.

2 Les méthodologies constituant une extension des méthodes orientées-objet

Les principaux avantages de ces approches sont :

- un agent peut être considéré comme un objet actif ayant un état mental, un objectif et une autonomie;
- les deux paradigmes (objet et agent) utilisent l'envoi de messages pour la communication, et peuvent utiliser l'héritage et l'agrégation pour définir leurs architectures;
- les méthodes orientées-objet sont populaires, en ce sens que plusieurs d'entre elles ont été utilisées avec succès dans l'industrie. L'expérience et le succès liés à cette utilisation peuvent faciliter l'intégration de la technologie agent;
- les modèles objet, dynamique et fonctionnel de ce paradigme peuvent être utilisés pour décrire les agents;
- les cas d'utilisation et les diagrammes de collaboration peuvent être utilisés dans l'identification des agents.

Mais, les méthodes orientées-objet telles quelles ne fournissent pas de techniques adéquates pour caractériser certains aspects spécifiques aux agents, tels que : leur état mental, le type des messages, le protocole de négociation, leur dimension sociale.

Dans ce qui suit, nous présentons des méthodes mieux adaptées aux agents.

2.1 La méthodologie GAIA

Selon [4], GAIA est le nom donné aux hypothèses formulées par l'écologiste James Lovelock selon lesquelles les organismes vivants du monde peuvent être compris comme des composantes d'une seule entité, lesquelles composantes régularisent l'environnement mondial. Les systèmes auxquels on peut appliquer cette méthodologie contiennent un petit nombre d'agents (moins de 100). Les auteurs de GAIA considèrent un SMA comme une société ou une organisation artificielle ou encore un ensemble institutionnalisé de rôles. Cette idée vient du fait que dans une compagnie, il y a un poste de président, un poste de vice-président, etc.

L'instanciation (qui peut-être dynamique) de ces postes est fonction des individus qui sont présents à l'époque.

GAIA se compose de deux phases principales (figure 6.1) :

- la phase d'analyse qui produit deux modèles : le modèle de rôles et le modèle d'interactions;
- la phase de conception qui transforme les modèles abstraits de la phase d'analyse en modèles moins abstraits. Le processus de conception se base sur les trois modèles modèle d'agent, modèle de service et modèle de relation.

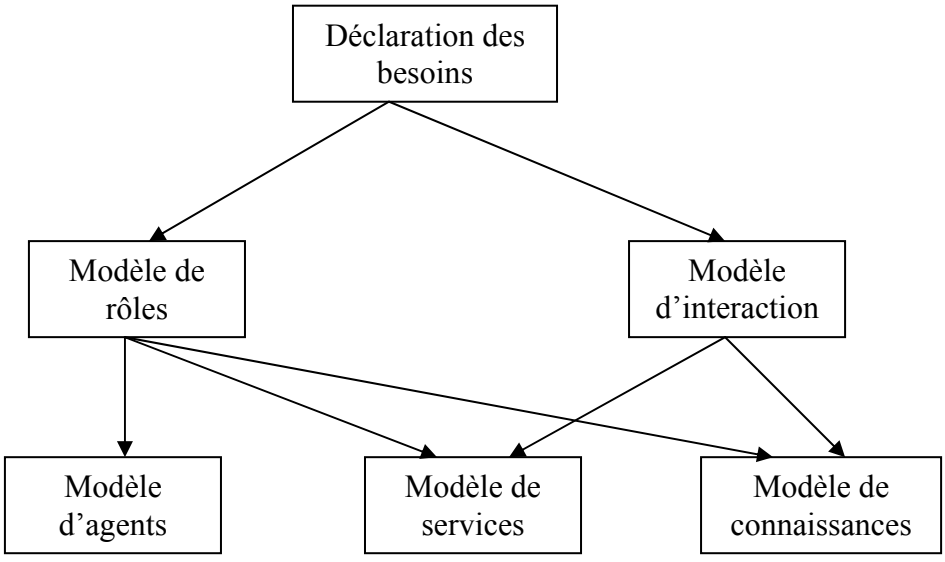


***Fig 6.1** : Les modèles de GAIA*

2.2 La méthode MaSE (Multi-agent Systems Engineering)

Les auteurs de cette méthodologie [43] définissent un agent comme un objet actif ayant un objectif et un langage de communication. Cette méthodologie utilise les techniques de OMT (Object Modeling Technique) [37] ou de UML (Unified Modeling Language) [44] avec quelques caractéristiques de plus et quelques modifications de la sémantique du paradigme objet pour pouvoir capter les concepts d'agent et les

comportements coopératifs des agents. Elle utilise deux langages pour décrire les agents et les SMA : Agent Modeling Language (AgML) et Agent Definition Language (AgDL).

Les différentes phases de MaSE (figure 6.2) sont :

- Conception du domaine (Domaine Level Design);
- Conception d'agents (Agent Level Design);
- Conception de composantes (Component Design);
- Conception du Système (System Design).

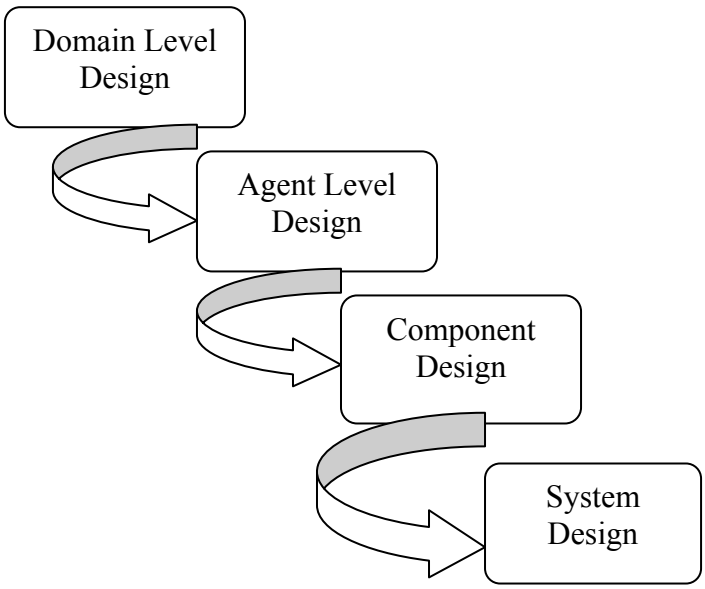


Fig 6.2 ***: Les différentes phases de MaSE***

2.3 La méthode HLIM (High-Level and Intermediate Models)

La méthodologie HLIM [45] fait ressortir, à travers deux phases, les aspects des agents tels que : leurs objectifs, leurs plans, leurs croyances et les rapports entre ces agents.

