USTL LIFL Laboratoire d'Informatique Fondamentale de Lille

Numéro d'ordre:

THÈSE

Nouveau Régime

Présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

Pour obtenir le titre de

DOCTEUR EN INFORMATIQUE

par

NOURREDINE BENSAID

Contribution à la réalisation d'un modèle d'architecture multi-agent hiérarchique.

Thèse soutenue le 11 Mai 1999, devant la Commission d'Examen composée de:

Président	:	J-M. Geib	Professeur à l'université de Lille 1
Directeur de thèse	:	P. Mathieu	Professeur à l'université de Lille 1
Rapporteurs	:	J-P. Briot	Chargé de recherche à l'université de Paris 6
		G. Goncalves	Professeur à l'université d'Artois
Examinateurs	:	J-P. Delahaye	Professeur à l'université de Lille 1
		A. Derycke	Professeur à l'université de Lille 1

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.R. d'I.E.E.A. Bât. M3. 59655 Villeneuve d'Ascq CEDEX
Tél. 03.20.43.47.24 Fax. 03.20.43.65.66

Remerciements

Je tiens à exprimer mes vifs remerciements pour **Jean-Marc Geib**, Professeur à l'université de Lille I, qui m'a fait un grand honneur d'avoir accepté de présider ce Jury.

Je remercie, particulièrement, **Philippe Mathieu**, Professeur à l'université de Lille I, de m'avoir accueilli au sein de son équipe et d'avoir dirigé cette thèse. Ses critiques constructives, ses conseils utiles et toutes les discussions fructueuses que nous avons eues, m'ont permis de mener à terme ce travail de recherche, et au delà, ont suscité en moi un grand intérêt pour la recherche. Je tiens à lui exprimer ma profonde gratitude.

Jean-Pierre Briot, chargé de recherche au LIP6, a contribué par ses remarques objectives et utiles, ainsi que les échanges féconds que nous avons eus, à l'aboutissement de ce travail. Je suis à la fois flatté et honnoré qu'il ait accepté d'être rapporteur de cette thèse.

Gilles Goncalves, Professeur à l'université de l'Artois, a également contribué par des discusions enrichissantes et objectives pour mener à terme cette thèse. Je suis flatté et honoré qu'il ait accepté de rapporter ce travail en un temps aussi court.

Jean-Paul Delahaye, Professeur à l'université de Lille I, a accepté de faire partie de mon Jury. Je tiens à le remercier vivement pour l'intérêt qu'il a porté à mon travail en acceptant d'examiner cette thèse.

Je remercie également **Alain Derycke**, Professeur à l'université de Lille I, pour les quelques discussions enrichissantes et constructives que nous avons eues dans les séminaires et autres occasions.

Je remercie aussi tous les membres de l'équipe **SMAC** (Systèmes Multi-Agents et Coopération) pour leur encouragement et l'agréable ambiance qui a régnée durant des années.

Sur un plan strictement personnel, mais non moins important, je remercie mes parents et tous les ami(e)s qui m'ont tant encouragé et soutenu, et qui ont donc contribué chacun à sa manière pour que ce travail puisse être un succès. Sans eux, cette thèse n'aurait jamais pu voir le jour, **je leur dédie ce manuscrit...**

Table des matières

Résumé

Introduction

Aujourd'hui, les besoins de communication et de traitement de l'information des entreprises, de plus en plus croissants, exigent la conception et le développement de systèmes complexes et intelligents. La variété des services requis, la diversité et la répartition physique des sources de connaissances ainsi que les changements dynamiques des besoins des utilisateurs appelle une approche nouvelle capable de contenir toutes ces contraintes. L'intelligence artificielle distribuée (IAD) est une discipline qui offre un riche répertoire d'outils, de techniques et de métaphores pour supporter des systèmes complexes, adaptatifs et ouverts. Elle vise fondamentalement à améliorer la manière de conceptualiser et d'implémenter plusieurs types de logiciels.
Dans cette lignée, les architectures tableau noir représentent un modèle de résolution de problèmes suffisamment fort et puissant pour implémenter les mécanismes de raisonnement et de calculs qui interviennent à l'intérieur d'un agent. L'archétype pour ce modèle d'architecture est le système DVMT [1] [LC83] qui s'est illustré par l'utilisation d'un système tableau noir pour la coordination. DVMT utilise deux tableaux noirs dont l'un est consacré aux données, l'autre pour les buts des agents. Il a été utilisé dans la régulation du trafic automobile. Le système ATOME [2] [LMH87a] est un autre système similaire qui repose sur une architecture de contrôle hiérarchique. Il réalise un bon compromis entre l'expressivité et l'efficacité du contrôle. Ce dernier est décrit sous forme de sources de connaissances sans pour autant perdre en efficacité par rapport à un contrôle procédural pur. ATOME a été utilisé dans plusieurs applications notamment, un système d'aide à la décision pour la gestion des réfections de la voirie de Nancy et un outil de gestion temps réel des contre-mesures pour un pilote d'avion de chasse.
Les systèmes multi-agents (SMAs) représentent un nouveau paradigme pour le développement d'une large variété de services de haut niveau. Dans ce cadre, la technologie multi-agent est utilisée dans un nombre considérable d'applications, allant des simples systèmes tels que le filtrage de messagerie électronique jusqu'aux systèmes ouverts et complexes, caractérisés par des contraintes critiques à l'image, par exemple, du système de contrôle du trafic aérien. En effet, cette approche postule qu'un système est conçu et développé comme un ensemble d'agents interagissant mutuellement en vue de résoudre un problème donné. En outre, elle est plus générale et plus intéressante du point de vue génie-logiciel. Les systèmes multi-agents sont parfaitement adaptés à résoudre des problèmes qui présentent une multiplicité de méthodes de résolution ou une multitude de perspectives et d'entités de résolution de problèmes. En plus des avantages classiques qu'ils procurent dans la résolution de problèmes concurrents

1. Distributed Vehicle Monitoring Testbed
2. Another TOol for building Multi-Expert systems

et distribués, ils présentent des avantages supplémentaires liés aux modèles d'interaction de haut niveau. Les types d'interaction les plus connus sont: (i) la *coopération* qui implique une activité commune d'un ensemble d'agents afin d'atteindre un but commun; (ii) *la coordination* qui suppose l'organisation de l'activité de résolution afin d'éviter les interactions inutiles et d'exploiter celles qui sont bénéfiques; (iii) *la négociation* dont l'objet est d'arriver à un compromis acceptable entre tous les agents engagés dans un conflit.

Si le modèle tableau noir représente une architecture pratique pour la résolution de problèmes, en offrant un cadre explicite pour appréhender l'état global d'un système, l'approche multi-agent est caractérisée en partie, par ses modèles d'interaction puissants et sophistiqués qui la distingue des autres paradigmes et autres approches de résolution.
Notre thèse est centrée autour de l'idée qu'une architecture supportant des interactions de haut niveau et offrant un cadre global pour représenter le comportement et les contraintes globaux du système constitue une contribution intéressante pour résoudre facilement certains problèmes complexes qui ne peuvent être résolus par les approches évoquées dans les paragraphes précédents.

Contribution

L'objectif de cette thèse est de proposer un modèle d'architecture multi-agent intermédiaire aux systèmes d'agents autonomes et aux modèles tableau noir. L'architecture proposée, appelée MAGIQUE[3], vise à supporter des applications complexes mettant en œuvre un grand nombre d'agents cognitifs. MAGIQUE est caractérisée principalement par son architecture de contrôle qui est hiérarchique et dont les multiples niveaux de contrôle sont représentés par des agents particuliers appelés *superviseurs*. Ces derniers jouent un double rôle: coordonner les interactions d'un groupe d'agents du niveau immédiatement inférieur, et servir de cadre de coopération global aux agents du groupe. La connaissance du domaine d'expertise de l'application est représentée au niveau des agents *spécialistes* dont le rôle est de résoudre une tâche qui leur est spécifique. Les agents du système sont dotés de fonctionnalités qui leur permettent de percevoir leur environnement, de raisonner sur leurs croyances, et d'établir un plan d'action dont l'exécution engendre des changements dans l'environnement.

Un aspect important introduit dans notre travail a trait au modèle de coopération et de coordination. Notre architecture autorise deux modes de coopération: la coopération *horizontale* entre des agents spécialistes et la coopération *verticale* entre les spécialistes et les superviseurs. Le protocole de coopération est symétrique dans le cas de la coopération horizontale, et asymétrique dans le cas de la coopération verticale. L'agent superviseur détient une vue globale de son groupe d'agents et coordonne, en conséquence, leurs interactions afin d'imposer un contrôle global sur l'orientation du processus de résolution.
L'architecture est opérationnelle sur un réseau de stations Unix. Elle est implémentée dans un langage objet et multi-threadé fournissant un haut degré de transparence en matière de localisation physique des agents. Le modèle de communication est asynchrone, et est basé sur un modèle de boîte aux lettres. Nous avons utilisé le langage IC-Prolog afin de tirer profit de ses aspects déclaratifs, de sa puissance d'unification ainsi que de sa bibliothèque de threads. Tous les aspects énumérés rendent le langage adapté au prototypage de systèmes multi-agents.

3. Modèle d'Architecture multi-aGent hiérarchIQUE

Des instanciations de notre modèle sont présentées pour illustrer son fonctionnement. Nous proposons notamment un système de gestion des moteurs d'inférence répartis illustrant la coopération d'un ensemble d'agents ayant des capacités d'inférence différentes. À cet effet, les exemples développés renferment des agents spécialistes dont les moteurs d'inférence peuvent fonctionner en chaînage avant, en chaînage arrière, ou même dans différentes logiques.

Nous avons appliqué notre architecture à l'exploration d'un ensemble de robots autonomes explorant un territoire inconnu. L'application s'appuie sur la métaphore de l'exploration d'une planète inconnue par des robots autonomes. Dans notre cas, les robots sont représentés par des agents spécialistes, tandis que les parties explorées (la solution courante) sont rangées dans un tableau noir. Le superviseur se charge de décomposer le travail en tâches qu'il affecte aux robots et coordonne leurs activités (sérialisation des accès au tableau noir, affectation d'une nouvelle tâche à un robot ayant fini l'exploration de sa zone respective, etc...). En plus, le système supporte un mécanisme de régulation de la charge entre les robots. Dès que la charge d'un robot dépasse un certain seuil, le superviseur est informé. Conséquemment, le superviseur identifie le robot le moins chargé et déclenche un processus d'interaction entre le robot chargé et le robot le moins chargé afin d'équilibrer la charge. Un autre aspect de l'application concerne les stratégies d'exploration du territoire. Dans ce cadre, trois groupes de robots, dont chacun est caractérisé par sa propre stratégie d'exploration, coopèrent pour explorer le territoire. Dans notre cas, le territoire est représenté par un labyrinthe dont les dimensions ainsi que le nombre de robots du système sont paramétrables.
Nous envisageons d'utiliser MAGIQUE comme support pour des applications de navigation virtuelle à travers Internet. L'idée est de fournir à un usager distant géographiquement un service de haute qualité afin de pouvoir naviguer dans un magasin virtuel (surface de distribution, musée, visite des vestiges d'une ville, etc...) en fonction de son profil et de ses préférences.

Plan de la thèse

Cette thèse est composée de 6 chapitres et 2 annexes. Les deux annexes apportent un complément d'informations sur l'état de l'art, vu que le domaine des systèmes multi-agents est caractérisé par une littérature dense et variée.

- Le premier chapitre donne une idée globale de la thèse. En effet, il met en évidence la genèse de l'IAD, ainsi que les travaux les plus importants qui ont vu le jour dans ce domaine. Il présente les motivations et les objectifs visés par ce travail. Enfin, il esquisse un bref aperçu de notre contribution.

- Le second chapitre présente les architectures tableau noir. Le principe d'un système tableau noir, ainsi que la problématique du contrôle sont abordés aussi. Dans ce chapitre, est examiné aussi le rôle de chacune des entités du système, notamment les modules, le tableau noir et le mécanisme de contrôle.

- Le troisième chapitre est consacré aux architectures multi-agents. En premier lieu, les architectures BDI[4] (croyances, désirs, intentions) sont passées en revue. Ensuite, Les architectures réactives sont abordées. Leurs fondements, ainsi que leurs principes sont

4. Beliefs, Desires, Intentions

spécifiés. Enfin, les architectures hybrides, dont le but est de concilier les fonctionnalités des architectures BDI et des architectures réactives, sont présentées.

- Le quatrième chapitre présente l'architecture MAGIQUE. Une description détaillée des principes et fondements de notre architecture est donnée. Une comparaison suivie d'une discusion de notre modèle d'architecture avec les autres modèles sont également fournies. Un modèle formel de l'architecture joue le rôle de cadre à une analyse des propriétés des agents de MAGIQUE.

- Le cinquième chapitre est consacré à l'implémentation de notre architecture. Les détails d'implémentation de la plateforme développée, notamment l'aspect objet dans un contexte distribué et l'aspect multi-threadé ainsi que la notion de boîte aux lettres qui permette une communication asynchrone entre des agents distribués sur des machines différentes. Un ensemble d'exemples d'instanciation de MAGIQUE, consacrés à des moteurs d'inférence illustre le fonctionnement de notre modèle.

- Le sixième chapitre se rapporte à l'application de MAGIQUE à la simulation d'un ensemble de robots explorant un territoire inconnu. Il décrit le principe de l'application et aborde l'aspect régulation de la charge supporté par le modèle d'interaction de notre architecture. Un ensemble d'exemples d'exécution sont donnés.

- La thèse se termine par une conclusion générale et quelques perspectives, avec une partie importante consacrée aux travaux futurs. Ces derniers consistent à spécifier une nouvelle application de MAGIQUE à la navigation d'un usager à distance dans une surface virtuelle. L'objectif est de fournir un service de haut niveau permettant à l'usager de naviguer dans un magasin d'articles ou un musée ou une autre surface, en fonction de ses préférences et de son profil.

- L'annexe A dresse un panorama des différents contrôleurs de tableau noir. Chaque type de contrôle est traité à travers un système connu servant d'archétype. Le principe de chaque contrôleur, ses caractéristiques, ses avantages ainsi que ses limites sont également discutés.

- L'annexe B se rapporte à la problématique de l'interaction dans un système multi-agent. Ainsi, une classification des différentes situations d'interaction est donnée. Ensuite, la coopération est abordée en tant que forme d'interaction entre les agents. Les multiples formes de coopération sont présentées. Enfin, l'aspect communication dans un système multi-agent met la lumière sur la communication en tant qu'action intentionnelle et spécifie le rôle des actes de langage pour la communication dans un système multi-agent.

Contexte du travail

Mes travaux de recherche sont effectués dans l'équipe SMAC (Systèmes Multi-Agents et Coopération) du LIFL (Laboratoire d'Informatique Fondamentale de Lille) dirigée par le Professeur Philippe Mathieu.
Les travaux présentés dans cette thèse sont liés au projet LOGIDIS (LOGIque DIStribuée) du programme Ganymède du Contrat de plan État-Région Nord/Pas-de-Calais.

Chapitre 1

Introduction: l'IAD et la résolution de problèmes

Préambule:

Ce chapitre trace un bref historique de l'Intelligence Artificielle Distribuée (IAD) et des travaux les plus importants qui ont marqué le parcours de cette jeune discipline en tant que nouvelle approche pour la résolution de problèmes.

Le modèle tableau noir ainsi que la problématique du contrôle dans ce type d'architectures sont présentés et examinés succinctement. Les systèmes multi-agents, leurs architectures, leurs modèles de coordination, de coopération et de communication sont aussi abordés brièvement dans ce chapitre.

Une nouvelle architecture appelée MAGIQUE est présentée. Ses objectifs, ses fondements ainsi que ses avantages sont passés en revue.

Le chapitre se termine par la description d'une application de l'architecture MAGIQUE au problème de simulation d'un territoire inconnu par un ensemble de robots. Un bref aperçu du travail futur et des perspectives de ce travail sont également donnés.

1.1 Mécanismes universels de résolution

Une nouvelle vision de la résolution de problèmes suppose qu'il existe des mécanismes universels qui peuvent être appliqués à tous les problèmes possibles et en trouver ainsi une solution. GPS [NS61] et SOAR [LNR85] sont les réalisations qui ont été faites dans ce domaine, sans pour autant déboucher sur une quelconque satisfaction car la vision consistant à réduire l'IA à l'application de la psychologie cognitive est réductionniste. Comme conséquence de

cette vision, toute connaissance concernant la résolution de problèmes se ramène à quelques méthodes et représentations.

L'énoncé d'un problème consiste à spécifier des objets initiaux, ainsi que les propriétés d'un objet n'existant pas et qu'il faut construire à partir des objets disponibles. En examinant les objets initiaux, il est possible de déterminer l'action à appliquer sur ces objets. L'application de cette action permet de modifier l'état des objets, voir de construire de nouveaux. Le processus est réitéré jusqu'à ce que l'ensemble des objets en contienne un qui vérifie les propriétés requises dans l'énoncé. Le processus de résolution se présente alors de la manière suivante:

- **Un état initial:** c'est un ensemble d'objets initiaux n'ayant pas les propriétés de l'objet que l'on veut construire. Ces objets sont considérés pour générer de nouveaux plus adéquats.
- **Des actions:** elles sont appliquées sur les objets courants du système. Par conséquent, le choix de l'action à appliquer à un instant donné dépend de la considération de la représentation de l'objet recherché et des objets courants.
- **Un état final:** c'est une représentation de l'objet recherché. Elle stipule la condition pour qu'un objet quelconque puisse convenir. Elle constitue aussi le but de la résolution du problème posé.

En résumé, résoudre un problème revient à trouver une séquence d'opérateurs ou d'actions à appliquer pour atteindre l'état final, à partir de l'état initial.

1.1.1 GPS

GPS - General Problem Solver - est un système destiné à résoudre tous les types de problèmes. Il a été mis au point par Allan Newell, Cliff Shaw et Herbert Simon à la fin des années 50 [NS61][Lau87] [BF81]. Un problème est spécifié par l'objet final à atteindre, l'objet initial disponible et un ensemble d'opérateurs permettant d'effectuer les transformations sur les objets. Ces transformations ont pour but de faire progresser le processus de résolution de proche en proche jusqu'à atteindre l'objet final qui représente la solution recherchée. GPS utilise la méthode *means-ends analysis*. A chaque étape, il mesure la différence entre l'objet courant et l'objet à atteindre; puis choisit l'opérateur approprié afin de réduire au mieux la différence. Pour résoudre un problème donné, on doit fournir à GPS une table d'association reliant aux différents opérateurs les différences qu'ils peuvent réduire. GPS fut le premier programme en IA à dissocier les connaissances inhérentes au processus de résolution - buts, méthodes - des connaissances propres au domaine d'application - opérateurs et table.