Comme les techniques orientées-objet capturent les objets dans le système, leurs attributs, leur structure et leurs liens, HLIM capture les agents, leurs attributs, leur structure et leurs liens.

La phase de découverte, qui est une phase exploratoire du domaine, est la première phase de HLIM qui produit le modèle High-Level Model. Ce modèle capture la structure et le comportement du système.

La deuxième phase de définition produit les 4 modèles suivants :

- Internal Agent Model;
- Relationship Model;
- Conversational Model;

- Contract Model.

La figure 6.3 donne une vue globale des différents modèles de HLIM avec leur séquencement.

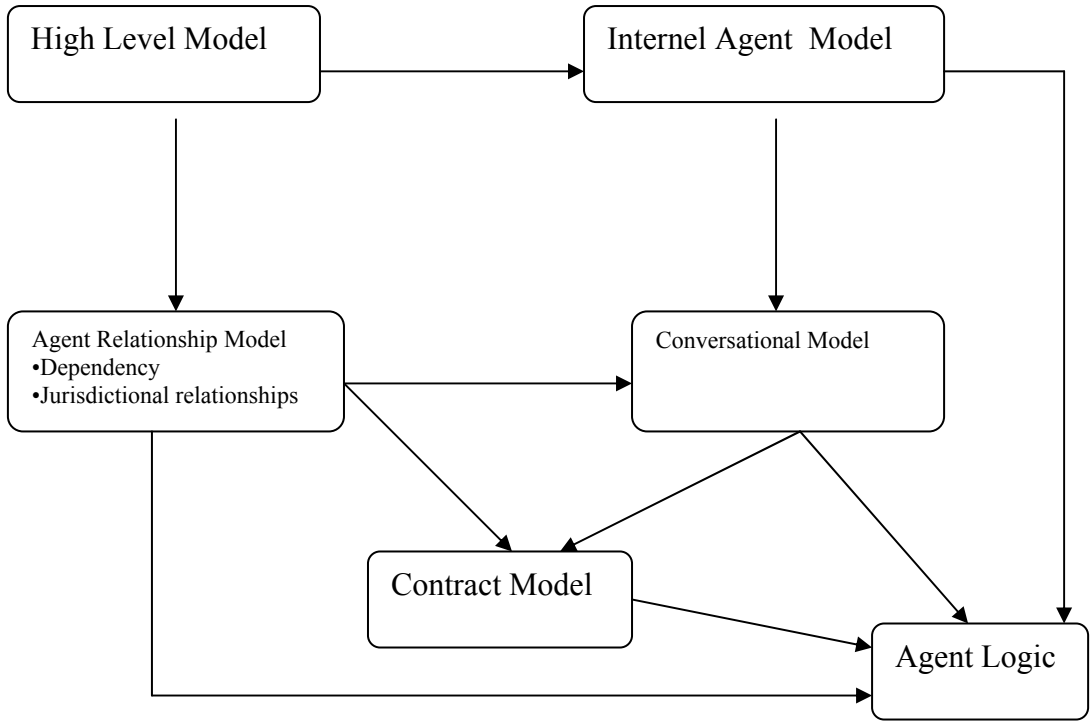


*Fig 6.3***: *Les modèles de HLIM***

2.4 MMTS (A Methodology and Modeling Technique for Systems of BDI agents)

Cette méthode [46] définit deux principaux niveaux pour la modélisation des agents BDI :

- le niveau externe (qui utilise la notation OMT) consiste en la décomposition du système en agents et la définition de leurs interactions. Ceci se traduit à travers deux modèles : le modèle d'agent et le modèle d'interactions ;
- le niveau interne fait ressortir la modélisation de chaque classe agent BDI à travers trois modèles : le modèle de croyance, le modèle d'objectif et le modèle de plan.

2.5 AOAD (Agent-Oriented Analysis and Design)

Burmeister [BUR99], l'auteur de AOAD définit trois modèles pour l'analyse d'un système agents :

- le modèle d'agent : il définit les agents internes au système et leur structure (croyance, plans, objectifs, etc.). Les agents et leurs environnements sont identifiés en utilisant une extension des diagrammes de collaboration pour inclure, la croyance, les motivations, les plans et les attributs de coopération;
- le modèle organisationnel : ce modèle décrit les rapports entre agents du système (l'héritage et les rôles dans l'organisation). Cette description utilise la notation OMT;

- le modèle de coopération : il décrit les interactions entre agents.

2.6 Multi-Agents Scenario-Based Method (MASB)

Cette méthode [47] est destinée aux SMA coopératifs et est composée essentiellement de deux phases:

- la phase d'analyse qui se compose des modèles suivants : modèle de scénarios, modèle fonctionnel de rôle, modèle conceptuel de données, modèle d'interactions utilisateur-système;
- la phase de conception est constituée des modèles suivants : architecture SMA, modèle d'agent, modèle d'objet.

2.7 Agent-Oriented Methodology for Enterprise Modeling (AOMEM)

Cette méthode [48] propose une combinaison de Object-Oriented Software Engineering (OOSE) [49] des méthodes de modélisation d'entreprises IDEF (Integration Definition for Function modeling) [50] et de CIMOSA (Computer Integrated Manufacturing Open System Architecture) [51]. Elle est constituée des modèles suivants : le modèle fonctionnel, le modèle des cas d'utilisation, le modèle dynamique et le système agent.

3 Les méthodes constituant une extension des méthodes à base de connaissance

Les méthodes à base de connaissance fournissent des techniques pouvant prendre en compte l'état mental des agents. De plus, ces méthodes possèdent une librairie d'outils pouvant être utilisés. Cependant, ces méthodes ne peuvent pas modéliser le comportement social des agents dans un SMA.

Plusieurs méthodes pour la modélisation des SMA, étendant la méthode CommonKADS, [62] ont été proposées (CommonKADS étant la méthode standard de modélisation des connaissances en Europe).

3.1 La méthode CoMoMAS

Un agent est vu ici comme une entité ayant des compétences réactives, cognitives coopératives et/ou sociales. La méthodologie CoMoMAS [52] utilise la syntaxe de CML (Conceptual Modeling Language) pour décrire ses différents modèles qui sont : le modèle d'agent, le modèle d'expertise, le modèle de tâche, le modèle de coopération, le modèle du système et le modèle de conception (figure 6.4).

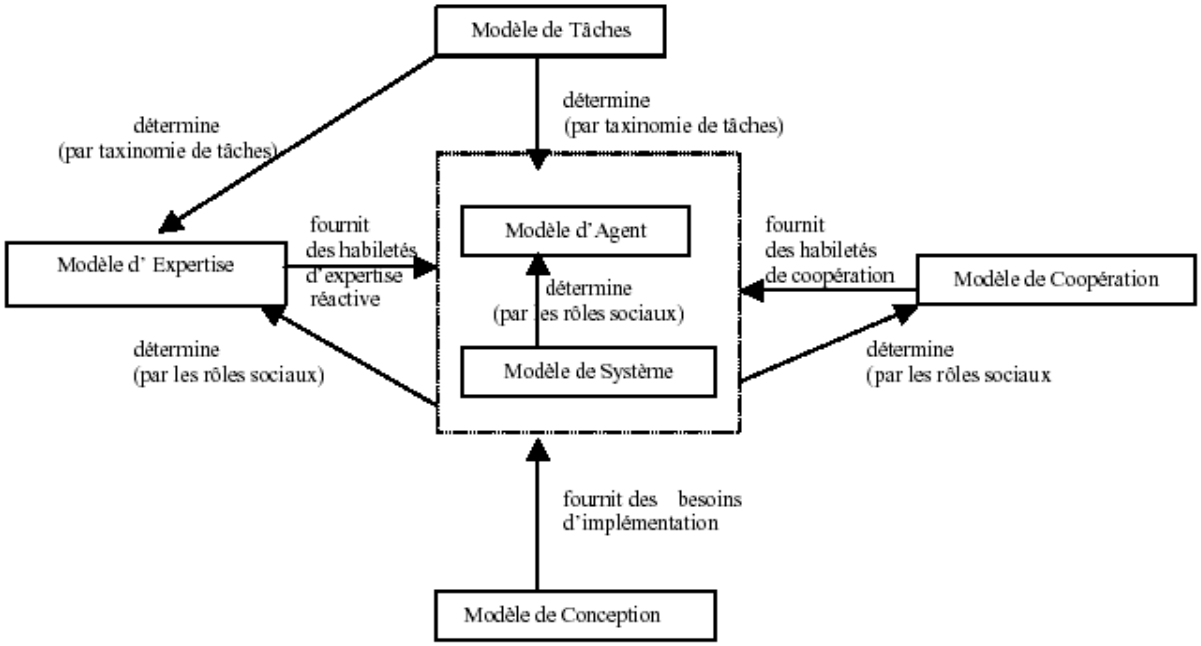


***Fig 6.4** : Les modèles de CoMoMAS*

3.2 La méthode MAS-CommonKADS

Cette méthode [53] est une combinaison de CommonKADS, OOSE (Object Oriented Software Engineering), OMT (Object Modeling Technique) et des protocoles d'agents SDL (Specification and Description Language) [54] et MSC96 (Message Sequence Charts) [55]. Les différents modèles utilisés dans cette méthode sont : modèle agent, modèle des tâches, modèle d'expertise, modèle de coordination, modèle organisationnel, modèle de communication et modèle de conception.

4 Les méthodes qui ont été conçues pour un contexte particulier

4.1 La méthode Cassiopée

Cette méthode [35], dédiée à la robotique collective, distingue trois principales étapes dans la conception d'un SMA :

- les comportements élémentaires des agents sont listés en utilisant les techniques orientées-objet;
- les dépendances entre les agents sont étudiées en utilisant un graphe de couplage. Un graphe de couplage contient les dépendances entre rôles (des agents) qui sont jugées pertinentes pour l'accomplissement collectif d'une tâche. Les chemins et les circuits élémentaires de ce graphe définissent les possibilités de regroupement des différents rôles du domaine et fournissent ainsi une représentation globale de la structure des organisations auxquelles les agents, en jouant ces rôles, peuvent appartenir [35];
- Les dynamiques de l'organisation sont décrites en analysant le graphe de couplage.

4.2 La méthode CIAD (Cooperative Information Agents Design)

Selon [53], cette méthode [56] est destinée aux processus de commerce et est composée des modèles suivants :

- Modèle d'autorisation : décrit les communications autorisées, les obligations entre l'organisation et l'environnement, et les communications internes utilisant les diagrammes d'autorisation;
- Modèle de communication : raffine le modèle précédent en décrivant les détails des contrats entre les agents, et en utilisant les réseaux de Pétri. Les transactions entre les agents sont modélisées en utilisant les diagrammes de transaction qui décrivent les rapports entre les actes de discours et les objectifs;
- Modèle de l'univers de discours : concerne la modélisation des contenus des messages échangés entre les agents; cette modélisation utilise les techniques orientées-objet.

Les informations que nous possédons sur cette méthode ne nous ont pas permis d'en faire une analyse détaillée.

6 Conclusion

Dans ce chapitre, nous avons décrit brièvement les méthodologies SMA que nous avons pu recenser dans la littérature. Elles sont classées en deux catégories : soit elles constituent une extension des méthodologies orientées-objet, soit une extension des méthodologies à base de connaissance. Ces méthodologies ont fait l'objet d'une étude approfondie par plusieurs équipes. Il y en a d'autres sur lesquelles nous n'avons pas eu assez d'information.

L'objectif est de mettre la lumière sur l'importance des méthodes de conception, à prendre en considération pour la conception d'une plate forme multi-agent. Ces méthodologies, dans la plupart des cas, n'ont pas été appliquées à des cas pratiques dans l'industrie ou les entreprises.

Conclusions et Perspectives

Conclusions

Nous espérons à travers ce modeste travail, avoir mis l'accent sur, d'une part, les possibilités d'une éventuelle construction d'un paradigme de programmation Agent similaire à celui d'objet, et d'autres part nous espérons aussi avoir mis l'accent sur les innombrables difficultés et obstacles que rencontre la communauté SMA sur l'éclaircissement du formalisme agent et la levé des ambiguïtés qui l'entoure.