La procédure générale:

```
TANT QUE il reste un but non étudié :
    choisir un but B non étudié.
    Si ce but est le but majeur ALORS succès.
    TANT QUE il reste une méthode non appliquée sur ce but :
```

```
        choisir une méthode M.
        SI il n'est pas possible d'appliquer M à B,
           ALORS choisir une autre méthode.
           SINON SI M atteint le but B,
                    ALORS B = père(B)
                    SINON SI M engendre un nouveau but B'
                             ALORS SI B' a déjà été engendré
                                      ALORS l'éliminer.
                          FSI
                 FSI
          FSI
      FSI
      FTQ
FTQ
```

Cette volonté à concevoir un mécanisme général de résolution de problèmes, tout à fait naïve aujourd'hui, reflète l'optimisme des spécialistes de l'IA vers la fin des années 50.

1.1.2 SOAR

Le projet SOAR [LNR85] est une tentative de développer et d'appliquer une théorie unifiée de l'intelligence humaine et artificielle. L'architecture forme une large série d'investigations dans les différents domaines de l'IA tels que: la résolution de problèmes, la planification, l'apprentissage, la représentation de la connaissance, le langage naturel, la perception, la robotique, ainsi que les applications dans les domaines des systèmes experts et de la modélisation psychologique. La communauté des experts travaillant sur le projet est constituée d'informaticiens, de psychologues et de spécialistes en engineering et en sciences sociales, aux US et en Europe. L'idée maîtresse était de combiner l'aspect capacité de résolution de problèmes, flexible et basée sur des espaces de recherches, et des méthodes simples utilisant des connaissances cognitives supportées par des systèmes de production. La première version de SOAR, appelée SOAR1, a été développée en 1982. Son principe reposait sur le développement d'une méthode universelle. SOAR1 est conçu comme un système capable d'exhiber une large variété de stratégies de résolution de problèmes en combinant plusieurs espaces de recherches et des systèmes de production. Les buts, les espaces de recherche, les états et les opérateurs sont représentés au niveau de la *mémoire de travail* du système de production. Le traitement est conduit par un cycle d'application des décisions. Les opérateurs de SOAR sont exécutés une fois qu'ils aient été sélectionnés. La seconde version a vu le jour en 1983. Son objectif principal est d'expliquer comment fonctionne le mécanisme universel de division d'un but en sous-buts (appelé "subgoaling" dans la littérature anglo-saxonne). A chaque fois que le système se trouve dans une "impasse", il invoque le mécanisme qui crée automatiquement une liste de sous-buts pour remédier à l'impasse. Afin de faciliter la détection et le diagnostic des impasses, la procédure de décision, basée sur le vote, est remplacée par une procédure basée sur des préférences symboliques. La version 3 de SOAR complète la version précédente en rendant automatique la génération et la terminaison du mécanisme de subgoaling. La mémoire de travail est étendue pour contenir une pile de contextes de résolution de problèmes. Le déclenchement des règles de production ainsi que le processus de terminaison du subgoaling peuvent être effectués en parallèle. Un mécanisme d'apprentissage est développé. Il

repose sur les résultats de l'expérience de résolution de problèmes basés sur le subgoaling. SOAR3 est développé en 1984. SOAR4 est la première version "publique" et dotée d'une documentation officielle. Cette version avait connu un ensemble d'affinements, particulièrement au niveau du mécanisme d'apprentissage [GRL87][LRN86]. Les différentes analyses de cette version affirment que SOAR4 est une architecture pour une intelligence générale. SOAR5 a été développé pour supporter le principe d'état unique. Il postule qu'un seul état est associé à un espace de recherche. Au niveau du processus de résolution, un certain nombre de changements ont été effectués, notamment l'utilisation de préférences pour toutes les modifications de la mémoire de travail, la distinction entre les éléments persistants et éphémères de la mémoire de travail et le développement d'interfaces d'entrée sortie. SOAR5 est distribué au grand public, avec un manuel complètement nouveau. SOAR6 est la première réimplémentation complète de l'architecture SOAR. Cette version, très semblable à SOAR5, a apporté des améliorations au niveau de l'efficacité, à travers une sélection d'algorithmes plus rigoureuse, ainsi que la robustesse et la maintenabilité du système. Il faut noter que l'efficacité a été la préoccupation principale des experts à cause de son impact sur la taille et la complexité du système. La limitation des ressources (boundedness) est un autre aspect interagissant avec l'aspect efficacité, mais qui est fondamentalement différent. Il consiste à limiter les ressources, tel que le temps, utilisées par l'architecture.
L'efficacité a été l'aspect central sur lequel se sont focalisés les travaux de SOAR, à cause du large impact qu'il a sur la taille et la complexité des systèmes développés à base de cette architecture.

La résolution dans les modèles GPS et SOAR est axée essentiellement sur des méthodes de recherche dans un espace contenant un ensemble d'objets caractérisant le système. Par exemple, GPS est un agent capable de résoudre une classe de problèmes pouvant être représentés par un objet initial, un objet final, des opérateurs pour lesquels il est possible de donner une table de réduction. Bien que les modèles précédents permettent d'exhiber un mécanisme général de résolution d'une classe de problèmes, il existe une série d'autres problèmes plus généraux pour lesquels il n'est pas possible d'être représentés selon le schéma d'une méthode générale comme GPS. Une nouvelle approche de résolution de problèmes a vu le jour. Elle consiste à concevoir des systèmes capables de faire coopérer des agents spécialisés, ayant chacun la capacité de résoudre un type de problème. L'architecture tableau noir représente ce type de modèle de résolution. Elle a pour but de faire coopérer différents agents spécialisés chacun dans un domaine donné. La coopération dans un système tableau noir est opportuniste, i.e., à un instant donné, l'agent le mieux placé pour faire progresser la résolution est choisi pour agir. Les connaissances utilisées par les agents ne sont pas nécessairement homogènes.

1.2 Genèse de l'IAD

Bien qu'elle présente un caractère de nouveauté, l'intelligence artificielle distribuée (IAD) est loin d'être à ses débuts. En effet, les premiers travaux en grande partie américains remontaient aux débuts des années 70. Parmi les travaux influencés par cette approche, nous pouvons citer ceux de Lenat sur les "beings" [Len75] et ceux de Hewitt [Hew77] qui donnèrent naissance aux langages d'acteurs. Cette époque a été marquée aussi par le développement d'une nouvelle architecture de systèmes autour de la notion de *tableau noir*. Initialement créés pour l'analyse de la parole et des signaux à travers le système HEARSAY-II [EHRLR80], l'utilisa-

tion de ces systèmes a été rapidement généralisée à d'autres domaines de l'IA, en particulier pour l'établissement de diagnostics, ou encore la conception et la planification.
Dans la phase qui suivit, trois systèmes ont eu un impact considérable. Le premier est DVMT (Distributed Vehicle Monitoring Test), qui fut développé par l'équipe de V. Lesser à l'université de Massachusetts [LC83]. DVMT est un projet de recherche consistant à percevoir et reconnaître des situations distribuées. Des agents captent des informations qu'ils envoient à d'autres agents de traitement, implémentés sous forme de tableaux noirs. Les agents doivent identifier et suivre des véhicules à partir d'informations redondantes, contradictoires et bruitées, qui proviennent des capteurs. Le système DVMT a permis d'une part, d'examiner le problème de la planification multi-agent à l'aide de plans partiels, et d'autre part de définir les bases des mécanismes de coopération et de négociation entre agents. DVMT a été l'un des systèmes qui ont contribué à la définition de l'IAD en tant que domaine d'étude à part entière.
Un autre système ayant eu un grand impact sur l'ensemble des recherches en IAD est le système MACE développé par L. Gasser [GBh87]. Dans ce projet, il était spécifié la manière de réaliser un système d'IAD, ainsi que les composants inhérents à une plate-forme générique pour le développement de systèmes d'IAD. L. Gasser a relié ses travaux à ceux de Hewitt sur les acteurs pour montrer qu'il est possible de réaliser un système multi-agent à partir de la notion d'envoi de messages et de l'introduction de nouvelles notions telles que les représentations d'autrui, le raisonnement d'un agent sur ses compétences et ses croyances. MACE faisait la distinction entre la compétence effective i.e., "le savoir faire", de la connaissance qu'un agent peut avoir de sa propre compétence. Ainsi, on peut dire que toutes les plate-formes actuelles de développement de SMAs sont des descendants directs ou indirects de MACE.
Vers le début des années 80, R. Smith développa un modèle de répartition dynamique des tâches et de problèmes appelé *réseau contractuel* (contract net) [Smi80]. Ce dernier est un système d'allocation de tâches basé sur un protocole d'appel d'offre de marchés publics. En particulier, il suppose l'existence d'un langage commun pour la description des tâches et des informations à transmettre. Dans contract net, chaque agent est soit un décideur soit un exécutant. Ce modèle a acquis un droit de cité dans le paysage des études en IAD.
Ayant suscité un intérêt grandiose au sein de la communauté des chercheurs en IA, l'IAD repose sur un paradigme qui consiste à considérer un système comme une "société" d'agents autonomes travaillant en commun, selon des modes de coopération, de conflit et de concurrence spécifiques, pour aboutir à un objectif global qui est la résolution d'un problème, l'établissement d'un diagnostic ou encore la construction d'un plan.

1.3 Distribution de l'intelligence

L'essor des systèmes d'IAD, et particulièrement de la notion d'agent est due à de multiples raisons tant pratiques que théoriques. Les motivations ayant présidé à la distribution de l'intelligence sont les suivantes: *la multiplicité des points de vue, le raisonnement distribué et les problèmes physiquement distribués.*

1.3.1 La multiplicité des points de vue

L'approche classique, à travers le développement industriel des systèmes experts , a montré que la conception d'un système d'IA soulève de nombreuses difficultés telles que la taille

importante des bases de connaissances, la multiplicité des sources de connaissances et d'informations, l'incohérence des bases de règles et le raisonnement dépendant de multiples points de vue. Souvent la réalisation d'un projet nécessite la mise en œuvre de plusieurs compétences différentes. Par exemple, concevoir un système de surveillance d'une chaîne de montage nécessite des connaissances diverses: celles de l'ingénieur, mais aussi celles des techniciens, et celles des utilisateurs de l'appareil qui ont fait une projection du comportement du système. Ainsi, les connaissances dont dispose un individu sont nécessairement partielles et limitées par rapport au système dans sa plénitude. Néanmoins, l'agent doit tenir compte des considérations générales avant de prendre une éventuelle décision. Par exemple, l'ingénieur d'étude ne tient compte des problèmes de trésorerie ni de ceux du service juridique. A la fin, le produit réalisé tiendra compte de ces points de vue et de toutes ces décisions partielles.
En résumé, des systèmes nécessitant l'intégration d'informations contradictoires provenant de sources aux intérêts duaux sont au centre des préoccupations de l'IAD. Cette dernière tient compte du fait qu'il n'y a pas de vérité générale, et que toute décision est le fruit d'un compromis entre de multiples points de vue.

1.3.2 Le raisonnement distribué

Une autre motivation qui milite pour le développement des systèmes d'IAD consiste à observer que dans la réalité les hommes travaillent généralement en groupe, mettant en commun des connaissances diverses afin d'atteindre le même but. Ainsi, la gestion d'une entreprise requièrt la compétence de nombreuses personnes dont chacune est spécialisée dans un domaine donné.
Bien qu'il soit plus difficile a priori de représenter la connaissance de plusieurs personnes que celle d'une seule, l'analyse du comportement d'un ensemble de personnes, collaborant dans le cadre d'un but commun, semble être facile si elle s'appuie sur les communications et les échanges qui ont eu lieu entre ces personnes. Cette observation reflète le caractère novateur des système d'IAD par rapport aux systèmes d'IA classique. Chaque entité du système dispose d'une connaissance partielle et son raisonnement est simplifié. Cette simplification du raisonnement au niveau de l'agent est compensée par une complexité des structures de communication et d'interaction entre agents. Finalement, dans un système d'IAD, l'importance du raisonnement de chacun des agents décroît au profit de l'interaction entre agents.

1.3.3 Problèmes physiquement distribués

Généralement, les problèmes complexes sont distribués physiquement. Par exemple, un réseau de transport est réparti sur un espace géographique important, un système de contrôle industriel, similaire à ceux qui sont en œuvre dans des raffineries ou des chaînes de montage, implique une distribution des activités dans l'espace géographique. D'un autre coté, à l'heure des réseaux interplanétaires où toute l'information et la puissance de traitement sont réparties sur un nombre impressionnant de sites, il faut concevoir des systèmes ouverts, comme l'indique C. Hewitt [Hew85], afin de privilégier l'interopérabilité des systèmes informatiques. Dans cette optique, l'idée avancée par M. Tokoro [Tok93] concernant la réalisation d'un "champ calculatoire" (Computing field) vient appuyer le point de vue de Hewitt concernant l'interopérabilité des systèmes. Ainsi, l'espace informatique n'est qu'une gigantesque toile d'araignée dont le *World Wide Web* constitue une première approche. L'ordinateur, qu'il soit fixe, portable ou mobile, est connecté aux autres ordinateurs du monde entier. Les systèmes

d'IAD se présentent comme des candidats sérieux pour le développement d'architectures ouvertes, distribuées, hétérogènes et souples, en mesure d'offrir une grande qualité de service dans un travail collectif.

1.4 L'IAD au carrefour de plusieurs disciplines

L'IAD se trouve au carrefour de plusieurs domaines scientifiques tels que l'intelligence artificielle (IA), les systèmes distribués et la robotique.

1.4.1 L'intelligence artificielle

Alors que les systèmes d'IA classique sont des "penseurs" refermés sur leur propre raisonnement et ignorant leur environnement, les systèmes multi-agents sont des véritables sociétés d'"êtres" qui doivent planifier, communiquer, percevoir, négocier, agir et réagir dans un environnement où parfois ils peuvent rentrer en conflit.
Contrairement à l'IA, les systèmes multi-agents font appel à plusieurs nouveaux concepts comme la *coopération, la coordination d'actions, la négociation, les conflits, l'engagement, la perception, l'action et la réaction.* Autre différence par rapport à l'IA, les nouveaux concepts introduits par l'IAD sont des métaphores qui tirent leur origine dans la sociologie, la biologie et même l'éthologie.
La conception d'êtres intelligents n'est pas un but pour l'IAD car elle s'intéresse essentiellement aux relations entre agents et aux conséquences de leurs interactions. C'est un moyen pour développer des systèmes plus performants par le biais de la réduction de la redondance et de l'augmentation de l'efficacité des actions effectuées.

1.4.2 Les systèmes distribués

Il existe un fort chevauchement entre la problématique soulevée par les systèmes multi-agents et les problèmes inhérents aux systèmes distribués. Ces derniers s'intéressent à la conception de systèmes informatiques en mesure de supporter l'exécution de tâches en gérant efficacement les ressources réparties comme la mémoire, le processeur et autres entités spécialisées dans la gestion de l'information. Les techniques mises en œuvre relèvent plus spécifiquement de la pratique informatique que ne le fait l'IAD. Les systèmes distribués cherchent à concevoir des systèmes d'exploitation répartis dans lesquels l'utilisateur n'a pas à spécifier sur quel processeur son programme va s'exécuter, ni à s'assurer que les ressources nécessaires à son exécution sont présentes sur sa machine. Les problèmes qui se posent dans le domaine des systèmes distribués sont la conception, la spécification et la validation d'applications réparties, en définissant des protocoles permettant à plusieurs modules informatiques répartis d'utiliser mutuellement leurs services.
Par rapport aux systèmes distribués, la problématique des systèmes multi-agents est plus ouverte. Les SMAs s'intéressent au problème de l'interaction entre entités individualisées de manière générale et abstraite. Ils considèrent que les systèmes distribués ne sont qu'une application ou qu'une réalisation possible de ce concept général. Cette approche repose sur des fondements ayant trait aux notions d'autonomie et de coopération (pas au sens partage de tâches ou de ressources informatiques). Ainsi, les notions de "résolution de problèmes", "coordination d'actions" ou encore "partage de connaissances", soulevées dans le domaine des systèmes multi-agents ne correspondent pas à celles soulevées dans les systèmes distribués.

Une comparaison entre un agent et un processus montre que la notion de processus se rapporte à l'exécution d'un morceau de code, alors que l'agent correspond à une individualisation des compétences et des objectifs, à une autonomie des capacités d'action et d'interaction. Néanmoins, les liens qui unissent les deux domaines sont étroits. Les SMAs utilisent un grand nombre de techniques provenant des systèmes distribués, tout en se situant à un niveau plus conceptuel et méthodologique. Ils utilisent des algorithmes tels que l'allocation de tâches, ainsi que certains formalismes de représentation comme les réseaux de Petri, qui sont inhérents aux systèmes distribués. En conséquence, les systèmes distribués et les systèmes multi-agents ont un rapport synergique. Les premiers apportent leur rigueur et leur algorithmes, tandis que les seconds offrent une conceptualisation et une approche plus générale et moins centrée sur les mécanismes d'exécution.

1.4.3 La robotique et la simulation multi-agent

La robotique, et particulièrement la robotique distribuée, est une application possible des systèmes multi-agents. Ces derniers décrivent ce que les robots doivent être capables de faire pour coopérer à l'accomplissement de tâches mettant en œuvre plusieurs robots. En revanche, la robotique s'intéresse à un ensemble de problèmes mécaniques, électroniques, géométriques, ainsi que des commandes de mouvement qui ne font pas partie du domaine d'intérêt des systèmes multi-agents. D'une manière générale, l'application des systèmes multi-agents à la robotique porte essentiellement sur l'interaction d'un ensemble de robots mobiles.
Parmi les travaux les plus connus, on peut citer ceux de Brooks [Bro90] qui se rapportent aux robots ayant pour mission d'explorer la planète Mars. L'idée provenait de la NASA qui decida de financer un projet spatial ayant pour finalité l'exploration de Mars. Il est clair que cette mission ne pourrait se réaliser dans un premier temps que grâce aux robots. Vu la distance qui sépare Mars de la terre, les liaisons entre les deux planètes durent entre 3 et 5 minutes pour un trajet dans un seul sens, ce qui rend l'idée de télécommander des robots à partir de la terre irréaliste. C'est ainsi que les experts de la NASA ont eut l'idée de concevoir et de construire des robots autonomes capables d'évoluer par leur propres moyens dans un univers sans doute hostile. C'est ainsi que Brooks lança l'idée de l'envoie sur Mars d'un grand nombre de petits robots simples dans leur capacité et moins chers. Conséquemment, la coordination des actions des robots nécessite d'être examinée selon une perspective multi-agent. J.L Deneubourg [DAG+86] et L. Steels [Ste90] proposèrent l'idée de réaliser des robots autonomes dont le comportement ressemblerait aux fourmis, et capables de réaliser des tâches efficacement. Les premiers travaux dans ce domaine ont porté d'abord sur la simulation de groupes de robots, avant de passer à la réalisation effective de ces robots réactifs. Il convient de souligner que les travaux qui ont vu le jour dans ce domaine ont été le résultat de collaborations entre d'une part les experts en sciences de l'artificiel, de l'informatique et de la robotique, et d'autre part, les spécialistes en sciences de la vies. Parmi ces collaborations, nous pouvons citer celle entre L. Steels et D. McFarland, éthologue de renom [McF90]. Il y a eu aussi une collaboration entre A. Drougoul et les biologistes spécialisés dans le comportement animal [DFCF92].
Les chercheurs qui se sont investis dans ce domaine se sont intéressés au problème d'extraction d'échantillons de minerai. Ainsi, le problème se décrit comme suit: plusieurs robots mobiles partent à partir d'une base fixe (sinon très peu mobile) afin d'explorer un espace supposé inconnu pour trouver, récupérer et transporter le minerai découvert jusqu'à la base de départ. Le problème consiste à trouver et identifier la structure et le comportement appropriés des robots ainsi que l'organisation du groupe afin d'obtenir des résultats intéressants en terme de

temps mis par les robots pour extraire le plus de minerai possible. Ainsi, pour résoudre un problème de ce type, il faut passer par un certain nombre d'étapes:

- Définir les différents types de robots à construire: cela revient à décider si ces robots sont identiques ou au contraire, ils sont spécialisés? L'idée de construire des robots spécialisés dans le transport de minerai ayant des capacités de forage faibles, mais qui peuvent transporter des charges plus importantes que les robots généralistes (des robots qui savent tout faire!).
- Définir les capacités cognitives des agents: sont-ils en mesure de construire des cartes à partir de l'itinéraire qu'ils ont parcouru? sont-ils réactifs? i.e., capables uniquement de perceptions locales ou peuvent-ils communiquer entre eux par messages?
- Déterminer les groupes de travail: les robots travaillent-ils en groupe ou de manière individuelle? les groupes sont-il fixes ou dynamiques? est-il possible de rajouter d'autres robots? comment?
- Déterminer les protocoles de coopération et d'interaction entre les robots afin qu'ils accomplissent collectivement leur tâches respectives à l'abri de nombreux conflits qui risquent de surgir.

L'application d'exploration du labyrinthe que nous avons proposée dans le chapitre 6 présente une métaphore similaire à celle décrite dans cette section. L'architecture MAGIQUE (voir chapitre 4) pourrait être utilisée comme cadre de coopération entre des robots spécialisés. Ainsi, les problèmes de conflit et de coordination entre les robots se trouvent pris en charge au niveau des agents superviseurs.

Parmi les autres travaux de simulation dans le domaine des systèmes multi-agents, nous pouvons citer le simulateur SIMDELTA, utilisé pour synthétiser les connaissances d'un ensemble de spécialistes (halieutes, écologistes, biologistes, anthropologues, etc...) ayant travaillé pendant plusieurs années sur l'étude du système de pêche du delta central du Niger au Mali. Le simulateur [Cam94] permet aussi de simuler la dynamique de population des poissons, en tenant compte des nombreux facteurs biologiques et topologiques qui peuvent affecter son évolution.
Deux séries d'expérimentation ont été réalisées. La première a porté sur l'étude de la dynamique de population des poissons en fonction d'un effort de pêche de plus en plus grand. Elle a permis de reproduire la courbe en trois phases (la reproduction, la croissance, la migration et la mortalité) caractérisant l'évolution d'un système de pêche limité et surtout de faire apparaître des variations portant sur la composition spécifique et la taille des poissons.
La seconde a pour but la modélisation des pêcheurs qui prennent des décisions et agissent sur la réserve renouvelable [BCM93].
Tous ces travaux illustrent bien que les systèmes multi-agents sont très bien adaptés à la simulation d'environnements réels.

1.5 Architecture tableau noir

L'architecture tableau noir a été motivée par les besoins suivants rencontrés dans des problèmes concrets.

Intégration de connaissances hétérogènes: ce point pose le problème de la représentation de la connaissance qui est central à la résolution de problèmes. Il permet d'exprimer les états de l'espace de recherche, ainsi que les opérateurs permettant de se déplacer dans l'espace d'états. Par exemple, le problème du traitement des langues nécessite une grande masse de connaissances qui sont souvent hétérogènes. Afin de comprendre des phrases ou textes écrits en langage naturel, il faut spécifier des connaissances morphologiques sur la formation des mots, des connaissances syntaxiques, des connaissances sémantiques et des connaissances pragmatiques. Bien que ces connaissances soient distinctes et correspondent à des niveaux d'analyse différents, leur coopération est nécessaire afin de construire un système de compréhension.

Utilisation de formalismes différents: en faisant référence à l'exemple de la compréhension du langage naturel, chaque type de connaissance manipule un formalisme différent. Par exemple, les grammaires manipulent des arbres syntaxiques, les connaissances sémantiques manipulent des frames, tandis que les connaissances pragmatiques utilisent des plans.

Coopération des connaissances: les connaissances de même nature sont regroupées en modules et coopèrent en s'échangeant des informations. Ces connaissances sont manipulées par la partie contrôleur du système. L'inconvénient réside au niveau du fait que chaque module a besoin des connaissances des autres modules pour conduire son fonctionnement.
L'intégration réunit dans de mêmes modules des connaissances de niveaux différents. Bien qu'elle évite les ambiguïtés artificielles dues à la division en modules et assure une bonne efficacité, elle n'est pas recommandée dans les domaines où une mise au point expérimentale de l'utilisation des connaissances et de leur interaction est requise. Un exemple d'analyseur ayant intégré les connaissances syntaxiques et sémantiques est décrit dans [SR83]. Néanmoins, parmi les travaux effectués dans le domaine de l'analyse des langues, aucune théorie linguistique à ce jour n'a réussi à intégrer l'ensemble des connaissances dans un seul module. Afin de prendre en compte tous les aspects du langage, une approche coopérative est recommandée: [Sab90b].

1.5.1 Le modèle du tableau noir

L'architecture de tableau noir est un modèle général de résolution de problèmes où les connaissances sont regroupées en modules coopérant entre eux (voir Fig.1.1). Les modules ont les caractéristiques suivantes:

- Les modules renferment des connaissances homogènes permettant de coopérer à la résolution du problème. Ces connaissances reposent sur les mêmes formalismes caractérisant un ensemble d'états. Ces derniers constituent un espace de recherche du module.

- Les modules possèdent un même espace de recherche: le tableau. Ce dernier est caractérisé par un ensemble d'états qui sont des objets accessibles à tous les modules. Un module reconnaît un type d'objets qui le concerne. Il connaît la structure interne ainsi que le formalisme de ses objets. Le tableau contient des objets dont la structure n'est pas spécifiée a priori.

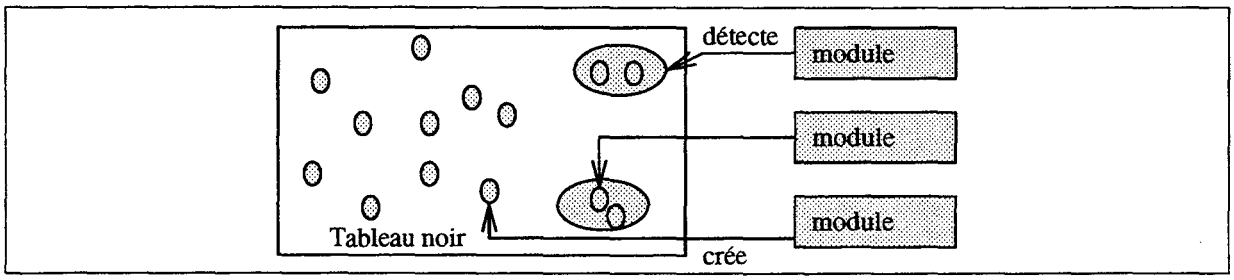

FIG. 1.1 - *Principes des systèmes Tableaux noirs*

- La seule activité des modules consiste à lire, écrire et modifier des objets dans le tableau. Les modules ont une structure de type condition/action. Quand une configuration est reconnue, la partie action est exécutée. Son exécution créera des objets dans le tableau.
- Chaque modification du tableau, engendrée par un module, induit un nouveau état du tableau qui correspond à une solution partielle. Le processus de résolution évolue, ainsi incrémentalement, d'une solution partielle à une autre plus affinée.

1.5.2 Contrôle des tableaux noirs

Dès qu'une configuration d'objets satisfait la partie condition du module, il y a création d'objets dans le tableau noir. Le module ignore l'activité des autres modules, ainsi que les autres objets du tableau.
Le modèle du tableau noir est bien illustré par la métaphore d'une classe d'élèves. Le but est de faire coopérer les élèves à poser un puzzle sur le tableau. Chaque élève possède une ou plusieurs pièces du puzzle. Il ne voit devant lui que le tableau. Initialement, une pièce du bord est posée. Chaque élève remarquant que la configuration des pièces posées permet de poser la sienne, se lève et va poser sa pièce sur le puzzle. Quand, toutes les pièces sont posées, le puzzle est terminé. Le puzzle est ainsi construit incrémentalement par l'ensemble des élèves. En faisant le parallèle avec le modèle du tableau noir, les élèves sont des sources de connaissances, les pièces représentent les objets, la pièce initiale est l'objet initiale du tableau noir.
Le modèle du tableau noir est défini par les éléments suivants:

- Les connaissances sont structurées en modules indépendants qui "s'ignorent" mutuellement.
- La structure tableau noir contient la solution courante du problème. Chaque nouvelle configuration du tableau est une solution partielle du problème.
- Un module se déclenche dès qu'une configuration du tableau lui permet d'exécuter une action.
- La solution est construite d'une manière incrémentale.

Contrôle séquentiel: les modules d'un système tableau noir ont une structure de la forme condition/action similaire aux règles des systèmes experts. Quand la partie condition d'un

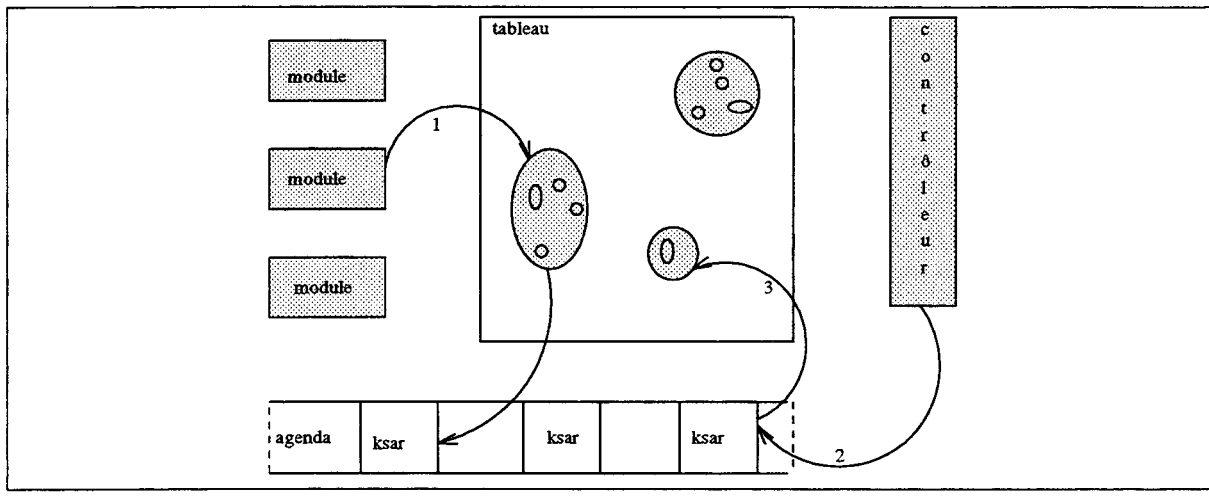

FIG. 1.2 - *Cycle de base d'un système tableau noir*

module est satisfaite, il y a création d'un **KSAR** (acronyme de Knowledge Source Activation Record). Un KSAR est un couple (module, liste d'objets). L'exécution d'un KSAR est une possibilité d'évolution du système. Elle implique l'application du module sur la liste d'objets correspondants. Le déclenchement d'un KSAR provoque la création d'un ou de plusieurs événements. Un événement peut être représenté par une structure de données de contrôle enregistrant les changements opérés dans le tableau noir (cas du système ATOME [LMH88b]. Un événement résulte de l'exécution d'une action. Une action consiste à (i) créer de nouveaux objets, (ii) à en supprimer, (iii) ou encore à modifier des objets du tableau. Le déclenchement d'un KSAR permet donc de constituer de nouvelles configurations d'objets rendant déclenchables certains modules du système. Le cycle de base du système est le suivant.

1. Les modules créent des KSARs qu'ils stockent dans une structure de données appelée *agenda.*

2. Le contrôleur du système choisit un KSAR dans l'agenda.

3. Déclenchement du KSAR choisi, engendrant l'exécution de l'action correspondante qui se traduit par la création, la suppression ou la modification d'objets.

La phase 2 du cycle pose un problème de choix du KSAR à déclencher, dans le cas où plusieurs KSARs sont créés par les modules (voir Fig.1.2). Le système ne peut déclencher qu'un seul KSAR à la fois. Ce point soulève la problématique du contrôle dans les architectures tableaux noirs.
Les multiples implémentations du modèle tableau noir proposent un contrôle particulier de telle sorte, qu'à un instant donné, soit choisi le meilleur KSAR en fonction de l'état courant du tableau. Parmi les instanciations les plus connues, on peut citer le système HEARSAY-II [EHRLR88], utilisé pour la reconnaissance de la parole. Le modèle a été annoncé en même temps que son application à la reconnaissance de la parole. D'autres systèmes tableaux noirs ont vu le jour comme CRYSALIS [Ter88], HASP/SIAP [NFAR88] qui se sont ajoutées aux applications existantes. Le reste des travaux dans le domaine des systèmes tableaux noirs s'est

focalisé sur les architectures de contrôle. D'autres systèmes sont conçus comme des environnements de développement d'applications tableau noir, comme ATOME [LMH87b] et GBB [CGM86].
En résumé, nous pourrons dire que le tableau noir est un modèle de résolution de problèmes où la connaissance est structurée en modules indépendants, communiquant par le biais d'un mécanisme de lecture/écriture d'objets sur le tableau. Bien qu'il est bien adapté pour résoudre des problèmes qui nécessitent d'appréhender d'une manière explicite l'état global du système, beaucoup d'autres problèmes exigent une approche parfaitement distribuée où les agents doivent avoir des fonctionnalités complexes. Nous reviendrons plus en détails sur les propriétés et les fondements des architectures tableau noir dans le chapitre 2 et l'annexe A.

Notre contribution consiste principalement à proposer une nouvelle architecture, appelée MAGIQUE, capable de supporter des systèmes tableau noir et adaptée pour résoudre des problèmes où les agents doivent avoir des capacités élaborées.

1.6 De l'IAD vers les systèmes multi-agents

Après l'apparition des systèmes décrits dans les sections précédentes, les travaux ont connu une évolution pour se focaliser sur la notion de systèmes multi-agents (SMAs). Cette dernière notion postule qu'un système peut être pensé en terme d'un ensemble d'entités appelées *agents* qui coopèrent pour atteindre un but commun: la solution au problème. Les activités de recherche dans le domaine des SMAs se déploient dans toutes les directions, au point où il est difficile de faire un panorama exhaustif des tendances actuelles. Parmi les courants de pensée qui se sont distingués, il y a lieu de citer des travaux menés aux USA et centrés essentiellement sur les systèmes multi-agents cognitifs autour de Victor Lesser [LC83] et de ses disciples Ed Durfee [DLC87] et Susan Conry [CML88]. Il y a aussi d'autres groupes animés par des chercheurs comme L. Gasser [Gas91], M. Huhns [Huh87] et K. Sycara [Syc89].
Des travaux sur la formalisation des agents rationnels autonomes ont été initiés par Cohen et Levesque qui ont proposé un modèle formel des notions d'intention et de croyances à partir de logiques modales [CL90][CP79]. Ce courant est représenté par Y. Shoham [Sho93] et Georgeff [RG92b]. Ces derniers ont ajouté à leur démarche théorique une dimension pratique qui les a conduit à réaliser des systèmes qui implémentent leurs théories. En Europe, ce courant est relayé par J. Galliers [Gal90], E. Werner [Wer89], C. Castelfranchi et R. Conte [CMC91], [WJ94] et au Québec, par Chaib-Draa [CD90].
Un autre groupe de chercheurs considère qu'il existe des liens étroits entre les actes de langage et la communication dans les systèmes multi-agents. À l'instar des travaux menés par [MC94] [Bra92], une tendance très nette consiste à définir effectivement des protocoles de communication basés sur la théorie des actes de langage. L'exemple le plus illustratif est l'ensemble des travaux portant sur le standard KQML [1] [FFMM94].
Une autre tendance analyse les SMAs à partir d'un point de vue de la théorie des jeux. Elle est représentée par J. Rosenschein [ZR92]. Une synthèse de ces travaux se trouve dans [RZ94].
Une école essentiellement française a utilisé les réseaux de pétri, en tant qu'outils théoriques des systèmes distribués, dans l'analyse, la conception et la validation des SMAs. Elle comprend les équipes de P. Estraillier [BE92], J. Ferber [FM94], S. Pinson et S. Haddad [FSH94].

1. Knowledge Query Manipulation Language

D'autres travaux sur les langages d'acteur et les SMAs sont menés particulièrement au Japon par M. Tokoro [MIT90], T. Ishida [Ish90] et A. Yonezawa [Yon90]. C. Carle et J. Ferber [Car92] ont tenté de concilier les recherches sur le parallélisme en général et les langages d'acteurs en particulier, avec les concepts et les objectifs des SMAs.
Les agents réactifs ont constitué un domaine d'intérêt pour des chercheurs comme R. Brooks. Ce dernier est relayé par la suite par J.-L. Deneubourg, J. Ferber et A. Drougoul, Y. Demazeau, P.Bourgine, J.A. Meyer et L. Steels. Cette approche des SMAs fait partie de la mouvance "vie artificielle". Une partie des aspects décrits dans cette section sont explicités dans les chapitres B, 3.