Malgré la richesse et la diversité des connaissances du domaine, on a montré qu'il est possible de trouver un palliatif entre les différentes tendances dans ce domaine pour la conception d'une plate plates formes Multi-agents générique; présenter ici par la plate forme NewObject ;

Enfin, et en particulier, l'accent est mis sur l'architecture de l'agent lui même à travers le modèle d'agent présenté.

Pour terminer, nous signalons qu'à l'heure actuelle la technologie des plates formes multi agents est toujours à ces débuts et que seulement quelques plates formes prototype sont à l'essai. Les difficultés dans cette branche de l'intelligence artificielle résultes de la modélisation et l'implémentation des concepts comme la coopération, la négociation, la perception …etc. qui sont des concepts affiliés généralement au domaine du vivant et qui imposent une recherche pluridisciplinaire reliant ainsi l'informatique, la psychologie, la sociologie, l'économie, les mathématiques et toutes filière ayant un trait avec ce domaine.

Perspectives

La grande difficulté qui surgit lors de la conception des plates formes multi-agents réside en majeure partie dans la prise en charge de la concurrence entre agents et la gestion des situations conflictuelles. Nous suggérons l'introduction du concept de la négociation, qui intervient généralement lorsque des agents interagissent pour prendre des décisions communes au moment où ils poursuivent des objectifs différents, dans le but de préserver au maximum la compétitivité des agents et leurs indépendance.

Néanmoins, ce concept reste très difficile à formaliser et de nombreux travaux ont proposés des techniques de négociation ; elles sont soit centrées sur l'environnement soit sur l'agents.

Pour notre cas, nous pensons qu'une négociation sur l'environnement dit orienté tâches reste la plus adaptée. Ici les agents ont un ensemble de tâches à faire et une fois toutes les ressources nécessaires à l'accomplissement de ces tâches sont disponibles chaque agent peut accomplir ses tâches sans interférence, ni besoin d'aide avec les autres agents.

Nous suggérons aussi, le recours à la théorie des actes de langages [annexe] pour améliorer la communication entre agents.

Pour mieux profiter de la hiérarchisation nous suggérons une fois encore, la négociation hiérarchisée. Un agent peut demander à son supérieur hiérarchique de négocie pour lui avec un autre agent ; cette faculté permet à l'agent de se libérer pour exécuter les taches qui lui assignées et optimiser l'exploitation des ressources disponibles.

Bibliographies

[1] http : //www.granddictionnaire.com/_fs_global_01.htm

[2] héron Paul, Guide pratique du génie logiciel. Editions EYROLLES, 1988.

[3] Alliot Jean-Marc, Schiex Thomas, Intelligence Artificielle et Informatique Théorique ; Mars 1994.

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

[5] Chaïb-Draa Brahim, Agents et Systèmes Multiagents (IFT 64881A). Notes de cours. Département d'informatique, Faculté des sciences et de génie, Université Laval, Québec. Novembre 1999.

[6] Ferber, Les Systèmes Multi-Agents, vers une intelligence collective, InterEditions, 1995.

[7] Fox M.S., An organizational view of distributed systems. IEEE Trans. Syst.Man.Univ. Cybern., vol. SMC-11; 1981, pp. 70-80.

[8] Adam E., Kolki C., Étude comparative de méthodes de génie logiciel en vue du Développement de processus administratifs complexes, 1999.

[9] Decker, K., Williamson, M. and Sycara, K., Matchmaking and Brokering. In : Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), December, 1996.

[10] ***Agha,*** *Actor : a Model of Concurrent Computation in Distributed Systems, MIT Press, 1986.*

[11] Patil R.; Fikes R.; Patel-Schneider P.; Mckay D.; Finin T.; Gruber T.; Neches R., The DARPA knowledge sharing effort : Progress report In : Principles of Knowledge Representation and reasoning : Proocedings of the Third International Conference, November 1992. (www.cs.umbc.edu/kqml/papers/kr92.ps)

[12] C.Hewitt, Baker**,** Actors and Continuous Functionals, Technical Report, Computer Science.

[13] Gasser, L. (2000). Multi-agent systems infrastructure definitions, needs, and prospects. In Wagner, T. et Rana, O., editors, Proceedings of the first Workshop on Infrastructure for Scalable Multi-Agent Systems, number A paraitre in Lecture Notes in Computer Science. Springer.

[14] Ginot, V. et Le Page, C. (1998). Mobidyc, a generic multi-agents simulator for modeling populations dynamics. Lecture Notes in Computer Science, 1416 :805–814.

[15]. Epstein, Axtell, Growing Artificial Societies, MIT Press, 1996.

[16] Pitt, J. et Mandani, A. (1999). Some remarks on the semantics of FIPA's agent communication langua.ge. Autonomous Agents and Multi-Agent Systems, 2(4) :333–356.

[17] Steiner, D. (1997). An overview of FIPA 97. Technical report, FIPA Consortium.

[18] Ferber, Gutknecht, Un méta-modèle organisationnel pour l'analyse, la conception et l'exécution de systèmes multi-agents, publication, ICMAS'98.

[19] Sadek, M. (1991). Attitudes mentales et interaction rationnelle : vers une théorie formelle de la comm.unication. PhD thesis, Université de Rennes I.

[20] White, J. E. (1996). Telescript technology : Mobile agents. In Bradshaw, J., editor, Software Agents. AAAI Press/MIT Press, Menlo Park, Cal. Also available as General Magic White Paper.

[21] Cohen, 1994] Coen, M. (1994). Sodabot : a software agent environment and construction

system. Technical report, MIT AI Lab.

[22] Burkhart, R. (1997). Schedules of activity in the swarm simulation system. In OOPSLA'97 Workshop on OO Behavioral Semantics.

[23] Reticular Systems, I. (2000). AgentBuilder, an integrated toolkit for constructing multi-agent systems. Technical Report.

[24] Shoham, Y. (1991). AGENT0 : A simple agent language and its interpreter. In Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), pages 704–709, Anaheim, CA.

[25] OMG (1992). The Common Object Request Broker : Architecture and Specification. Object Management Group, Inc., 492 Old Connecticut Path, Framingham, MA 01701, 1.1edition.

[26] Arnold, K.,Wollrath, A., O'Sullivan, B., Scheifler, R., etWaldo, J. (1999). The Jini specification. Addison-Wesley, Reading, MA, USA.