1.7 Définition d'un agent

Dans [Fer89], J. Feber définit la notion d'agent comme suit:

"On appelle agent une entité réelle ou abstraite qui est capable d'agir sur elle même et son environnement, qui dispose d'une représentation partielle de cet environnement, qui, dans un univers multi-agent, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions avec les autres agents".
Une autre définition du même auteur permet d'intégrer dans la notion d'agent de nouvelles propriétés comme: *les ressources, les tendances et la reproduction.* Ainsi dans [Fer95], la définition donnée est la suivante:

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

Malgré la confusion qui règne autour de cette notion, les experts ont retenu un certain nombre de postulats pour caractériser un agent. Le premier postulat est que les agents sont dotés d'*autonomie.* Cette dernière signifie que les agents ne sont pas dirigés par des commandes en provenance de l'utilisateur, ou d'un autre agent. Néanmoins, l'agent est guidé par un ensemble de buts à satisfaire. Il a la possibilité d'accepter de répondre à une requête en la traitant, ou de la refuser. Cette propriété permet de distinguer l'agent de la notion d'"objet", de la notion de "module logiciel", ou encore de la notion de "processus". L'autonomie porte aussi sur les ressources. Pour agir, l'agent a besoin de ressources comme: le CPU, l'espace mémoire, l'accès à certaines sources d'information. Ces ressources sont gérées par l'agent

lui même. Par conséquent, l'agent est dépendant de son environnement. L'autre principe, sur lequel repose la notion d'agent, est la *capacité d'agir* de l'agent. L'action est une notion cruciale pour les systèmes multi-agents. Les agents exécutent des actions dont l'effet est la modification de l'environnement des agents. La modification de l'environnement influe sur la prise de décision future des agents. Ces derniers ont aussi la possibilité de *communiquer* entre eux. La communication est un des modes principaux d'interaction entre les agents.
Les agents ont une *représentation partielle* de leur environnement. Chaque agent n'a qu'une vue partielle correspondant à son domaine de compétence.

1.7.1 Agents cognitifs et agents réactifs

Il est tout à fait légitime de se poser la question suivante: faut-il concevoir les agents comme des entités "intelligentes", capables de résoudre des problèmes par eux-mêmes, ou bien faut-il les assimiler à des entités simples réagissant directement aux variations de l'environnement?
La réponse à cette question a donné naissance à deux écoles de pensée. La première est l'école "cognitive" qui est très représentée dans le domaine de l'IAD. Elle trouve son origine dans la volonté de faire coopérer des systèmes experts classiques. Dans cette optique, un SMA contient un nombre réduit d'agents disposant chacun d'une base de connaissances comprenant l'ensemble des informations nécessaires à l'accomplissement de sa tâche et à la gestion des interactions avec les autres agents, ainsi qu'avec son environnement. Les agents sont "intentionnels" car ils disposent de plans explicites leur permettant d'atteindre leur buts. Les problèmes de coopération entre agents ressemblent à ceux de petits groupes d'individus, qui doivent coordonner leur activité et négocier pour résoudre d'éventuels conflits. Beaucoup de chercheurs dans ce domaine font appel à des métaphores de sociologie et en particulier la sociologie des organisations pour résoudre des problèmes.
L'autre tendance est l'école "réactive". Cette école postule que les agents ont un fonctionnement très simple, de type stimulus/action. Un système contient un grand nombre d'agents réactifs dépourvus d'intelligence. L'intelligence du système émerge à partir de l'interaction de ce grand nombre d'agents réactifs.

1.8 Définition d'un SMA

À l'instar de la notion d'agent, une définition formelle de cette notion est définie dans [Fer95] par J. Ferber.

"On appelle système multi-agent (ou SMA), un système composé des éléments suivants:
1. Un environnement E, c'est à dire un espace disposant généralement d'une métrique.
2. Un ensemble d'objets O. Ces objets sont situés, c'est à dire que pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est à dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
3. Un ensemble A d'agents, qui sont des objets particuliers ($A \subseteq O$), lesquels représentent les entités actives du système.
4. Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.
5. Un ensemble d'opérateurs Op permettant aux agents de A de percevoir, de produire, de consommer, de transformer et de manipuler des objets de O.
6. Des opérateurs chargés de représenter l'application de ces opérations et la réaction du

monde à cette tentative de modification, que l'on appellera les lois de l'univers".

Pour qu'un système soit considéré comme un système multi-agent, il est nécessaire de satisfaire certains critères:

1. Disposer d'un ensemble d'agents autonomes fonctionnant en parallèle et cherchant à satisfaire un but. L'autonomie signifie que l'agent opère sans qu'il y ait une intervention directe d'un humain ou des autres agents, et a le contrôle de ses actions et de son état interne.
2. Les agents sont dotés d'un mécanisme d'interaction de haut niveau indépendant du problème à résoudre. Les agent peuvent utiliser un *langage de communication agent* appelé, dans la littérature anglo-saxone, *Agent-Communication Language* (ACL).
3. Un agent perçoit son environnement qui peut être le monde physique, un utilisateur à travers une interface graphique, une collection d'autres agents, le réseau Internet, ou encore une combinaison de tous ces éléments. L'agent doit répondre dans un délai acceptable.

En s'appuyant sur les critères précédents, nous pouvons affirmer qu'une décomposition modulaire d'un programme ne constitue pas un système multi-agent. Les raisons fondamentales se résument au fait que les modules n'ont pas de but à atteindre, moins encore une fonction de satisfaction comme c'est le cas pour les agents. Les mécanismes d'interaction entre les modules sont de bas niveau. Ils se réduisent à des appels de procédure ou des envois de message pour demander un service que les modules sont obligés de rendre, sinon ils produisent une erreur. Pour des mêmes raisons que celles évoquées précédemment, un programme d'IA qui s'exécute en parallèle n'est pas un système multi-agent. A travers ces comparaisons, la question qui est souvent posée est la suivante:

Quelle est la différence entre un objet et un agent?

1.8.1 Agent ou Objet?

Dans le domaine de l'informatique, les notions d'objet et d'acteur sont vues comme des entités informatiques caractérisées par leur structure et leur mécanisme d'exécution. Le paradigme de la programmation orientée objet sous-entend trois concepts. La relation classe/instance, décrivant la classe comme un modèle structural et comportemental, tandis que l'instance comme représentant de cette classe. Le second concept est l'héritage qui permet de dériver une classe d'une autre et de faire doter la première des caractéristiques de la seconde. Enfin, l'envoi de message qui autorise la définition de procédures polymorphes dont le code diffère en fonction du receveur de message.
Vu sous un angle implémentation, les objets se situent à un niveau modèle d'exécution avec les langages à objets. Parcontre, ils se situent à un niveau logique, dans le cadre de l'analyse et de la conception par objet. Par conséquent, les objets ne sont pas des agents car ils ne satisfont pas les critères définis précédemment. D'un côté, ils n'ont ni but ni une fonction de satisfaction; d'un autre côté, le mécanisme d'envoi de message se réduit à un appel de procédure.

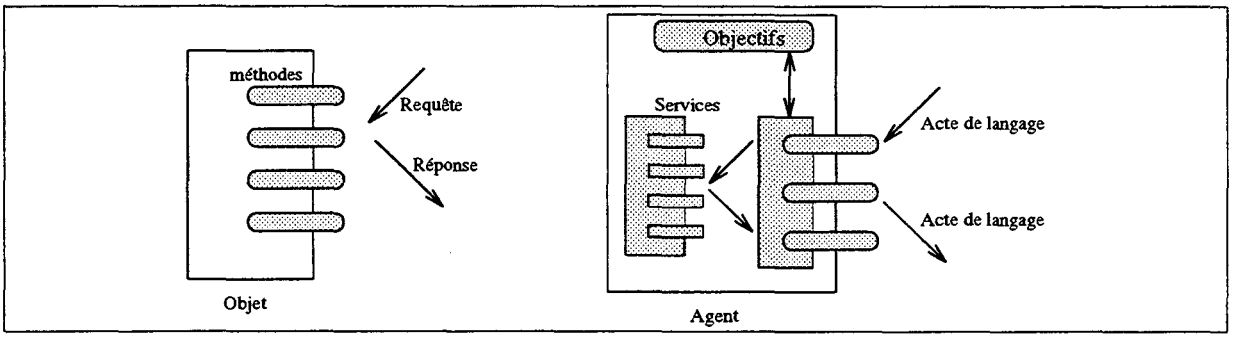

FIG. 1.3 - *Différence entre un objet et un agent*

Les *acteurs* sont des entités qui s'exécutent en parallèle. La communication entre acteurs est basée sur l'envoi de message asynchrone et bufferisé. Généralement, un acteur n'attend pas l'envoi d'un résultat de calcul mais demande au receveur du message de renvoyer la réponse à un autre acteur appelé la continuation locale du calcul [Agh86].
Un objet est défini par un certain nombre de services, ce sont ses *méthodes* qu'il ne peut refuser d'exécuter si un autre objet lui envoie un message. Dans l'étape de développement d'un logiciel à base d'objets, le programmeur doit s'assurer que tous les objets reçoivent des messages sensés qu'ils seront effectivement en mesure d'exécuter.

En revanche, les *agents* peuvent recevoir des messages qui ne sont pas seulement des demandes d'exécution mais aussi des informations ou des requêtes d'information sur leurs capacités (voir Fig.1.3). En outre, les agents tentent de satisfaire des objectifs, ce qui leur procure une autonomie supplémentaire par rapport aux objets. Contrairement à l'objet, un agent peut refuser d'accepter l'accomplissement d'une tâche donnée à cause de son manque de compétence (il ne possède pas le savoir-faire requis), ou encore à cause de sa trop grande "occupation". Un objet encapsule les méthodes qu'un agent peut offrir sous la forme d'un ensemble de services qui ne sont accessibles que par le biais d'un langage connu par l'ensemble des agents. Ce langage utilise, généralement, la théorie des actes de langage. Néanmoins, le lien existant entre la notion d'objet et celle d'agent est loin d'être sous-estimé. Un objet peut être considéré comme un agent dont le langage d'expression se résume à l'emploi de mots clés correspondant à ses méthodes. Un autre aspect qui rajoute un peu plus à la confusion entre la notion d'objet et celle d'agent, est que souvent un agent est implémenté, pour des raisons pratiques, sous la forme d'objets ou d'acteurs.
Pour résumer, on peut dire que les agents sont définis à un niveau conceptuel tandis que les objets et les acteurs sont définis à un niveau d'implémentation et d'exécution. L'approche multi-agent n'offre point un cadre de représentation de la solution globale ni des contraintes globales du système. Les problèmes d'interaction et les différentes architectures des systèmes multi-agents sont examinés dans le chapitre 3 et l'annexe B.

Le problème qui consiste à appréhender le comportement global du système est une des motivations de notre contribution.

1.9 MAGIQUE: une architecture multi-agent hiérarchique

Alors que les agents d'un système tableau noir sont réduits à de simples sources de connaissances n'ayant aucune capacité de perception ni de raisonnement sur le comportement des autres agents du système, une bonne partie des problèmes nécessite une approche parfaitement distribuée, représentée au travers des systèmes multi-agents. Dans cette approche, les agents sont dotés de toutes les capacités leur permettant de percevoir leur environnement de raisonner sur leur comportement ainsi que celui de leur accointances et de coopérer mutuellement afin d'atteindre leur buts respectifs. Cependant, les systèmes multi-agents ne permettent guère d'appréhender le comportement global du système car la connaissance du système ainsi que l'activité de contrôle se trouvent totalement distribuées au niveau des agents du système. Dans ce cadre, MAGIQUE[2] a pour objectif la conception de systèmes complexes moyennant un contrôle hiérarchique efficace. L'architecture de contrôle est représentée par des niveaux de contrôle constitués d'agents *superviseurs* dont le but est de coordonner les activités de groupes d'agents. La connaissance du système est répartie entre des agents *spécialistes* dont chacun a pour rôle de résoudre une tâche qui lui est spécifique.

1.9.1 Le spécialiste

Les spécialistes sont des agents cognitifs capables de structurer un comportement consistant à percevoir leur environnement, à raisonner sur les croyances perçues et à agir sur l'environnement en exécutant un plan d'actions dépendant de leur état interne. Ils renferment de la connaissance qui dépend du domaine d'expertise de l'application. En outre, les spécialistes ont un comportement adaptatif leur permettant de pouvoir s'adapter à une situation non prévue. La structure interne d'un spécialiste est constituée des attributs et fonctions suivants: une base de croyances, un ensemble de bases de règles de croyances, un ensemble de bases de règles de coopération, un filtre de sélection des bases de règles de croyances, un ensemble de capacités, une fonction de maintien de vérité, et une fonction de calcul des priorités à attribuer aux capacités déclenchables.
La multiplicité des bases de règles des croyances a pour but de susciter un comportement qui puisse s'adapter à plusieurs situations disjointes susceptibles de survenir dans le temps. La base des croyances est une mémoire locale de l'agent. Son rôle consiste à maintenir la connaissance représentant l'environnement. Les capacités représentent ce que le spécialiste est capable de faire, tandis que la base de coopération filtre et identifie la connaissance de contrôle afin d'informer les accointances de l'agent, qui sont intéressées par cette connaissance.

1.9.2 Le superviseur

À l'instar du spécialiste, le superviseur adopte un comportement qui repose sur une activité de perception de l'environnement, de raisonnement sur les connaissances perçues et d'action sur l'environnement. Sa tâche consiste à collecter les croyances transmises par les autres agents, à générer de nouvelles connaissances à partir de celles déjà existantes et à exécuter un plan d'actions. Ce dernier consiste à exécuter des actions privées au superviseur ou à informer les agents du groupe qu'une croyance est satisfaite, ou encore à sélectionner une capacité au sein des agents du groupe. L'action d'un superviseur est appelée *méta-action*,

2. Modèle d'Architecture multi-aGent hiérarchIQUE

tandis que sa capacité est appelée *méta-capacité*. Ainsi, le rôle principal est de coordonner les interactions entre les agents de son groupe. La structure interne du superviseur est semblable à celle du spécialiste. Elle est caractérisée par les attributs et les fonctions suivants: une base de croyances, un ensemble de bases de règles de croyances, un filtre de sélection des bases de règles de croyances, un ensemble de capacités, une fonction de maintien de vérité, et une fonction de calcul des priorités. La base des croyances joue le rôle de tableau noir car tous les agents du groupe envoient la connaissance, relevant du domaine d'intérêt du groupe, au superviseur. Ce dernier mène une activité de raisonnement sur cette connaissance au travers de l'application de sa base de règles de croyances et génère de nouvelles croyances. La multiplicité des bases de règles de croyances a pour rôle d'adapter le processus de résolution en fonction des changements survenus dans l'environnement des agents.

1.9.3 Contrôle dans MAGIQUE

L'architecture de contrôle de MAGIQUE est fondamentalement hiérarchique. Elle repose sur plusieurs agents superviseurs appartenant à des niveaux d'abstraction hiérarchiques. Chaque superviseur coordonne les interactions d'un groupe d'agents du niveau immédiatement inférieur. Néanmoins, les agents spécialistes sont cognitifs, par conséquent, ils renferment une dimension de contrôle. Cette dernière repose sur une entité de l'agent spécialiste qui décide des actions à exécuter à un moment donné de l'évolution du système. Cette entité de contrôle exécute un cycle d'exécution de l'agent qui consiste à percevoir l'environnement du spécialiste, à raisonner sur ses croyances pour générer de nouvelles croyances, puis à sélectionner un ensemble de capacités à exécuter. En ce qui concerne l'agent superviseur, toute son activité est "dédiée" au contrôle car son rôle principale est de coordonner les interactions entre les agents de son groupe et de fournir un cadre de coopération partagé pour ces agents. Le superviseur a la capacité, en fonction de son état interne, de sélectionner certaines capacités au niveau des agents de son groupe. Les capacités sélectionnées deviendront "candidates" pour être exécutées dès qu'elles soient satisfaites. Son cycle d'exécution est similaire à celui du spécialiste. Il consiste à percevoir des croyances en provenances des agents de son groupe, à raisonner sur ces croyances et à exécuter un ensemble de méta-capacités. Le contrôle hiérarchique procure au système une meilleure efficacité. Les détails sur la structure des agents, leur cycle d'exécution ainsi que leur modes d'interaction sont donnés dans les chapitres ultérieurs (cf. chapitre 4).

1.9.4 Coopération dans MAGIQUE

L'architecture hiérarchique de MAGIQUE met en œuvre un nouveau modèle de coopération entre les agents. Ce modèle s'articule autour d'un protocole de communication constitué d'un ensemble de primitives de communication qui trouvent leur fondement dans les actes de langage. Ces primitives appelées aussi actes communicatives sont à l'origine des échanges d'informations entre agents coopératifs. On distingue deux types de coopération: la coopération "horizontale" et la coopération "verticale". La coopération horizontale a lieu entre les agents spécialistes. Le protocole est symétrique et s'appuie sur deux primitives appelées: *requête* et *informer*. Ces primitives sont similaires à celles proposées par Shoham dans Agent0 [Sho93]. La primitive *informer* permet à un spécialiste d'informer ces accointances qu'une croyance donnée est satisfaite. Cette opération engendre un changement de l'état mental de l'agent destinataire. La primitive *requête* a pour rôle de solliciter un service décrit par un

ensemble de propriétés fournies sous forme de paramètres de la requête soumise. Vu que le comportement d'un spécialiste vis à vis de ses accointances est autonome, la requête soumise risque d'être déléguée à un autre agent plus compétent, comme elle peut être retardée pour faute de ressources, ou encore elle peut être traitée dans l'immédiat.
À l'inverse de la coopération horizontale, la coopération verticale n'est pas symétrique. Le protocole de coopération contient deux actes communicatifs *informer* et *sélectionner*. La première primitive est identique à l'acte communicatif évoqué dans le cas de la coopération horizontale. Par contre, la primitive *sélectionner* est exécutée exclusivement par l'agent superviseur. Par conséquent, elle consacre la semi-autonomie des agents car elle autorise un agent de sélectionner une capacité au niveau d'un autre agent.
Ainsi, MAGIQUE "rompt" avec les approches de coopération classiques axées essentiellement sur des interactions directes entre des agents autonomes. En introduisant des entités de contrôle explicites, le modèle de coopération est le résultat de deux approches jadis antinomiques, en l'occurrence le modèle du tableau noir dont la coopération entre modules est matérialisée via un tableau, et les systèmes d'agents autonomes dont la coopération s'appuie sur un ensemble d'actes communicatifs.

1.10 Implémentation

Pour implémenter MAGIQUE, nous avons développé une plateforme en vue d'avoir un environnement de programmation capable de faciliter le prototypage de notre architecture. Nous avons utilisé IC-Prolog pour réaliser notre plateforme grâce à ses capacités d'inférence et de son aspect déclaratif jouant un rôle important dans le prototypage du langage dans un temps "record". En plus, IC-Prolog est doté d'un mécanisme de communication basé sur la notion de boîte aux lettres. ce mécanisme garantit une communication asynchrone entre des agents répartis sur un réseau de stations. Le langage développé offre un cadre de programmation distribué permettant à des objets "logés" sur des machines distinctes de communiquer mutuellement, et offre aussi tous les avantages du langage Prolog ainsi que les aspects objets nécessaires pour développer des applications multi-agents. Des informations supplémentaires sur les détails d'implémentation telles que la gestion de l'héritage, l'instanciation des objets ainsi qu'un ensemble d'exemples d'instanciation sont données dans le chapitre 5. Ainsi, chaque agent est caractérisé par un thread qui gère la boîte aux lettres. Le modèle est aussi caractérisé par ses méthodes de communication récursives qui permettent de diffuser un message aux agents spécialistes (feuilles de la hiérarchie) seulement, ou vers les agents du groupe ou encore destiné vers tous les agents de la hiérarchie. Une application est implémentée sous forme d'un ensemble d'agents qui coopèrent mutuellement. Les capacités, les méta-capacités et les bases de règles de croyances des agents sont spécifiées au moment de l'implémentation car elles dépendent du domaine de l'application. Aussi, un ensemble d'exemples d'instanciation illustrant le fonctionnement du système sont présentés. En effet, un moteur d'inférence réparti permet de faire coopérer un ensemble de moteurs d'inférence ayant des modes d'inférence différents. Les moteurs d'inférence peuvent fonctionner en chaînage avant, en chaînage arrière, ou encore en logique floue.

1.10.1 Apports de MAGIQUE

L'intérêt fondamental de MAGIQUE est sa capacité à supporter aussi bien les systèmes tableau noir que les systèmes d'agents autonomes. Cet aspect confère à l'architecture une très

bonne généricité. D'un autre côté, l'organisation de l'architecture de contrôle en niveaux de contrôle hiérarchiques a pour but de supporter des applications complexes mettant en œuvre un grand nombre d'agents regroupés en groupes.
En outre, l'architecture fournit un cadre de représentation des contraintes globales au travers des superviseurs et facilite l'appréhension de l'état global du système. Ainsi, la connaissance de contrôle se trouve explicitement représentée au niveau des superviseurs.
En résumé, MAGIQUE sert de cadre pour le développement d'applications complexes telles que celles qui relèvent du domaine de la robotique ou encore des applications physiquement distribuées sur Internet.

1.11 Simulation de l'exploration d'un territoire inconnu

L'une des applications ciblées par MAGIQUE a trait à la simulation d'un environnement multi-robot. Le problème repose sur l'idée de simuler l'exploration d'un territoire inconnu par un ensemble de robots qui coopèrent mutuellement pour atteindre leur but respectif. L'idée résulte de la métaphore qui consiste à larguer à partir d'un engin aérien un ensemble de robots sur une planète non explorée. Les robots entament l'exploration à partir des différents points du territoire. Chacun des robots a pour tâche de poursuivre une direction jusqu'à ce que le robot rencontre un obstacle. Dans ce cas, on considère que le robot a trouvé un chemin, par conséquent, il informe ses accointances que ce chemin est exploré. Si le robot a d'autres alternatives à explorer, il réitère le processus précédent. Dans le cas contraire, le robot déclenche un processus de coopération avec ses accointances afin de trouver un autre point à partir duquel il va commencer l'exploration.
Un agent superviseur a une vision globale du territoire à explorer. A chaque fois qu'un robot a fini l'exploration de sa zone, le superviseur lui indique une autre zone non explorée. La coopération entre robots intervient quand un robot a fini l'exploration de sa zone et qu'il n'y a plus de nouvelles zones à explorer. MAGIQUE a été utilisé comme cadre pour supporter cette application car tous les robots du système partagent le même espace d'exploration et les solutions obtenues sont stockées dans un tableau noir visible pour tous les robots. Le système est caractérisé par un mécanisme de régulation de charge entre les robots. Quand un robot a beaucoup de travail, alors qu'un autre a une charge très faible, le mécanisme de régulation de charge est déclenché afin d'équilibrer la charge entre les robots. Pour des raisons d'efficacité, nous avons utilisé PM^2 comme environnement de programmation. L'application est opérationnelle sur un réseau de stations Unix.
Un autre aspect de la robotique ciblé par notre architecture concerne le problème de la planification de la trajectoire d'un robot dans un environnement dynamique et inconnu. Le robot est un spécialiste ayant une vision locale de son environnement. Il est appelé à se déplacer dans un environnement inconnu. L'absence d'une représentation globale de l'environnement rend difficile voir aléatoire le déplacement du robot. MAGIQUE apporte une solution à ce problème en proposant une représentation globale de l'environnement au niveau des superviseurs. Ces derniers coopèrent entre eux afin d'établir un plan de déplacement du robot.

1.12 Vers des applications de navigation sur Internet

La tendance actuelle en terme d'applications multi-agents s'oriente vers les applications physiquement distribuées sur Internet. Dans cette optique, des travaux sont entrepris afin

d'utiliser MAGIQUE comme architecture pour supporter une application de navigation d'un client distant géographiquement dans une "surface" virtuelle. Cette dernière peut être un magasin de distribution de biens, ou un musée ou encore l'ensemble des sites archéologiques et des vestiges historiques d'une ville donnée.
Le système a pour but d'extraire les préférences d'un usager distant géographiquement, de les analyser et de générer un profil conforme à ses préférences. En fonction du profil de l'usager et de ses préférences, le système génère un plan de navigation dont l'exécution va permettre à l'usager d'avoir une vue du magasin adaptée à son profil. Le système contient trois agents spécialistes et un superviseur. Les agents spécialistes sont: un agent *interface* pour extraire les préférences du client et identifier son profil, un agent *planificateur* dont le rôle est de générer un plan de navigation adapté au profil du client, et enfin, un agent *navigateur* qui a pour but d'exécuter le plan de navigation. Le rôle du superviseur est de coordonner les interactions des trois spécialistes.

1.13 Résumé

L'approche multi-agent constitue un cadre conceptuel et méthodologique pour la résolution de problèmes complexes nécessitant l'accès à des ressources distribuées et caractérisés par des contraintes de réactivité liées à une forte dynamicité de l'environnement. Beaucoup de travaux intéressants, évoqués dans les sections précédentes ont contribué sérieusement à ce que la technologie multi-agent soit devenue une réalité aujourd'hui. Notre contribution consiste à proposer une architecture visant à intégrer deux modèles, jadis antinomiques, en l'occurrence le modèle tableau noir et les systèmes d'agents autonomes. L'intérêt est de tirer profit des avantages des deux modèles pour concevoir une architecture "synthèse". L'architecture proposée a été utilisée pour simuler l'exploration d'un territoire inconnu par un ensemble de robots autonomes. Un autre aspect de la robotique concerne la planification d'une trajectoire d'un robot en dehors de son environnement. En plus, MAGIQUE est appelé à supporter des applications client/serveur physiquement distribuées.
Néanmoins, l'architecture proposée ne prend pas en compte les aspects temps réel car le modèle ne fixe aucune contrainte sur le cycle d'exécution de l'agent. Une extension de ce travail aura pour but de prendre en considération cet aspect en vue d'une prise en compte les aspects réactifs afin de pouvoir supporter des systèmes nécessitant une forte réactivité du système.

Chapitre 2

Architecture tableau noir

Préambule:

Ce chapitre aborde la problématique du contrôle dans les systèmes tableau noir. Le problème de contrôle est examiné sous trois aspects liés respectivement à la nature de la connaissance de contrôle, sa représentation au sein de l'entité de contrôle et son utilisation par cette dernière.

Les modules du système, le tableau noir ainsi que le mécanisme de contrôle sont décrits et les interactions entre ces entités sont spécifiées.

D'un autre côté, le comportement du système est décrit au travers de la nature des objets du tableau, des relations existantes entre eux, de leur structure ainsi que des solutions partielles "construites" jusqu'à la génération d'une solution satisfaisante.

Un cycle fonctionnel du système illustre parfaitement le comportement du système via les multiples entités qui interagissent mutuellement afin de trouver la solution à un problème donné.

2.1 Le contrôle dans les tableaux noirs

Concevoir et développer un système tableau noir revient à réaliser une architecture de contrôle. La réalisation de cette architecture nécessite de résoudre le problème suivant:

Comment choisir de manière opportuniste le prochain KSAR à déclencher afin de construire de manière incrémentale la solution au problème ?

L'architecture de contrôle a pour but de spécifier la structure des modules du système ainsi que celle de l'entité de contrôle. Ce dernier composant s'appuie sur des connaissances qui lui permettent de choisir les KSARs en fonction des modules déclenchables et de l'état du tableau. Le problème du contrôle se ramène ainsi, à spécifier le contrôleur et les connaissances de contrôle. Dans un système tableau noir, un module regroupe des connaissances du domaine qui sont homogènes entre elles. Pour un module, le problème du contrôle du système est supposé résolu. Les modules s'ignorent mutuellement. Ils communiquent indirectement via le tableau noir. Ils reconnaissent des configurations d'objets dans le tableau, leur permettant d'y apporter les changements sans se soucier des activités des autres modules. Toutes les connaissances se rapportant à l'interaction et à d'éventuelles communications existantes entre les modules ne sont pas définies. Néanmoins, dans la plupart des applications, de telles connaissances existent. Elles sont exploitées par l'entité de contrôle afin d'effectuer le meilleur choix en fonction de l'état du système et des conséquences de ses choix. Les connaissances de contrôle portent sur les modalités d'interaction des modules.

Le problème du contrôle revient finalement à proposer une structure de contrôle, et de lui spécifier les connaissances de contrôle. La première dimension de l'entité de contrôle consiste à intégrer les connaissances disponibles sur le comportement des modules et à identifier leur nature. La seconde dimension consiste à faire un modèle de représentation de ses connaissances afin que l'entité de contrôle puisse remplir sa tâche.
L'entité de contrôle utilise le modèle de représentation fourni pour contrôler les activités des modules. Le caractère de connaissance de contrôle réside non pas au niveau des représentations de ces connaissances, mais au niveau de l'utilisation que l'entité de contrôle en fait pour contrôler ses activités. L'utilisation des connaissances de contrôle constitue la troisième dimension pertinente. Le problème de contrôle s'articule autour de trois aspects:

- La nature des connaissances de contrôle.
- La représentation des connaissances au niveau de l'entité de contrôle.
- L'utilisation des connaissances de contrôle par l'entité de contrôle.

Finalement, nous pourrons affirmer que la nature des connaissances de contrôle spécifie ce qui doit être pris en compte par l'entité de contrôle; la représentation de ces connaissances spécifie ce qui est explicite et leur utilisation précise la signification qu'elles possèdent pour l'entité de contrôle.

Nature des connaissances de contrôle: la problématique du contrôle consiste à savoir, à un moment donné du processus de résolution, d'une part, quel est le module à activer et d'autre part quelles sont les données ou les objets du tableau qui sont concernés par l'activité du module choisi?
Ce problème implique deux propriétés fondamentales caractérisant tout contrôle de tableau noir:

Incrémentalité: L'entité de contrôle construit progressivement la solution en activant les modules, à partir d'une configuration d'objets initiaux pour obtenir des configurations terminales.

Opportunisme : l'entité de contrôle choisit à chaque étape du cycle du système le prochain KSAR à déclencher en fonction de l'état courant du système. Le processus de contrôle ne s'appuie pas sur une stratégie de contrôle préétablie.
Les connaissances de contrôle permettent de reconnaître quel est le prochain KSAR à déclencher. Cette propriété résulte du caractère incrémental du fonctionnement d'un système tableau noir. En plus, le choix du KSAR dépend de l'état courant du tableau et non d'une stratégie préétablie - c'est la caractéristique d'opportunisme.
Les connaissances de contrôle contiennent donc les informations nécessaires concernant les objets et leur signification. Elles contiennent aussi des informations sur la nature des solutions recherchées. Elles dépendent de la nature des objets et de leur signification. Ainsi, le contrôleur doit avoir une idée des conséquences induites par les choix qu'il a effectués, sinon la notion d'opportunisme signifiera hasard. Les connaissances de contrôle portent sur les contributions éventuelles des modules et de leur interaction. Enfin, l'entité de contrôle doit avoir des stratégies d'utilisation des connaissances de contrôle.

État courant du système: l'état courant du tableau est constitué d'un ensemble de configurations d'objets. L'entité de contrôle possède des connaissances lui permettant de reconnaître si des solutions ont été construites, ainsi que le travail qui restait à faire pour les construire.

Comportement des modules: les connaissances de contrôle représentent les interactions des modules, ainsi que leur capacité à coopérer à partir de l'état courant pour atteindre les buts recherchés.

La définition d'une architecture de contrôle nécessite la spécification des trois aspects concernant les connaissances de contrôle:

- Spécification des connaissances de contrôle concernant les objets du tableau noir. Cette spécification revient à préciser les objets et les solutions recherchées.
- Spécification des connaissances de contrôle liées au comportement des modules.
- Précision des stratégies d'utilisation de ces connaissances.

Les connaissances de contrôle peuvent être spécifiées en terme d'objets ou d'actions. Les connaissances décrivant l'état du processus de résolution expriment l'état courant du système, i.e., les objets et les solutions partielles obtenues, les buts poursuivis et les multiples choix du moment. Ce sont plutôt des objets décrivant une situation. Les connaissances décrivant le comportement des modules et leur interaction précisent les lois d'évolution du système en cas de déclenchement des modules. Ce sont donc des actions. Ces dernières ont la structure suivante :

Condition : ce champs exprime la condition à satisfaire pour que l'action du module soit exécutée. L'objectif d'opportunisme exige de l'entité de contrôle de reconnaître les circonstances présidant à l'exécution d'une action.

Action : ce champs décrit le contenu de la connaissance.

Ces connaissances correspondent à des règles de production. Elles sont représentées explicitement dans les systèmes comme ATOME [LMH87a], BB1 [HR85], ou implicitement dans les systèmes comme HEARSAY-II [EHRLR80].

2.2 Le tableau noir, les objets et les solutions

La solution du processus de résolution est représentée par les objets présents dans le tableau à un moment donné de la résolution. Les connaissances concernant l'état courant du tableau et son état final (solution recherchée) permettent à l'entité de contrôle de mesurer l'état de progression du processus de résolution.

2.2.1 Le tableau noir

C'est une structure de données renfermant des informations partagées par des modules hétérogènes du système. Ces informations sont des objets dont la nature reflète celle des modules. Les objets de même nature sont rassemblés en niveaux hiérarchiques. L'ensemble des niveaux est généralement ordonné de telle sorte à former un ordre d'abstraction hiérarchique entre ces niveaux. Chaque niveau hiérarchique s'appuie sur des interprétations des données d'un niveau d'abstraction inférieur.

2.2.2 Les objets

Les objets représentent le contenu du tableau noir. Initialement, le tableau contient des objets constituant l'énoncé du problème. Une fois le processus de résolution enclenché, les changements opérés par les modules dans le tableau le font progresser incrémentalement vers une solution. A un instant donné, les objets du tableau constituent une solution partielle. Les objets du tableau possèdent les propriétés suivantes :

- L'ensemble des objets du tableau représente l'état courant du processus de résolution. En outre, les objets constituent les moyens de communication entre les modules.
- Les objets sont distribués à travers un ensemble de niveaux hiérarchiques du tableau. Les informations, inhérentes aux objets - i.e., leurs attributs ou leurs propriétés - appartenant à un niveau donné servent de données à un ensemble de modules. Ces derniers réagissent en fonction de ces informations pour écrire de nouvelles informations sur le même niveau ou sur d'autres.
- La distinction entre les données structurées et reliées entre elles, des traitements effectués sur ces informations, donne lieu à des objets structurés et entretenant des relations entre eux.

Cependant, la caractérisation précédente n'est pas suffisante pour appréhender l'état du tableau noir du point de vue de l'entité de contrôle. À cet effet, il est judicieux de trouver des

réponses aux questions suivantes:

- Quelle est la signification d'un objet du tableau noir?
- Quelle est la structure d'un objet?
- Que peuvent être les relations entre les objets du tableau?

Signification des objets: jusqu'à présent, nous avons vu que les objets du tableau sont des solutions partielles, ou des interprétations de certaines données selon un certain formalisme inhérent à un niveau d'abstraction. Nonobstant, l'interprétation des données sur un niveau d'abstraction pourrait conduire à des incohérences. Ainsi, plusieurs objets du tableau peuvent donner des descriptions différentes de la même donnée à un niveau d'abstraction donné. Ces objets sont des hypothèses concurrentes pour représenter ces données dans un niveau d'abstraction.
En conclusion, nous pouvons dire qu'un objet est une représentation de données initiales à un niveau d'abstraction donné. Ainsi, l'objet est vu comme une proposition pour interpréter des données selon un certain point de vue. Les objets peuvent être définis comme des hypothèses concurrentes représentant les mêmes données à un même niveau d'abstraction.

Structure d'un objet: un objet est un ensemble d'attributs supportant une ou plusieurs valeurs. Dans le cas du modèle tableau noir, l'entité de contrôle possède un accès explicite aux attributs des objets du tableau [Nii86a]. Les relations entre les objets sont exprimées par des attributs particuliers.

Relations entre objets: la définition d'un objet du tableau noir comme une hypothèse suppose la prise en compte de l'aspect cohérence induit par cette notion d'hypothèse. Certaines hypothèses engendrant des ambiguïtés peuvent coopérer ou au contraire être incompatibles. A noter qu'aucune représentation des relations de cohérence entre objets n'est prévue dans le formalisme général des objets du tableau.
Dans le système HEARSAY-II, le problème de cohérence est résolu en considérant que les hypothèses représentant les mêmes données sont compétitives, par contre celles représentant des données différentes sont coopératives. L'inconvénient majeur est que cette solution est procédurale et ne permet pas donc de déterminer des critères de cohérence pour des solutions partielles déduites. L'entité de contrôle n'accéde pas explicitement aux connaissances lui permettant de gérer la consistance. Généralement, les relations entre objets ne sont pas formalisées. Résultat, la redondance, la contradiction, l'incompatibilité ne sont pas définies clairement car elles dépendent largement des connaissances du domaine.

2.2.3 les solutions

Au début du processus de résolution, le problème posé est représenté par une configuration d'objets initiaux. Une solution est définie comme une configuration particulière du tableau. Une solution peut être un objet ou un ensemble d'objets particuliers. Les modules ou l'entité de contrôle doivent être en mesure de reconnaître ces configurations particulières. Une solution peut aussi être le tableau dans son ensemble. Dans ce dernier cas, la solution est soit la configuration d'objets, obtenue après activation de tous les modules, soit un état particulier

que l'entité de contrôle a détecté. Dans le système HEARSAY-II, les niveaux d'abstraction comprennent un niveau inférieur : les données, et un niveau supérieur constitué d'objets solutions. En revanche, dans CRYSALIS, les solutions sont des objets de niveau le moins abstrait. Il s'agit d'interpréter des informations sur la structure secondaire d'une protéine[1] pour en déduire sa structure tertiaire.
Nous concluons que la spécification des solutions recherchées consiste à définir les objets du tableau, leur structure ainsi que leur signification.

2.3 Les modules

Dans un système tableau noir, un module contient un ensemble de connaissances homogènes entre elles, représentées dans un même formalisme. La communication entre modules se fait indirectement par le biais du partage des mêmes informations dans le tableau. En outre, un module n'a pas de modèle sur le comportement des autres modules, ni sur l'entité de contrôle. Toutes les connaissances portant sur les interactions entre les modules sont représentées au niveau de l'entité de contrôle. Ainsi, les modules sont considérés comme de véritables "sourds muets". L'intégration des connaissances de contrôle au sein de l'entité de contrôle a pour but de lui permettre de choisir de façon opportuniste le prochain KSAR à exécuter.