[27] Labrou, Y. et Finin, T. (1997). Comments on the specification for FIPA '97 agent communication language. UMBC WWW.

[28] Bradner, S. (1996). The internet standards process. Technical Report RFC 2026, IETF.

[29] Nodine, M. (1998). The InfoSleuth agent system. Lecture Notes in Computer Science, 1435 :19.

[30] Chavez, A., Moukas, A., et Maes, P. (1997). Challenger : A multi-agent system for distributed resource allocation. In Johnson, W. L. et Hayes-Roth, B., editors, Proceedings of the First International Conference on Autonomous Agents (Agents'97), pages 32

[31] Guessoum, Z. (1996). Un environnement opérationnel de conception et de réalisation de systèmes multi-agents. PhD thesis, Université Paris 6.

[32] Marcenac, P. et Courdier, R. (1999). A java agent-oriented development environment. In Object-Oriented Application Frameworks.Wiley and Sons Books.

[33] Willmott, S., Faltings, B., Calisti, M., Macho-Gonzales, S., Belakhdar, O., et Torrens, M. (1999). Constraint choice language (CCL), language specification v2.0. Technical Report 99/320, EPFL Departement d'Informatique. 3–331, New York. ACM Press.

[34] Decentralized Artificial Intelligence (2). Yves Demazeau and Jean-Pierre Müller (Eds.), Elsevier Science Publisher B.V. (North-Holland), pp. 3-10, 1991.

[35] Collinot, A. et Drogoul, A. (1998). Using the cassiopeia method To design a soccer robot team. Applied Articial Intelligence (AAI) Journal, 12(2-3) :127–147.

[36] Kitano, H., Asada, M., Kuniyoshi, Y., et Noda, I. (1998). RoboCup : A challenge problem for AI and robotics. Lecture Notes in Computer Science, 1395.

[37] Rumbaugh J. ; Blaha M. ; Premerlani W. ; Eddy F. ; Lorensen W. Object-Oriented Modeling and Design ISBN 0-13-630054-5 Prentice-Hall 1991

[38] Guttag J. Abstract Data Types and the Development of Data Structures. Communications of the ACM, Vol.20, n° 6, June 1977

[39] Minsky M. A Framework for Representing knowledge. in "Psychology of Computer vision", Winston, P (Ed.) Mc-Graw & Hill, New-York 1975

[40] America P. Inheritance and Subtyping in a Parallel Object-Oriented Language in [ECOOP 87] pp. 281-290

[41] Meyer B. Sequential and Concurrent Object-Oriented Programming. in [TOOLS'90] pp. 17-28

[42] Francois B Systèmes Multi-Agents, Cours de DEA IAA Université de CAEN

[43] Scott A. Deloach., Multiagent Systems Engineering : A Methodology And Language for Designing Agent Systems . 1999.

[44] Muller P., Instant UML. Wrox Press, Birminggham, UK, 1997.

[45] Ealmmari M., Lalonde W., An Agent-Oriented Methodology : High-Level and Intermediate Models. Proceedings of AOIS-1999. Heidelberg (Germany), June 1999.

[46] Kinny David, Georgeff Michael, and Rao Anand, A methodology and modelingtechnique for systems of BDI agents. In W. Van der Velde and J. Perram, editors, Agents Breaking Away : Proceedings of the Seventh European Workshop on Modelling Autonmous Agents in a Multi-Agent World MAAMAW'96, (LNAI Volume1038). Springer-Verlag : Heidelberg, Germany,1996.

[47] Moulin Bernard and Brassard Mario, A scenario-based design method and an environment for the development of multiagent systems. In D. Lukose and C. Zhang, editors

[48] Kendall Elisabeth A, Malkoun Malgaret T., and Chong Jiang, A methodology for developing agent based systems for enterprise integration. In D.Luckose and Zhang C., editors, Proceedings of the First Australian Workshop on DAI, Lecture Notes on Artificial Intelligence. Springer-Verlag Heidelberg, Germany, 1996.

[49] Jacobson I., Christerson M., Johnson P., and Overgaard, Object-Oriented Software Engineering. A Use Case Approach. ACM Press, 1992.

[50] FIPS Pub183, Integration definition function modeling (IDEF0). Software Standard Modeling Techniques. FIPS Pub183, Computer Systems Laboratory National Institute of Standards and Technology, Gaithersburg, Md. 20899,1993.

[51] Kosanke K., CIMOSA- A European Development for Enterprise Integration. IOS Press, 1993.

[52] Norbert Glaser, Contribution to Knowledge Modeling in a Multi-Agent Framework (the CoMoMAS Approach). Ph.D thesis, l'Université Henri Poincaré, Nancy I, France, November,1996.

[53] Carlos A.I., Mercedes Garjo, José Gonzàlez C., and Velasco Juan R., Analysis and Design of Multiagent Systems Using MAS-CommonKADS. In AAA'97 Workshop on Agent Theories, Architectures and Languages, Providence, RI, July 1997. ATAL. An extended version of this paper has been published in INTELLIGENT AGENTS IV : Agent Theories, Architectures, and Languages, Springer Verlag,1998.

[54] ITU-T.Z100 (1993), CCITT specification and description language (sdl). Technical report, ITU-T, June 1994.

[55] Ekkart Rudolph, Grabowski Jens, and Graubmann Peter, Tutorial on message sequence chart (MSC). In Proceedings of FORTE/PSTV'96 Conference, October 1996.

[56] Egon M. Verharen, A language-Action Perspective on the Design of Cooperative Information Agents. Ph.D thesis, Katholieke Universieit Brabant, the Netherlands, March1997.

[57] Pascot D., Bernadas C., L'essence des méthodes : Étude comparative de six méthodes de conception de systèmes d'information informatisés. INFORSID'93 «Systèmes d'information, Systèmes à base de connaissances», Lille, 11-14 Mai 1993.

[58] Hogg T., Huberman B.A., Better than the best : The power of cooperation, Lectures in complex systems, Addison Wesley.

[59] Glize Pierre, Gleizes Marie-Pierre et Camps Valérie, Une théorie de l'apprentissage fondée sur l'auto-organisation par coopération. 1998.