2.3.1 Le choix des KSARs

Les KSARs représentent les contributions possibles des modules à un certain moment du processus de résolution. Les modules du système réagissent dès qu'ils auraient reconnu une configuration d'objets qui les concerne. La réaction du module consiste à créer spontanément un KSAR. Le problème de l'entité de contrôle se résume à faire un choix entre les multiples KSARs créés par les modules. Il est similaire à la problématique rencontrée dans les systèmes de production lors de la résolution de conflits. Les conflits que soulèvent les KSARs peuvent avoir les caractéristiques suivantes:

- Les modules ayant créé des KSARs sont concernés par les mêmes objets du tableau. Le problème se ramène alors à savoir si les traitements associés aux KSARs sont compétitifs ou coopératifs, s'il existe un meilleur ordre pour leur exécution, etc...

- Certains KSARs contiennent des objets compétitifs, i.e., des hypothèses concurrentes. Dans cette situation, l'ordre d'exécution est important. L'exécution d'un KSAR peut influencer l'exécution des autres.

- Cas où les deux situations précédentes sont combinées.

Afin d'appréhender la nature du conflit entre les KSARs, il convient de connaître ce que représentent les objets, ainsi que les répercussions résultant du déclenchement des modules sur ces objets. L'entité de contrôle doit savoir si les hypothèses générées à la suite du déclenchement d'un KSAR sont concurrentes ou complémentaires vis à vis de ces mêmes objets. Ainsi, appréhender les influences réciproques entre des déclenchements de KSARs concurrents permet d'éviter les inconsistances au système.

1. Une protéine est une séquence d'acides aminés dont la liste est la structure primaire - e.g., AGTAA...

2.3.2 Dimensions des connaissances

D'après les conflits énumérées dans la section précédente, la connaissance de contrôle est caractérisée par deux aspects importants: la contribution et l'interaction.

- Contribution : les KSARs postulent que les modules sont déclenchables par des objets ou événements du tableau. L'entité de contrôle doit être capable d'identifier l'apport du déclenchement d'un module à l'avancée du processus de résolution.

- Interaction: cette dimension concerne les relations liant les modules entre eux. Ces relations peuvent être la coopération ou la compétition des modules entre eux. La coopération suppose que les modules sont engagés dans un processus consistant à construire des objets constituant une solution partielle, résolvant ainsi un sous-problème. La compétition suppose l'application de plusieurs modules différents sur les mêmes configurations d'objets du tableau. Ce processus peut engendrer des résultats divergents voir opposés.

En outre, la connaissance de contrôle possède un contenu qui peut être statique ou dynamique. Fondamentalement, le contenu dynamique se rapporte à la détection des situations où cette connaissance s'applique. Par exemple, la création d'un KSAR est détectée par l'entité de contrôle afin de cerner sa contribution éventuelle. En revanche, le contenu statique décrit a priori le rôle de la connaissance de contrôle en terme d'apports pour le processus de résolution. Finalement, les connaissances de contrôle sont utilisées dynamiquement en considérant l'état du tableau (les objets créés), et statiquement en suivant un schéma a priori. Dans ce dernier cas, l'entité de contrôle ne tient pas compte de l'état du processus de résolution et en conséquence perd sa propriété d'opportunisme. En conclusion, la partie détection de la connaissance de contrôle garantit l'opportunisme du système.
Les rapports entre les modules, représentés par les connaissances de contrôle ont pour but de rendre le choix du prochain KSAR à déclencher opportuniste.

2.4 Manipulation des connaissances de contrôle

L'autre aspect concernant les connaissances de contrôle consiste à examiner les stratégies de leur utilisation. En effet, les choix d'utilisation de ces connaissances établissent quels sont les schémas et les stratégies de résolution de problèmes que l'entité de contrôle doit conduire. Dû au fait que la connaissance de contrôle s'appuie conjointement sur l'état courant du tableau et les apports des KSARs ainsi que leur coopération/compétition, les stratégies de leur utilisation comportent aussi deux dimensions:

- Les choix opérés reflètent la région du tableau qui doit faire l'objet d'une attention de la part de l'entité de contrôle. Le choix de la région sur laquelle l'attention est focalisée engendre la sélection du prochain sous-problème à résoudre.

- Les choix déterminent l'utilisation et l'orientation des contributions/ interactions.

2.4.1 Localisation des régions d'attention

La localisation des régions d'attention est un processus consistant à déterminer quelle est la portion du tableau sur laquelle le processus de détection des connaissances de contribution et d'interaction doit se concentrer. Elle est similaire à une opération de filtrage. Ainsi, il existe

trois stratégies de focalisation d'attention de l'entité de contrôle à partir de l'état courant.

- L'entité de contrôle choisit les modules à déclencher. En second lieu, elle choisira les objets sur lesquels elle activera ces modules pour constituer un couple (module, objets). La détermination du couple (module, objets) permet de savoir quel est le prochain KSAR à déclencher.
- L'entité de contrôle désigne d'abord les objets constituant la région d'attention, ensuite, elle détermine quels sont les modules qui sont susceptibles de s'appliquer sur ces objets. Cette opération lui permet de déduire le prochain KSAR à exécuter.
- L'entité de contrôle choisit une stratégie qui combine le choix sur les objets et les modules à appliquer.

Ainsi, la partie détection de la connaissance de contrôle correspond à la localisation de la région de focalisation, tandis que le contenu correspond au foyer.

2.4.2 Planification

Dans une architecture tableau noir, le tableau est organisé en niveaux hiérarchiques d'abstraction. Les objets sont des descriptions abstraites d'objets initiaux. Ces descriptions sont exprimées dans un formalisme caractérisant les niveaux auxquels appartiennent les objets. D'après cette organisation, un module se déclenche sur des objets appartenant à un niveau donné afin de créer des objets sur un niveau supérieur. Le module en question est dit ascendant ("bottom-up"). En opposition, si les objets créés appartiennent à un niveau inférieur, le module est dit descendant ("top-down"). Parmi les systèmes qui ont une interprétation ascendante, on peut citer le système HEARSAY-II. En revanche, CRYSALIS est un système qui a une interprétation descendante.
En fonction de la nature des modules et des connaissances qu'ils renferment, la stratégie peut être dirigée par: des objets créés, des attentes, des objectifs, etc...
Aussi, l'organisation du tableau n'est pas absolue. Elle montre que la détermination de la focalisation d'attention et de la stratégie de planification dépend directement de la signification des objets et des connaissances qui portent sur le comportement des modules.

2.5 Représentation des connaissances de contrôle

Afin d'être exploitables, les connaissances de contrôle doivent avoir un modèle de représentation afin d'être représentées dans le système. Elles peuvent être spécifiées implicitement à l'entité de contrôle. Dans ce cas, l'inconvénient majeur est qu'il faut modifier le corps du programme à chaque fois qu'une application est instanciée. Néanmoins, les connaissances de contrôle peuvent être incorporées sous forme de données ayant une représentation explicite.
L'information de contrôle est fournie par le concepteur de l'application à l'entité de contrôle. Le concepteur de l'application a une représentation statique de la connaissance de contrôle qui est perçue comme une connaissance a priori, disponible initialement. Ces connaissances sont statiques car elles n'évoluent pas au cours du processus de résolution. Les interaction entre modules sont par exemple des données a priori fournies par l'utilisateur à l'entité de contrôle.

En outre, l'utilisation de ces connaissances confère une signification à ces représentations. Afin de rendre compte de la bonne utilisation, i.e., savoir si l'utilisation est conforme à la nature des connaissances de contrôle, il est indispensable de représenter cette utilisation sous forme de structures dynamiques évoluant avec l'utilisation des connaissances représentant les principaux aspects. Les structures dynamiques sont des connaissances de contrôle générées par l'entité de contrôle durant le processus de résolution. On les appelle aussi des connaissances *a posteriori*. L'agenda contenant les KSARs représente l'ensemble des stratégies possibles à un instant donné de la résolution. Par ailleurs, la représentation des connaissances de contrôle sous forme de structures dynamiques possède un double avantage:

- L'entité de contrôle s'appuie sur des connaissances explicites. Ainsi, le concepteur peut suivre l'évolution de ces connaissances.

- Les connaissances a posteriori sont une conséquence du fonctionnement du système. L'entité de contrôle utilise explicitement les connaissances dont elle a besoin pour effectuer sa tâche.

D'après les sections précédentes, trois types de connaissances sont distinguées: les objets et les solutions, les connaissances ayant trait au comportement des modules et les stratégies concernant l'utilisation des connaissances de contrôle.

2.5.1 Les objets et les solutions

Les objets décrivent la structure de données dans un niveau d'abstraction donné. Leur représentation dépend de la structure du tableau noir. Par contre, la représentation des solutions découle de celles des objets car elles sont des instances particulières de ces mêmes objets. Les objets sont caractérisés par certaines propriétés décrites ci-dessous.

- Les objets ont une structure de type: attribut/valeur. Ils définissent l'espace des solutions et sont considérés comme des instances du niveau auquel ils appartiennent. Dans le système ATOME par exemple, les objets sont des instances d'une classe qui est le niveau.

- Les relations entre les objets sont considérées comme des propriétés particulières de l'objet. Elles sont représentées par des liens explicites. Les relations peuvent s'appliquer sur des objets de niveaux différents ou identiques. Elles peuvent être soient des relations structurelles ou de dépendances. Les relations structurelles caractérisent les positions respectives des objets d'un même niveau. Par contre, les relations de dépendances concernent des objets appartenant à des niveaux différents.

Dans les systèmes où le tableau noir a une organisation hiérarchique des niveaux d'abstraction, une solution est définie comme une configuration d'objets du plus haut niveau. C'est le cas dans le système HEARSAY-II. Dans les systèmes ATOME ou BB-1, l'entité de contrôle a un modèle explicite de la solution recherchée.

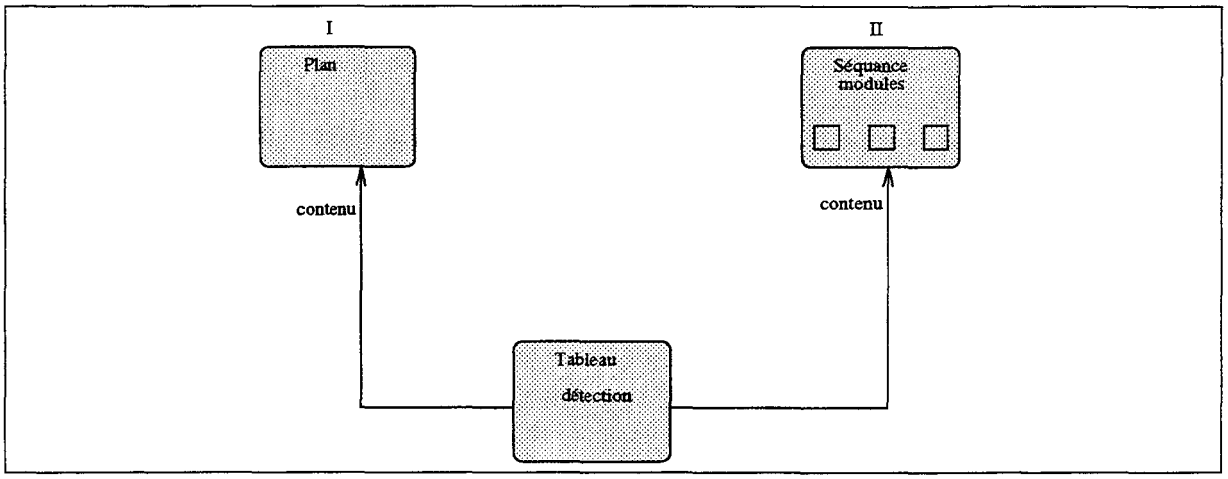

FIG. 2.1 - *Représentation de la coopération entre modules*

Les modules

Les connaissances de contrôle incorporées au niveau des modules sont de trois types: les contributions des modules au processus de résolution, la coopération et l'opposition des modules durant la résolution des sous-problèmes. Il faut noter qu'il n'y a pas de formalisme spécifique, ni de représentation particulière de ces connaissances.
La participation des modules au processus de résolution se manifeste par la création spontanée de KSARs reflétant les événements survenus au niveau du tableau noir. Les KSARs doivent être représentés explicitement afin que les contributions potentielles qu'ils véhiculent soient facilement détectables pour l'entité de contrôle. L'accès de l'entité de contrôle aux connaissances de contrôle se fait par le biais de deux méthodes:

- Quand le KSAR est exécuté, un objet *événement* est construit pour informer l'entité de contrôle des changements opérés dans le tableau. L'objet créé renferme les informations importantes pour l'entité de contrôle.

- Dès que le KSAR est créé, l'entité de contrôle consulte l'information qu'il contient et déduit ainsi la nature de la contribution du module.

Dans le premier cas, l'entité de contrôle doit d'abord activer le module pour rendre compte de la contribution de ce dernier. Dans le second cas, l'entité de contrôle accède au champs information et examine les possibilités qui s'offrent à elle.

Coopération

La coopération entre modules est exprimée par des connaissances de contrôle ayant une structure comprenant une partie détection et une partie description (voir Fig.6.7). La partie détection est représentée sous forme de condition d'une règle de production indiquant le contexte engendrant la coopération. La description correspond à la partie action.

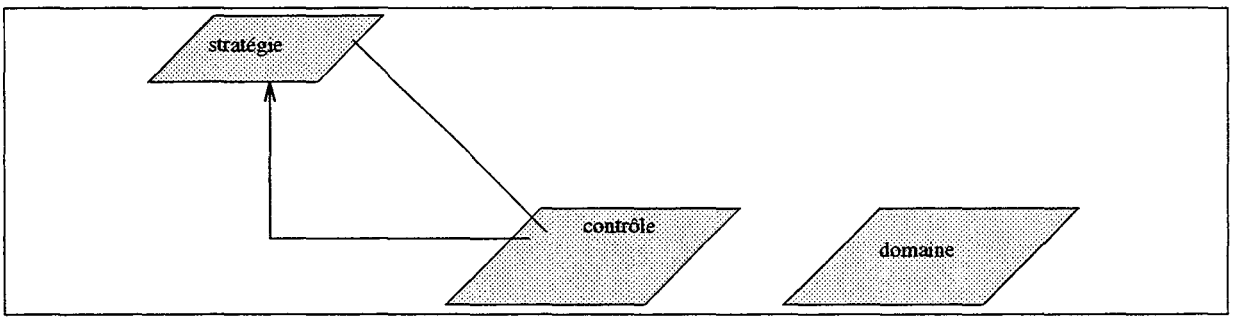

FIG. 2.2 - *Stratégies de contrôle dans BB-1*

Compétition

Des situations de conflit apparaissent quand plusieurs choix entre modules conduisent à des solutions différentes, voir opposées. Les connaissances concernant l'opposition des modules ont pour but de permettre à l'entité de contrôle de reconnaître les situations de conflit et de choisir en fonction de ces situations. Dans CARAMEL [Sab90a], le problème de la compétition est résolu via le parallélisme. Les actions concurrentes sont lancées en parallèle puis départagées en fonction des résultats obtenus.

2.5.2 Stratégies d'utilisation

Les stratégies d'utilisation indiquent comment l'entité de contrôle utilise ses connaissances sur la nature des objets et le comportement des modules. Dans des systèmes comme HEARSAY-II et CRYSALIS, les stratégies sont implicites. En revanche, elles sont explicites dans des systèmes comme BB-1 où la manière dont les connaissances sont manipulées est spécifiée explicitement (voir Fig.2.2).
La structure concernant les stratégies d'utilisation est similaire à celle des connaissances concernant le comportement des modules. Elle contient une partie détection et une partie description. Les connaissances descriptives repésentent une stratégie sous la forme d'une structure de données décrivant la stratégie. Dans le système ATOME, le contexte représenté par les règles est une abstraction des tableaux appelée résumé des tableaux. Les résumés des tableaux contiennent les objets les plus importants. Les résumés sont des structures dynamiques mises à jour durant le processus de résolution. L'entité de contrôle du système ATOME possède une représentation de l'état du tableau par le biais des résumés qui sont des représentations explicites de la connaissance de contrôle du système.
Dans le système BB-1, les connaissances de contrôle sont représentées au niveau d'un tableau noir qui leur est dédié exclusivement. Les stratégies de contrôle sont des modules structurés en règles qui réagissent aux changements du tableau induits par le processus de résolution. Cette réaction consiste à proposer des stratégies sous forme d'objets du tableau de contrôle. Ainsi, dans BB-1, le tableau de contrôle renferme des connaissances de contrôle dynamiques, i.e., *a posteriori*, tandis que les modules contiennent des connaissances de contrôle statiques, i.e., *a priori*.

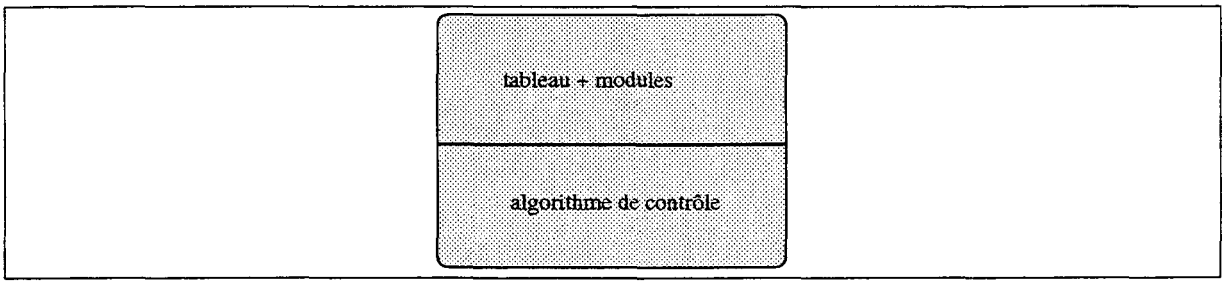

FIG. 2.3 - *Contrôle procédural*

2.6 Utilisation des connaissances de contrôle

L'entité de contrôle s'appuie sur des connaissances de contrôle initiales et statiques. Ces connaissances ont trait, à la coopération et à la compétition des modules, ou encore aux stratégies d'utilisation. En plus, l'entité de contrôle s'appuie sur des connaissances de contrôle dynamiques générées durant le processus de résolution.
Dans BB-1, le cycle de base est constitué d'une boucle gérant indifféremment les opérations de contrôle ou celles du domaine. Par contre, dans ABACAB [Bac90], les mises à jour des structures dynamiques sont opérées automatiquement en parallèle au processus de contrôle. Cette gestion automatique des mises à jour repose sur des règles de mise à jour inspectant l'état courant du système. Ces règles se déclenchent dès qu'elles soient vérifiées. Bien que le contrôle se penche sur le problème de représentation des connaissances de contrôle, il concerne aussi leur mise en œuvre. Le cycle de base est un processus incorporant plus au moins les connaissances de contrôle d'une façon implicite. Le cycle de base d'une entité de contrôle procédural est le suivant:

- Création de KSARs en guise de réaction à des changements opérés dans le tableau noir.
- Choix d'un KSAR en fonction de l'état courant du tableau et des connaissances de contrôle dont il dispose.
- Déclenchement du KSAR choisi entraînant à nouveau des changements dans le tableau.

Ce type d'entité de contrôle caractérise le système HEARSAY-II. Les autres architectures de contrôle exploitent le même cycle de base, en explicitant d'une part les connaissances qu'ils utilisent et d'autre part l'utilisation qu'elles en font. Elles sont décrites en détail dans l'annexe A.

2.7 Typologie des entités de contrôle

La spécification des connaissances de contrôle revient à définir les objets et les solutions recherchées, à décrire les comportements des modules et la stratégie d'utilisation. La typologie des architectures de contrôle peut être centrée sur le respect de l'aspect déclarativité. Ainsi, nous pouvons distinguer les architectures de contrôle suivantes:

- Entité de contrôle procédural: Le système illustrant cette catégorie est HEARSAY-II (voir Fig.2.3). Les connaissances de contrôle ainsi que l'algorithme d'utilisation sont

intégrés dans l'entité de contrôle.

- Entité de contrôle dont les connaissances initiales sont explicites, mais l'utilisation procédurale: l'archétype est le système CRYSALIS.

- Entité de contrôle dont les connaissances initiales et leur utilisation sont explicites : parmi les systèmes illustrant cette catégorie, nous pouvons citer BB-1, ATOME et ABACAB.

2.8 Critères d'évaluation

Afin de mieux appréhender le fonctionnement et les propriétés de chaque type de contrôle, il est important de définir certains critères de comparaison qui sont recensés comme suit:

- Adéquation nature/représentation: ce critère permet d'évaluer l'adéquation entre les représentations des connaissances de contrôle pour le concepteur et le formalisme utilisé par l'entité de contrôle. Il a pour rôle d'évaluer l'expressivité du formalisme de représentation.

- Adéquation représentation/utilisation: ce critère évalue si l'utilisation des représentations est explicite et s'il est formulée d'une manière pertinente pour le concepteur. Si les représentations initiales sont utilisées de façon explicite par l'entité de contrôle, il est aisé de les modifier, d'en ajouter ou de les supprimer. L'intérêt est que l'utilisation résultante est facilement déduite et le comportement du système est facile à prévoir. L'adéquation représentation utilisation est un critère de *transparence.*

- Adéquation nature/utilisation: ce critère évalue si le contrôle effectué par l'entité de contrôle est correct ou non. Il vérifie si la sémantique des représentations initiales est identique à celles de l'entité de contrôle. L'objectif est de garantir un contrôle correct et efficace.

2.9 Conclusion

Le rôle de l'entité de contrôle dans un système tableau noir est de choisir à un moment donné un module donné parmi d'autres et de l'appliquer sur un ensemble d'objets du tableau. Le concepteur d'un système tableau noir sait a priori en quoi consiste le contrôle de son système. Il spécifie les connaissances de contrôle ainsi que leur représentation. Ces connaissances se rapportent à la nature des objets et leur signification, ainsi que la nature des solutions recherchées. Elles déterminent aussi quels sont les apports potentiels des modules, les interactions entre ces modules et les stratégies utilisées pour contrôler le système. Le concepteur transmet les connaissances a priori à l'entité de contrôle doté d'un formalisme d'expression permettant de représenter explicitement ces connaissances. L'entité de contrôle dispose de structures dynamiques comme support de représentation de l'utilisation des connaissances initiales. Ces structures sont des connaissances a posteriori générées durant le processus de résolution.
En résumé, nous dirons qu'une architecture de contrôle est une machine virtuelle utilisant

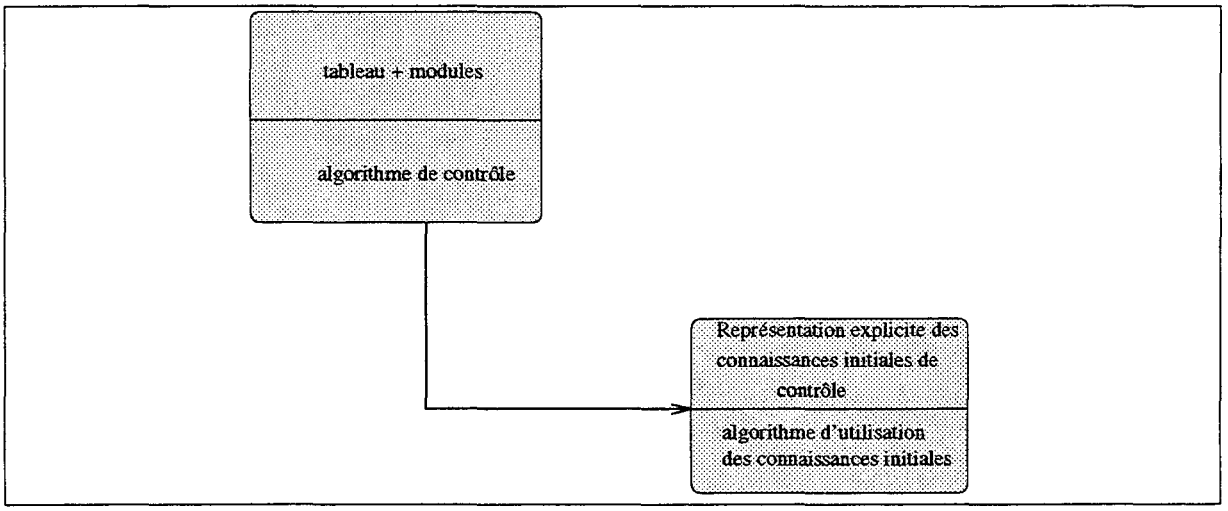

FIG. 2.4 - *Contrôle hiérarchique*

des connaissances a priori et a posteriori pour contrôler le déclenchement des modules sur le tableau. Les principaux éléments pour décrire une telle architecture sont la nature des connaissances de contrôle, la représentation initiale fournie à l'entité de contrôle, la représentation de l'utilisation qu'il en fait et ses procédures d'utilisation: cycle de base et procédure d'interprétation.

Chapitre 3

Architectures des SMAs

Préambule:

Ce chapitre trace un panorama non exhaustif des différentes architectures qui ont vu le jour dans le domaine des systèmes multi-agents. Parmi les plus connues, il y a les architectures BDI[1] *qui postulent qu'un agent doit être représenté par ses attitudes mentales définies par les croyances, les désirs et les intentions. IRMA est l'un des archétypes des architectures BDI.*

Les modèles réactifs sont le second type d'architecture présenté dans ce chapitre. Deux exemples d'architectures sont explicités à l'image des architectures de subsomption et des modèles ECO.

Le dernier type d'architecture concerne les architectures hybrides. Le but de ces dernières est de proposer un modèle d'agent intermédiaire à un agent cognitif et un à agent réactif. Un exemple de ce type d'architecture est représenté par InteRRaP.

3.1 Architecture BDI

Dans une architecture BDI, un agent est représenté par ses attitudes mentales. Ces dernières sont les croyances, les désirs et les intentions de l'agent. Les croyances d'un agent représentent le modèle de son environnement. Elles sont modélisées à l'aide de la sémantique des mondes possibles. Un ensemble de mondes possibles est associé à une situation décrivant les mondes que l'agent croit possibles. Le désir est une notion abstraite qui spécifie les préférences sur les états des mondes dans le futur. Une propriété importante des désirs est qu'un agent peut avoir des désirs inconsistants. Ainsi, l'agent n'est pas contraint de croire que tous ses désirs sont satisfiables. Par exemple, un agent a la possibilité de maintenir des

1. BDI est une abréviation qui signifie: Belief, Desire, Intention

désirs contenant les règles suivantes: (i) être capable de voyager, (ii) être riche, et (iii) ne pas avoir de travail; même si l'agent ne croit pas qu'il est possible de voyager sous l'hypothèse qui consiste à dire: *être riche implique qu'on a un travail.*
Dans certaines approches BDI plus pratiques, deux autres concepts ont été définis: il s'agit des notions de *but* et de *plan.*
La définition "légère" des désirs entraîne une étape supplémentaire de sélection d'un sous-ensemble de désirs qu'un agent peut poursuivre. Ces désirs sont appelés buts. Ils dénotent les options que l'agent a retenues. En outre, il est toujours recommandé qu'un agent croit que ses buts peuvent être accomplis.
Les intentions représentent ce que l'agent a décidé de faire. L'une des propriétés importantes d'un agent est qu'il est limité sur le plan des ressources. Par conséquent, il ne peut pas poursuivre tous ses buts simultanément. Néanmoins, même si un ensemble de buts est consistant, il est toujours nécessaire de sélectionner un certain but (ou un ensemble de buts) à accomplir. C'est ce qu'on appelle le processus de formation d'intention. Ainsi, les intentions courantes d'un agent sont décrites par un ensemble de buts sélectionnés, et l'état courant de l'agent. Par exemple, un agent peut avoir l'intention d'acheter un livre particulier, mais il n'a pas encore décidé dans quelle librairie il allait acheter ce livre. Ainsi, une structure intentionnelle est similaire à un plan hiérarchique où les opérateurs des différents niveaux d'abstraction du plan et les intentions d'une structure intentionnelle ont des propriétés similaires. Un agent BDI affine graduellement ses intentions afin d'avoir des actions primitives qui peuvent être exécutées.
Bien que la notion de plan n'est pas un composant conceptuel de la théorie BDI, il joue un rôle très important pour une implémentation pragmatique des intentions. Bratman [BIP87a] s'est focalisé sur le fait que les intentions sont des plans partiels d'action que l'agent s'est engagé d'exécuter afin d'accomplir ses buts. Ainsi, il est possible de structurer les intentions en plans, et de définir les intentions de l'agent comme des plans courants qu'il a adoptés.
Rao et Georgeff [RG92a][Rao96] ont proposé un modèle formel de l'architecture BDI, en incluant la définition des logiques utilisées, la description des croyances et la définition des désirs et des intentions comme des opérateurs modaux. L'architecture proposée est caractérisée aussi par une sémantique basée sur les mondes possibles et une axiomatisation définissant les relations et les propriétés des opérateurs BDI. Rao et georgeff ont traité les intentions comme un concept ayant le même statut que les croyances et les désirs.
un agent BDI exécute les fonctions suivantes:

1. Générer une liste d'options. Ces dernières sont des moyens pour satisfaire les intentions courantes. De nouvelles options sont générées sur la base des croyances et des désirs des agents. Les options disponibles pour l'agent permettent de satisfaire ses intentions courantes qui sont générées à travers l'affinement. Le processus d'affinement est similaire à la fonction qui consiste à affiner un plan hiérarchique. Les options disponibles qui sont des alternatives aux intentions courantes, sont générées sur la base des désirs et des croyances courantes.

2. Sélectionner un sous-ensemble de ses options pour adoption. La sélection est généralement faite sur la base des croyances, des désirs et des intentions courantes de l'agent. Ces nouvelles intentions, i.e., les options que l'agent a décidées d'adopter, sont ajoutées à la structure de l'intention.

3. S'il existe une action atomique qui peut être exécutée dans la structure intentionnelle, elle est exécutée.

4. Alors, si l'agent a satisfait une intention ou a décidé qu'une intention ne peut être satisfaite, cette intention est supprimée. Il faut souligner que Cohen et Levesque suggèrent qu'une intention doit être supprimée si elle n'est pas justifiable.

5. Modifier les croyances de l'agent et réitérer le processus à partir de la première étape.

Les architectures BDI sont un bon exemple des systèmes d'agents autonomes où le contrôle est parfaitement distribué entre les agents du système. MAGIQUE a pour spécificité d'offrir un cadre de coopération et de communication partagé entre un groupe d'agents.

3.2 Un interpréteur pour des agents BDI

La critique majeure de la théorie BDI est que les logiques BDI multi-modales n'ont pas une axiomatisation complète, par conséquent, il n'y a pas d'implémentations efficaces de ces théories expliquant ainsi la faible influence sur les actuelles implémentations des systèmes BDI. Afin de pallier à la limite précédente, Rao et Georgeff ont proposé un interpréteur abstrait pour les agents BDI. L'interpréteur abstrait décrit le contrôle d'un agent via le cycle de traitement suivant:

```
initialize_state();
repeat
      options := option_generator(event_queue);
      selected_options := deliberate(options);
      update_intentions(selected_options);
      execute();
      get_new_external_event();
      drop_successful_attitudes();
      drop_impossible_attitudes();
end repeat
```

Dans chaque cycle, la file d'événements est consultée par l'interpréteur. Subséquemment, un ensemble d'options est généré. Ces options représentent les buts que l'agent peut potentiellement poursuivre en fonction de l'état courant de son environnement. L'ensemble des options est étendu par les options générées par le délibérateur. Finalement, l'étape de formation des intentions est effectuée dans la procédure *update_intentions*. Un sous-ensemble des options générées est sélectionné comme intentions dont l'agent s'engage à exécuter les actions correspondantes. Dès que l'agent s'est engagé d'exécuter une action, le processus d'exécution est initié. Le cycle d'exécution se termine par l'incorporation de nouveaux événements dans la queue d'événements, et par une vérification des buts et intentions courants pour savoir s'ils peuvent être atteignables (dans le cas des buts), ou s'ils sont impossibles (dans le cas des désirs), ou si ils sont irréalisables (dans le cas des intentions).

Bien que cette architecture est à la base des systèmes de raisonnement pratiques, elle présente un ensemble de limites:

- L'architecture repose sur un ensemble logiquement clos, d'attitudes mentales. Les fonctions et procédures spécifiées dans le cycle impliquent des procédures de preuves qui ne sont pas calculables en général.
- La capacité du système d'être réactif est limitée par le temps nécessaire pour exécuter un cycle car les événements externes sont pris en considération une seule fois dans le cycle.
- D'ailleurs, en supposant qu'il n'y ait pas de restrictions supplémentaires sur le générateur d'options et la procédure de délibération, il n'est pas clair comment ces deux modules soient suffisamment rapides afin de satisfaire les demandes temps réel imposées à l'agent.
- Il est évident qu'il n'est pas possible d'utiliser un traditionnel démonstrateur de théorèmes car il n'y a pas de limite sur le temps de raisonnement et donc, aussi, sur le temps d'action.

Afin de remédier aux problèmes précédents, Rao et georgeff ont proposé un certain nombre d'hypothèses de restrictions et de choix de représentation qui ont conduit à une architecture pratique.

- Au lieu d'utiliser des formules arbitraires pour représenter les croyances et les buts, des restrictions ont été effectuées sur ces formules afin qu'elles soient représentées par un ensemble de littéraux sans disjonctions ni implications.
- Les croyances sur l'état courant du monde sont explicitement représentées comme étant les croyances courantes de l'agent et qui sont susceptibles de changer dans le temps.
- Les informations concernant les moyens d'atteindre certains états du monde et sur certaines options disponibles pour l'agent afin de procéder à l'accomplissement de ses buts, sont représentées par des plans.
- Les intentions de l'agent sont représentées implicitement à l'aide d'une pile d'exécution. Comme l'agent exploite une hiérarchie de plans hiérarchiques, il est possible qu'il y ait coexistence de plusieurs piles dénotant plusieurs actions courantes et indépendantes, ou encore plusieurs activités parallèles dans une action.
- Afin de garantir la réactivité à l'intérieur du cycle de l'agent, il est supposé que l'environnement subit des changements à des cadences qui soient comparables avec le cycle de calcul du système.

Il faut noter que le *système de raisonnement procédural* a été la première implémentation de l'architecture BDI basée sur ces hypothèses.
En conclusion, il n'est pas clair qu'il soit possible de traduire les spécifications abstraites en logique modale de l'agent, en programmes qui soient capables de résoudre des problèmes réels et complexes. L'approche de Rao et Georgeff représente les options de l'agent dans le contexte de ses plans. Néanmoins, ils ne fournissent pas un support pour la prise de décision entre

plusieurs options alternatives. Le cycle d'interprétation est monolithique, par conséquent, il ne dispose pas d'un support de traitement spécial des événements qui doivent être pris en charge en temps réel. Ainsi, toutes les situations sont traitées uniformément par le processus général de génération d'options. Finalement, le comportement réactif d'un agent n'est pas pris en compte dans ce paradigme.

3.3 L'architecture IRMA

IRMA [BIP87a] est une architecture pour agent à ressources limitées. cette architecture décrit comment un agent sélectionne ses actions courantes basées sur une représentation explicite de sa perception, de ses croyances et de ses intentions. L'architecture intègre un certain nombre de modules incluant: une structure intentionnelle qui est un ensemble de plans structurés en arbres, un analyseur d'opportunités, un processus de filtrage et une procédure de délibération. Au moment où les croyances de l'agent sont modifiées par sa perception, l'analyseur d'opportunité est capable de proposer des options pour des actions basées sur les croyances de l'agent. En plus, les options sont proposées par le module de raisonnement à partir de la structure intentionnelle de l'agent. Toutes les options disponibles passent à travers un processus de filtrage afin de tester leur consistance avec la structure intentionnelle courante de l'agent. Finalement, les options ayant réussi à passer le processus de filtrage avec succès sont passées à un processus de délibération qui modifie la structure intentionnelle en adoptant une nouvelle intention.
Le modèle IRMA incorpore deux vues différentes de la notion de plan. D'un coté, les plans qui sont stockés dans la librairie des plans sont vus comme des croyances que l'agent dispose sur les actions qui sont exécutées pour l'accomplissement de ses buts. D'un autre coté, l'ensemble des plans courants que l'agent a adoptés, définie sa structure intentionnelle locale. La seconde vue du plan (plans considérés comme intention) est devenue le paradigme le plus accepté dans la recherche sur les architectures BDI.
Le modèle IRMA ne fournit pas un modèle formel explicite des croyances, des buts et des intentions, ni de leur traitement. Ainsi, la contribution d'IRMA a plutôt été une définition d'un cadre de contrôle pour les agents BDI qui a servi de base à plusieurs affinements subséquents des concepts BDI.