[60] AFIA/PRC-13 Systèmes Multi-agents, Architectures de Systèmes d'Agents

[61] Huhns M., Mukhopadhyay U., and Stephens L. M., DAI for document retrieval : The MINDS project. In M. Huhns, editor, Distributed Artificial Intelligence, pages 249-284. Pitman Publishing : London and Morgan Kaufmann : San Mateo, CA, 1987.

[62] Schökle**,** Distribution Simulation Software for Complex Continuous Systems, 1996.

[63] Woods W. Transition network grammar for natural language analysis. Communication of Association of Computing Machinery 13 (10): 591-606,1970.

[64] Searle J. R. (1969). Speech Acts: An Essay in the Philosophy of Language. Cambridge, England.

[65] Austin J. L. (1962). How to Do Things With Words. Oxford, England : Oxford University Press.

[66] Vanderveken D. (1999). Analyse et simulation de conversations : de la théorie des actes de langage aux systèmes multi-agents, chapitre : La structure logique des dialogues intelligents. L'interdisciplinaire informatique. 61-100.

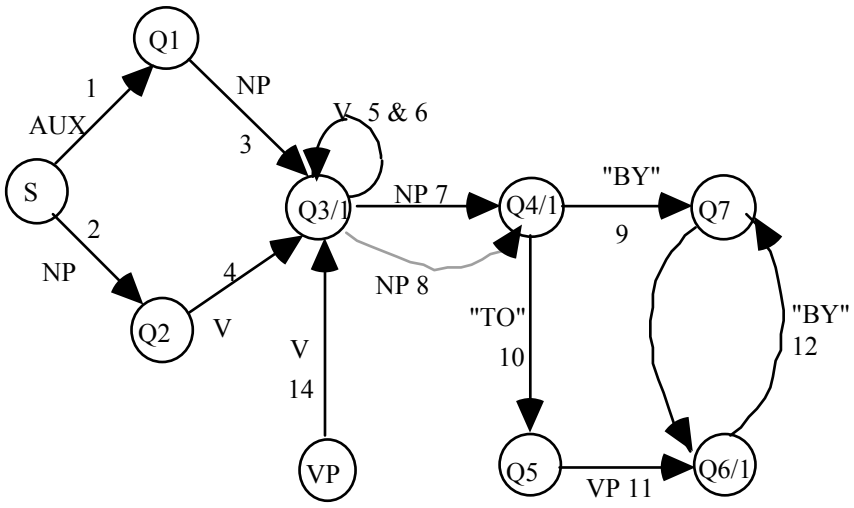
[67] Grosz B. J., Kraus S. (1996). Collaborative plans for complex group action. Artificial Intelligence, Vol. 86, 269-357.

A n n e x e

1 Les Réseaux de Transitions Augmentés (ATN) :

- Les Réseaux de Transition Augmentés sont représentés par un réseau de nœuds et d'arcs les reliant.
    - Les nœuds sont associés à des conditions et à des formes finales représentants les états.
    - Les arcs sont soumis quant à eux à des conditions et à des actions.
- Pour que l'automate passe d'un état à un autre, il faut qu'il satisfasse à un ensemble de conditions ayant la capacité de déclencher des actions.

Donnons tout de suite l'exemple que nous donne Woods [63] de ce formalisme.



Prenons la phrase *Jean aime Marie* et traitons la avec l'ATN précédent :

- L'automate lit le premier mot de la phrase : "Jean". C'est un nom; il emprunte donc l'arc 2 et il arrive en l'état Q2.
- L'arc 2 étant associé à une condition T (vraie), l'ATN exécute l'action correspondante :
    - le nom "Jean" est mis dans un registre SUBJ et "DCL" dans un registre TYPE.
- Rien ne se passe à l'état Q2.
- L'ATN lit le mot suivant. C'est un verbe. L'ATN emprunte l'arc 4 et arrive en l'état Q3.
- L'arc 4 étant associé à la condition T, l'ATN exécute l'action correspondante :
    - le verbe "aimer" est mis dans un registre V et la valeur du présent "PRES" dans un registre TNS du temps.

- L'état Q3 auquel nous arrivons est associé à la condition qui vérifie si le verbe est intransitif. Ce n'est pas le cas.
- L'ATN lit alors le mot suivant "Marie". Celui ci est un nom. cela implique que l'ATN emprunte l'arc 7 pour arriver à l'état Q4.
- L'arc 7 est associé à une condition qui vérifie si le verbe contenu dans le registre V est bien transitif. C'est le cas. Il s'en suit que l'ATN exécute l'action correspondante :
    - mettre dans le registre OBJ le mot "Marie".
- L'état Q4 est associé à la condition T. Il s'en suit l'exécution de l'action de construire la suite suivante :

(S DCL Jean (TNS PRES) (VP (V aimer) Marie))

2 Les Actes de Langages

L'idée de l'acte de langage est d'utiliser ce que l'on comprend du dialogue entre humains [64][65] comme modèle pour la communication entre agents. Dans cette approche l'énonciation, c'est à dire la production d'un énoncé, est un acte qui sert avant tout à produire des effets sur son destinataire. On parle alors d'actes de langages, comme étant l'ensemble des actions intentionnelles effectuées au cours d'une communication. Les principaux actes de langages décrits par Searle dans [64] et Vanderveken dans [66] sont les suivants :

1. ***Les assertifs*** servent à donner une information sur le monde an affirmant quelque chose (ex. il fait beau).
2. Les ***directifs*** sont utilisés pour donner des directives au destinataire (ex. donne moi ta montre).
3. Les ***promissifs*** engagent le locuteur à accomplir certains actes dans l'avenir (ex. je viendrai à la réunion).
4. Les ***expressifs*** servent à donner au destinataire des indications concernant l'état mental du locuteur (ex. je suis heureux).
5. Les ***déclaratifs*** accomplissent un acte par le fait même de prononcer l'énoncé (ex. je déclare la séance ouverte).

Cette classification n'est pas directement utilisable, mais elle permet de considérer certain des types de messages.

Les actes de langage, considérés comme des structures complexes, sont définis par trois composantes :

1. ***La locutoire*** : c'est la composante physique de la communication (envoi d'ondes sonores ou production de caractères) ; production de phrases à l'aide d'une grammaire et d'un lexique donné.

2. L***'illocutoire*** : elle se rapporte à la réalisation de l'acte effectué par le locuteur sur le destinataire de l'énoncé. En pragmatique du langage, un acte illocutoire est caractérisé par une force illocutoire (ex. affirmer, questionner, demander de faire, promettre, prévenir ...) et par un contenu propositionnel, objet de la force illocutoire. Par exemple *"affirmer qu'il pleut"* prend la forme **"Affirmer(il pleut)**".

3. La ***perlocutoire*** : cette composante porte sur les effets que les actes illocutoires peuvent avoir sur l'état du destinataire, ses actions, ses croyances et ses jugements. Par exemple convaincre, inspirer, effrayer, persuader ...

On trouve un autre exemple de ces composantes lorsque Jean dit à Marie, *"S'il te plaît, ferme la fenêtre*".

Dans cet acte de langage Jean émet des sons (composante *locutoir*e) ; il considère ce message soit comme une demande, soit comme un ordre (composante *illocutoir*e) ; enfin si effectivement tout va bien (dans l'intention illocutoire de Jean) Marie ferme la fenêtre (composante *perlocutoir*e). On appelle *performatif* un verbe qui permet d'identifier dans une communication, la force illocutoire correspondante. Les performatifs serviront précisément à définir les différents types d'actes de langage que pourront émettre et interpréter les agents sans ambiguïté. Cela va simplifier énormément la conception des agents.

Un message contenu dans le protocole de communication peut être ambigu, il peut ne pas avoir de réponse correspondante simple, ou nécessite une décomposition et l'assistance d'autres agents. Dans tous les cas le protocole d'échange doit clairement identifier le type du message échangé.

KNOWLEDGE QUERY AND MANI P UL ATI ON LANGUAGE (KQML)

Le principe de KQML est de séparer la sémantique liée au protocole de communication (indépendante du domaine d'application), de celle liée au contenu des messages (dépendante du domaine d'application). Le protocole de communication doit donc être universel (pour tous les types d'agents), concis et constitué d'un nombre restreint de primitives de communication.

Dans KQML un message contient toutes les informations nécessaires à sa compréhension :

```
(KQML.performatif
        :émetteur <texte>
        :récepteur <texte>
        :langage <texte>
        :ontologie <texte>
        :contenu <expression>
        ...)
```

La syntaxe utilisée est celle du Lisp. Les arguments peuvent être donnés dans n'importe quel ordre.

Les performatifs utilisés dans KQML sont indépendants du domaine d'application dans lequel évoluent les agents. En effet la sémantique des messages se trouve définie dans les champs **:contenu** (le message lui même), **:langage** (le langage de programmation avec lequel le message a été écrit) et **:ontologie** (le vocabulaire utilisé dans le message).

Dans KQML les communications peuvent être asynchrones grâce aux champs **:reply.with** côté émetteur et **:in.reply.to** côté récepteur.

Deux agents peuvent "se comprendre" (échanger des messages) s'ils comprennent un même langage et partagent une même ontologie. D'où la nécessité de définir un format de représentation des connaissances manipulées dans les messages ; c'est le cas de KIF que nous verrons dans la section suivante. La définition d'ontologies permettra aux agents de dégager des concepts communs, des attributs communs ou encore des relations communes entres des connaissances (organisation des connaissances). Par exemple, dans l'ontologie d'un *Monde de bloc*s, si le concept de "bloc de bois" d'une certaine dimension est représenté par le prédicat unaire *"Bloc*", alors le fait que le bloc *A* est sur le bloc *B* peut être exprimé en KQML de la façon suivante :

(informer
:émetteur Agent.1
:récepteur Agent.2
:langage KIF
:ontologie Monde de blocs
:contenu (AND (Bloc A) (Bloc B) (Sur A B))

On peut utiliser des langages comme Lisp, Prolog, SQL, Java ...

Les communications entre agents sont soit synchrones, quand l'émetteur attend une réponse, soit asynchrones si l'émetteur continue son raisonnement ou d'agir après avoir envoyé des messages. Dans ce dernier cas, il pourra s'interrompre pour traiter ultérieurement (quand elles arriveront) les réponses éventuelles !

Un message KQML peut contenir lui même un autre message KQML. Par exemple si *Agent.1* ne peut pas communiquer directement avec *Agent.*2, mais il peut le faire avec *Agent.*3, qui lui même peut communiquer avec *Agent.*2, alors *Agent.1* demandera à *Agent.3* de faire suivre un message à *Agent.*2.

:de Agent.1

:vers Agent.2

:émetteur Agent.1

:récepteur Agent.3

:langage KQML

:ontologie kqml.ontologie

:contenu (décrire

:émetteur Agent.1

:récepteur Agent.2

:langage KIF

:ontologie Monde de blocs

:contenu (Sur (Bloc A) (Bloc B))))

Dans ce type de message re-émis la valeur du champ **:de** devient celle du champ **:émetteur** dans le champ **:contenu** du message ; de même la valeur du champ **:vers** dans *"faire suivre"* devient celle du champ **:récepteur** dans le contenu du message.

Les performatives de KQML sont organisées dans sept catégories :

- Questions de base *(evaluate, ask.one, ask.all* ...).
- Questions à réponses multiples *(stream.all* ...).
- Réactions *(reply, sorry* ...).
- Informations *(tell, achieve, untel*l, ...).
- Etats *(standby, ready, next, rest* ...).
- Définition de compétences *(advertise, subscribe, monitor* ...).
- Réseau *(register, unregister, forward, broadcast …)*

Un agent pourra utiliser le performatif *"annonce"* pour dire à un autre agent ce qu'il sait faire (compétences). Exemple : *Agent.2* peut signaler à *Agent.1* qu'il connaît toutes les réponses à la question "Sur *(X,Y)"* qui précise que *X* est sur Y.

(annonce

:émetteur Agent.2

:récepteur Agent.1

:langage KQML

:ontologie kqml.ontologie

:contenu (demander.toutes.reponses

:émetteur Agent.1

:récepteur Agent.2

:en.reponse.a id1

:langage Prolog

:ontologie Monde.de.blocs

:contenu ” Sur (X, Y) ”))

Agent.1 peut alors demander les solutions à *Agent.2* :

(demander.toutes.reponses

: émetteur Agent.1

: récepteur Agent.2

: en.reponse.a id1

: repondre.avec id2

: langage Prolog

: ontologie Monde.de.blocs

: contenu "Sur (X, Y)"))

KNOWLEDGE INTERCHANGE FORMAT (KIF)

L’objectif de KIF est de définir un format standard de représentation des connaissances manipulées par les agents qui soit suffisamment générique pour être utilisé dans différents domaines, mais suffisamment précis pour ne pas conduire à des interprétations différentes.

KIF est un langage logique capable de décrire des faits concrets, des définitions, des abstractions, des règles d’inférence, des contraintes ou encore des meta-connaissances. Il a été conçu pour être utilisé dans le cadre des agents mais aussi celui des systèmes experts ou encore des bases de données.

L’idée la plus forte poursuivie par ses concepteurs est de proposer un langage pivot, sorte de médiateur entre différents langages.

KIF est une version préfixée du calcul formel des prédicats du 1er ordre, étendue pour supporter définitions et raisonnements non monotones. Il inclut une spécification de sa syntaxe et une autre pour sa sémantique. Il peut exprimer très simplement des expressions élémentaires comme par exemple des tuples de bases de données :

```
(Salarié 015.46.3946 departement.1 72000)
(Salarié 026.40.9152 departement.5 67889)
```

Il peut aussi exprimer des expressions plus complexes comme par exemple le fait que la *forme.1* est plus grande que la *forme.2* :

```
( >     ( * (largeur forme.1) (longueur forme.1))
        ( * (largeur forme.2) (longueur forme.2)))
```

On peut inclure des opérateurs logiques dans les formules, comme dans l'assertion suivante qui dit que l'élévation de n'importe quel réel à une puissance paire fournie un nombre positif :

```
( ⇒          ( and       ( nobre-réel     ?x)
                                ( nombre-paire ?n))
             (>          ( resultat   ?x ?n ) 0))
```

KIF peut représenter des meta-connaissances en utilisant les *"back.quote"* (‘) et les virgules. Sur l'exemple suivant *Joe* est intéressé par recevoir le fait que le salaire est composé de trois éléments, plus que par les instances effectives de cette relation :

```
( interest joe ‘ (salary ,?x ,?y ,?z ))
```

Avec KIF on peut également décrire des procédures ou des fonctions :

```
(progn      (fresh.line t)
            (print "Hello!")
            (fresh.line t))
```

Une fonction à *N* arguments est vue comme une relation particulière à *N+1* arguments dans laquelle le dernier argument est le résultat de l'interprétation de la fonction avec les *N* premiers arguments.

Les classes sont représentées par des relations unaires. Par exemple *(C ?q)* signifie que *?q* est une instance de la classe C.

LA NOTION D'ONTOLOGIE

Une ontologie est la spécification d'objets, de concepts et de relations dans un domaine donné *(universe of discours*e). Dans l'exemple de l'ontologie *Monde de bloc*s, *Bloc* représente un concept alors que *Sur* représente une relation.

Les prédicats logiques unaires définissent des concepts ; ceux d'arité supérieure représentent des relations. Formellement, une ontologie est une base pour une théorie logique de la connaissance et de son interprétation.

Par exemple pour représenter le fait qu'un *Bloc* est un objet physique on utilise L'expression suivante :

$$\forall\ x\ (\ Bloc\ x\ ) \Rightarrow (\ ObjetPhysique\ x\ )$$

L'ontologie doit au de la de la hiérarchie de classes, décrire les relations entre les concepts et non pas les instances de classes. Une ontologie est comparable à un schéma de base de données et non au contenu des données de la base.

L'agent doit donc exprimer ses connaissances en utilisant le vocabulaire lié à telle ou telle ontologie. L'ontologie joue alors le rôle d'un dénominateur commun garant d'un dialogue possible entre entités différentes.

On peut par exemple définir une relation entre objets en précisant le domaine d'application de cette relation :

(domaine Sur ObjetPhysique)

Cette restriction permet de limiter l'usage de la relation *Sur* à des instances de la classe *ObjetPhysiqu*e. Par fermeture transitive des sous classes on admettra :

(Sur A B) si *A* et *B* sont des objets physiques.

En revanche on rejettera **(Sur A rêve1**).

Un agent est conforme à une ontologie si ses actions observables de l'extérieur sont en accord avec les définitions décrites par cette ontologie. Ce principe proposé par Newell dans [66] et appelé *Knowledge.Level,* est de décrire les connaissances d'un agent indépendamment des symboles qu'il utilise en interne (cf. les

architectures vues précédemment) pour ces connaissances. Une ontologie garantit la consistance et non pas l'exhaustivité dans l'échange de connaissances (questions . réponses).

Quels sont les critères retenus pour concevoir une ontologie, dont l'objectif est le partage et l'interopérabilité entre des programmes partageant un niveau conceptuel donné ? Une proposition d'étude dans ce sens est proposée dans [67]. Nous en reprenons ici les grandes lignes :

- ***Clarté*** : Les définitions retenues doivent être claires et objectives, pour être les plus indépendants possibles du contexte social ou technique. Il faut essayer d'axiomatiser (logique) autant que possible les définitions, tout en les décrivant en langage naturel ; une définition complète (conditions nécessaires et suffisantes) sera préférée à une définition partielle (conditions nécessaires ou suffisantes).

- ***Cohérence*** : Les inférences produites doivent être cohérentes avec les définitions axiomatisées mais aussi avec les définitions informelles (exprimées en langage naturel).

- ***Extensibilité*** : L'ontologie doit pouvoir supporter des extensions (ajout de nouveaux termes définis à partir d'existants) sans remettre en cause sa cohérence.

- ***Minimiser les biais dus au codage*** : Ne pas dépendre d'un niveau particulier de symboles utilisés dans le codage interne des connaissances. En effet les agents peuvent exprimés leurs connaissances chacun dans des modes de représentation différents.

- ***Engagements ontologiques minimaux*** : Une ontologie requiert des engagements ontologiques (actions en accord avec les définitions partagées) minimaux, de telle façon que la modélisation commune du monde dans lequel évoluent les agents, leur laissent la possibilité de spécialiser cette ontologie. Les engagements ontologiques sont basés sur les termes partagés qui décrivent une vision théorique du monde, la plus ouverte possible, laissant place à toutes les interprétations possibles qui lui sont consistantes.

Certains de ces critères étant ago-antagonistes, leurs emplois conduit à la recherche de compromis. Par exemple une simplification extrême des termes (soucis de clarté) restreindra leurs interprétations possibles.