3.4 Programmation orienté agent

Shoham [Sho93] a proposé un nouveau paradigme appelé *Programmation Orientée Agent* (POA). Dans ce paradigme, un agent est considéré comme une entité constituée d'un ensemble de composants mentaux tels que les croyances, les capacités, les choix, et les engagements. Il a adopté la notion d'*attitude intentionnelle* telle qu'elle est définie par Denett [Den87]. Un système de programmation orienté agent est défini par trois composants, (i) un langage formel pour la description des états mentaux des agents; (ii) un langage de programmation dont la sémantique correspond à celle des états mentaux des agents; (iii) un mécanisme permettant de relier les processus de bas niveau de la machine aux niveaux intentionnels représentés à travers les programmes de l'agent. Il faut noter que Shoham s'est focalisé sur les deux premiers aspects du paradigme POA.
Le langage formel comprend les catégories mentales formées des croyances, des obligations et des capacités. La notion d'obligation coïncide avec les notions d'intention et d'engagement

telles que définies par Rao et Georgeff. La capacité n'est pas représentée directement comme un concept mental dans l'architecture BDI. Elle est, plutôt, supportée par des plans exécutés afin d'atteindre certains buts. Le langage formel du paradigme de POA décrit dans [Sho93] ne discute pas les problèmes de planification et de prise de décision. Néanmoins, le langage PLACA de Thomas [Tho93] a étendu la notion de POA en fournissant un environnement de planification des capacités. Le langage repose sur un langage temporel propositionnel basé sur le point. Les catégories mentales sont introduites sous forme d'opérateurs modaux interprétés via des sémantiques de mondes possibles.
La programmation d'un agent est vu comme la spécification des conditions pour effectuer des engagements. Dans [Sho93], les agents s'engagent à exécuter directement des actions. Thomas a étendu la notion d'engagement à celle du plan. Le langage de programmation de Shoham incorpore la représentation des structures de données comme les phrases logiques et les opérateurs exta-logiques. Le processus de contrôle est implémenté à l'aide d'un interpréteur d'agent générique qui s'exécute dans une boucle à deux phases. Dans chaque cycle, l'agent lit, en premier plan, les messages courants et modifie son état mental. Ensuite, il exécute les engagements courants qui résultent des modifications supplémentaires de ses croyances. AGENT0 est une simple instance de l'interpréteur générique. Son langage comprend des représentations des faits, des actions conditionnelles et inconditionnelles (aussi bien celles qui peuvent être privées ou communicatives) et les règles d'engagement décrivant les conditions sous lesquelles l'agent va entrer dans de nouveaux engagements basés sur son état mental courant et les messages reçus à partir des autres agents. Les messages échangés entre les agents sont de trois types différents: *INFORM, REQUEST* et *UNREQUEST*. L'interpréteur de l'agent instancie la boucle de base en fournissant les fonctions de mise à jour des croyances et des engagements. Les croyances sont modifiées ou révisées après que l'agent soit informé du résultat d'exécution d'une action. Les engagements sont modifiés après des changements de croyances ou après avoir reçu des messages UNREQUEST des autres agents.

3.4.1 Limites de AGENT0

Bien qu'AGENT0 est un langage très simple, il n'est pas adapté pour développer des applications intéressantes. Ainsi, plusieurs aspects importants ont été négligés. Par exemple, AGENT0 ne spécifie pas comment les agents sélectionnent une alternative parmi plusieurs. Une autre critique de taille a été adressée par Rebecca Thomas dans sa thèse de doctorat [Tho93] intitulée: **AGENT0 agents cannot plan**. Le langage PLACA est un développement supplémentaire de AGENT0. PLACA s'est focalisé sur les capacités de planification des agents et permet aux agents de communiquer des requêtes de buts aux autres agents, tandis que, seulement, des requêtes renfermant des actions directement exécutables sont possibles dans AGENT0. PLACA étend AGENT0 en introduisant de la connaissance sur les buts que l'agent peut atteindre et affine le cycle de base de l'agent en ajoutant une étape, dépendante du temps, pour construire et affiner le plan. Thomas adopte le même point de vue que Bratman en considérant les plans comme des intentions. Elle suppose qu'un agent a un ensemble de plans; ses intentions sont décrites par un sous ensemble de plans que l'agent s'est engagé de réaliser. Bien que PLACA étend clairement l'aspect expression de AGENT0 en fournissant la notion de plan, il n'examine pas d'autres restrictions comme la motivation, la prise de décision, et la faible expression du langage sous-jacent.

3.5 Les agents réactifs

Vers le milieu des années 80, une nouvelle école de pensée émergeait. Elle est potentiellement influencée par la psychologie behavioriste. Guidées par des chercheurs comme Brooks [Bro86][Bro91], Chapman et Agre [AC87], Kaelbling [KR90], et Maes [Mae80], les architectures réactives sont développées pour des agents réactifs. Ces derniers ont un comportement de type stimulus-action. Ils prennent leurs décisions en temps réel, exploitent une quantité d'informations réduite et disposent de simples règles d'action leur permettant de réagir à des changements dans l'environnement. Certains chercheurs, à l'image de Brooks, évitent de faire référence à une représentation symbolique pour représenter la réalité. Ils préconisent d'utiliser des agents réactifs dont la prise de décision est basée sur les entrées sensorielles. La conception d'architectures réactives obéit aux hypothèses de Simon [Sim81] qui postule que la complexité du comportement d'un agent peut être une réflexion de la complexité de l'environnement dans lequel il opère, plutôt qu'une réflexion d'une conception interne complexe de l'agent. Les architectures BDI et MAGIQUE sont caractérisés par leurs agents cognitifs qui sont guidés par des buts et dont les fonctionnalités sont complexes.

3.5.1 Architectures de subsomption

Brooks a défini une architecture pour des agents intelligents appelés *créatures*. Ces dernières sont caractérisées par des propriétés de robustesse et de flexibilité leurs permettant de faire face aux changements de l'environnement. Ils doivent aussi avoir la capacité de maintenir plusieurs buts. Un système est décomposé en activités produisant des sous-systèmes pouvant être actifs en parallèle. Ainsi, l'espace de représentation est divisé en un ensemble de sous espaces. Les couches les plus basses de l'architecture sont utilisées pour implémenter le comportement de base tels que *éviter de casser certains objets fragiles ou encore de marcher autour de quelque chose dans un espace.* Les couches de haut niveau sont utilisées pour incorporer des facilités telle que la capacité de poursuivre des buts, comme exemple, *regarder autour de soi et prendre des objets, au moment où on marche dans un espace.*
Le contrôle est basé sur deux mécanismes généraux, appelés *inhibition* et *suppression.* Il est structuré en couches de telle sorte que les couches de haut niveau subsument le rôle des couches de bas niveau quand elles désirent prendre le contrôle. La capacité d'un agent robot, appartenant à une couche de haut niveau, d'accomplir son but au moment où les buts appartenant à la couche de bas niveau (par exemple, le monitoring de situations critiques) sont en cours d'accomplissement, dépend de la programmation des deux mécanismes de contrôle inter-couches. Brooks a réussi à construire des robots pour l'exploration de pièces, pour la construction de cartes et la planification de parcours.

3.5.2 Les modèles ECO

Dans [Fer89] [FJ91], les modèles ECO sont présentés comme une technique pour la résolution de problèmes distribués. Selon cette approche, les objets de la réalité sont représentés par des agents ECO. Ces derniers sont définis comme des acteurs utilisant le modèle de continuation de Agha [Agh86]. Ils sont caractérisés par la simplicité de leur comportement. La résolution d'un problème est considérée comme étant un processus consistant à atteindre un état stable dans un environnement dynamique. Les parties de base d'un système ECO sont un ensemble de comportements ECO indépendants du domaine, un ensemble d'actions dé-

pendantes du domaine d'application, et de la connaissance locale. La connaissance locale de l'agent peut être modifiée par son comportement. Un agent connaît l'état de satisfaction de son but, les agents dont dépend la satisfaction de son but, ainsi que les agents qui inhibent ses actions.
Le modèle ECO définit un modèle d'agent très simple avec des mécanismes d'interaction également simples. Dans ce cas, la coopération émerge à partir des actions locales des agents, ce qui la rend difficile à décrire et à comprendre. Comme le modèle ECO est développé sous le point de vue de la résolution de problèmes distribués, les buts et les intentions des agents doivent être codés à travers des états de dépendance et de satisfaction. En général, la conception de tels systèmes est une tâche non trivialle.

3.6 Les architectures hybrides

Les approches discutées jusqu'à présent souffrent d'un certain nombre de déficits. Tandis que les systèmes purement réactifs ont une portée limitée dans la mesure où ils implémentent difficilement les comportements dirigés par les buts, les systèmes délibératifs, basés sur des agents cognitifs, s'appuient sur des mécanismes de raisonnement dont la mise en œuvre n'est pas facile, et dont l'objectif est général. Ces derniers sont aussi moins réactifs. Une manière de surmonter ces limites consiste à faire recours aux architectures en couches. Ces dernières offrent un moyen puissant pour structurer les fonctionnalités et le contrôle d'un agent. Ainsi, elles constituent un outil important pour la conception de systèmes supportant plusieurs propriétés telles que la réactivité, la délibération, la coopération et l'adaptativité. L'idée principale est de structurer les fonctionnalités d'un agent en deux ou plusieurs couches organisées hiérarchiquement. Ces dernières interagissent entre elles afin d'avoir un comportement cohérent de l'agent. La structuration en couches d'un agent offre plusieurs avantages. L'architecture MAGIQUE ne prend pas en compte l'aspect réactivité du système car il n'y a pas de modèle particulier sur la manière d'interrompre le cycle d'exécution qui est supposé indivisible, afin de prendre en compte des événements urgents. Néanmoins, MAGIQUE peut être considéré comme hybride dans le sens où elle intègre simultanément deux modèles différents à l'image des architectures tableau noir et des systèmes d'agents d'autonomes.
Les architectures hybrides sont caractérisées par un certain nombre d'avantages:

1. Permet de modulariser un agent: des fonctionnalités différentes sont clairement séparées et liées par des interfaces bien définies.

2. Rend la conception des agents plus compacte, augmente la robustesse et facilite le debugage.

3. Exécute en parallèle des processus appartenant à des couches différentes, ce qui améliore l'efficacité du système.

4. Augmente, particulièrement, la réactivité de l'agent. Durant la planification, la couche réactive peut scruter l'environnement de l'agent afin d'identifier des situations contingentes.

5. Du moment que des types et partitions différents sont indispensables pour l'implémentation des diverses fonctionnalités, il est possible de restreindre la quantité des connaissances qu'une couche individuelle a besoin de considérer afin de prendre ses décisions.

Par exemple, la couche réactive doit seulement utiliser des informations sur l'état courant de l'environnement, tandis que la couche de planification a besoin des informations relatives à ses buts, ses engagements et les plans disponibles. Ainsi, la restriction permet à l'agent d'agir effectivement et de susciter un accroissement de la réactivité de l'agent.

Ces avantages ont rendu cette approche très populaire. Cette dernière est considérée comme une technique fortement utilisée pour réconcilier l'aspect réaction et l'aspect délibération des systèmes multi-agents.

3.6.1 Planification et réactivité

Dans [LH92], une approche pratique vers l'intégration de la réactivité et de la délibération dans le domaine de la robotique est introduite. Elle est basée sur un modèle, incorporant les aspects réactifs et les aspects planificateurs, proposé par Bresina et Drummond [DB90] dans leur architecture **ERE**. Cette dernière intègre la planification, le scheduling et le contrôle. Lyons et Hendriks ont proposé une architecture dont la structure de base est décrite par un planificateur, un réacteur et l'environnement. À l'opposé des approches hybrides examinées jusqu'à présent, le modèle de planification de Lyons et de Hendriks a pour tâche d'adapter incrémentallement le système réactif, s'exécutant sous forme de processus concurrents séparés, afin qu'il soit en concordance avec les buts poursuivis. Ainsi, le planificateur peut itérativement améliorer le comportement du composant réactif. Ce dernier est composé d'un réseau de *réactions*, i.e., des processus sensoriels sont couplés avec les processus d'action dans le sens où les premiers activent les seconds à chaque fois que leurs conditions de déclenchement sont satisfaites. Le réacteur peut agir en temps réel à tout moment indépendamment du planificateur. Le planificateur peut raisonner sur le modèle de son environnement (ME), sur la description de son réacteur (R), et sur la description de ses buts (G) dont le processus d'accomplissement est en cours, ainsi que sur les contraintes imposées par ses buts. La tâche du planificateur est de savoir continuellement si le comportement de R est conforme à celui de G. S'il n'y a pas de conformité, le planificateur change incrémentallement la configuration de R en spécifiant les adaptations à effectuer. En outre, le réacteur peut envoyer des données sensorielles collectées au planificateur afin de lui permettre de prédire l'état futur de l'environnement.
Finalement, le modèle de Lyon et Hendrik est restreint au cas d'un simple agent et ne supporte pas la modélisation des agents dans un univers multi-agent. Le modèle d'adaptation des réacteurs laisse certaines questions ouvertes, i.e., l'adaptation directe, exemple: comment les inconsistances engendrées par cette adaptation sont traitées dans un système actif?
En plus, le modèle ne traite pas la question des mécanismes de reconnaissance de situations générales permettant au planificateur de savoir si le comportement du réacteur est conforme avec les spécifications, les buts et leurs contraintes, ou non.

3.6.2 L'architecture de Ferguson

Dans [Fer92], Ferguson a décrit une architecture de contrôle en couches pour des agents mobiles et autonomes exécutant des tâches de navigation contraignantes, dans un environnement dynamique. L'architecture contient trois couches: *une couche réactive*, *une couche de planification* et *une couche de modélisation.* Ces trois couches opèrent d'une manière concurrente, i.e., chacune est reliée au sous-système sensoriel de l'agent à partir duquel elle reçoit des informations que l'agent a perçues, et au sous-système d'action vers qui elle envoie des

commandes d'action. La couche réactive est conçue pour réagir à des stimulus spécifiques à l'environnement. De l'autre coté, la couche de planification a pour tâche de générer et d'exécuter des plans pour accomplir des tâches à long terme que l'agent doit réaliser. Les plans sont stockés sous forme de structures partielles hiérarchiques dans une librairie de plans. L'exécution et la planification interfèrent afin de faire face à certaines situations de panne. La couche de modélisation fournit à l'agent la capacité de modifier des plans, suite à des changements dans l'environnement. Ces changements ne peuvent pas être traités par des mécanismes de re-planification fournis par la couche de planification. En outre, cette dernière fournit un cadre de modélisation de l'environnement de l'agent, particulièrement, un cadre de construction et de maintien des modèles mentaux des autres agents. Néanmoins, la solution de Ferguson au problème de contrôle présente certains problèmes. Les accès concurrents non restrictifs des couches de contrôle aux informations, aux actions et aux règles de contrôle globales impliquent un bon effort de conception pour analyser, prédire et prévoir les interactions possibles entre les agents. Comme chaque couche peut interagir avec les autres couches de différentes manières, par exemple, par le biais de déclenchement d'actions incompatibles et contradictoires, un grand nombre de règles de contrôle sont nécessaires. Ainsi, pour des applications complexes, la conception de règles de contrôle consistantes est un problème très difficile. Le modèle de Ferguson est, certainement, le plus avancé des approches discutées dans cette section car il incorpore la modélisation des autres agents dans la couche de modélisation.

3.7 L'architecture InteRRaP

Les agents autonomes qui exécutent des tâches dans un environnement multi-agent dynamique, doivent satisfaire certaines contraintes liées au temps réel ou encore à l'environnement. L'architecture InteRRaP a pour but de prendre en compte les quatre exigences suivantes:

1. Comportement situé: les agents doivent reconnaître des événements non attendus et doivent réagir rapidement et d'une manière appropriée à ces événements.

2. Comportement dirigé par les buts: les agents doivent sélectionner leurs actions en fonction du but qu'ils veulent atteindre, et doivent tenir compte des moyens disponibles.

3. Efficacité: les tâches doivent être résolues efficacement. Les contraintes temps réel doivent être respectées. Ces derniers objectifs sont accomplis en fournissant des accès à un ensemble de procédures ayant des propriétés d'exécution garanties.

4. Coordination: les agents doivent faire face à la présence des autres agents avec les conséquences positives ou négatives qui peuvent provenir des interactions.

L'idée centrale sur laquelle est basée cette architecture consiste à définir un agent par trois couches comme suit:

1. Une couche basée sur le comportement prenant en compte la réactivité, ainsi que la connaissance procédurale pour des tâches routinières.

2. Une couche de planification locale: elle fournit des facilités sur les moyens de raisonnement permettant d'accomplir des tâches locales et de produire un comportement dirigé par les buts.

3. Une couche de planification coopérative: elle permet aux agents de raisonner sur leurs accointances et supporte des actions coordonnées avec les autres agents.

Les couches précédentes, ainsi que leur architecture de contrôle combinent le raisonnement réactif, le raisonnement délibératif et incorpore la capacité d'interagir avec les autres agents. L'utilisation des couches constitue une technique puissante pour intégrer des fonctionnalités différentes sous forme de niveaux d'abstraction. Cette technique est utilisée de deux manières. En premier lieu, elle réconcilie les différentes fonctionnalités qui rentrent en compétition pour l'utilisation des ressources d'un agent en tenant compte du postulat que ces ressources sont limitées. Ces fonctionnalités sont la réactivité, la délibération locale et le raisonnement sur les autres agents. En second lieu, elle fournit des niveaux différents d'abstraction au niveau de la représentation de la connaissance: connaissance de niveau objet, méta-connaissance locale et connaissance coopérative. En limitant les accès des différentes couches fonctionnelles aux différentes portions de connaissances, cette classification limite la complexité représentationnelle des couches fonctionnelles basses.
InteRRaP est une architecture BDI: les états informationnels, motivationnels et délibératifs d'un agent sont décrits [RG95] par le biais des croyances, des buts et une version généralisée des notions de plan et d'intention. Les entrées de l'agent (perception) sont reliées à ses sorties (actions) par le moyen d'un ensemble de fonctions illustrant la relation entre les catégories mentales d'un agent. Contrairement aux autres architectures BDI [BIP88][RG91] [BS92], les catégories mentales ainsi que les relations fonctionnelles entre elles sont distribuées à travers trois couches hiérarchiques connectées entre elles par un mécanisme de contrôle hiérarchique. L'objectif est de trouver une solution pragmatique au problème en réconciliant les différentes exigences déclarées au dessus.

3.7.1 Modèle conceptuel d'un agent

Le modèle conceptuel de l'agent InteRRaP a été influencé par l'architecture abstraite d'un agent, proposée par Bratman [BIP87b] (voir Fig.3.1). Il décrit les ingrédients de l'état mental d'un agent et la connexion fonctionnelle de ces ingrédients, définissant les entrées et les sorties de l'agent.

Catégories mentales

L'état mental d'un agent comprend un ensemble de composants différents:

1. La *perception* courante de l'agent.

2. Un ensemble de *croyances* représentant l'état informationnel de l'agent.

3. Un ensemble de *situations* décrivant les portions structurées pertinentes des croyances de l'agent.

4. Un ensemble d'*options* représentant l'état motivationnel de l'agent. Basé sur la situation courante, un ensemble d'options dépendantes du contexte est sélectionné à partir d'un

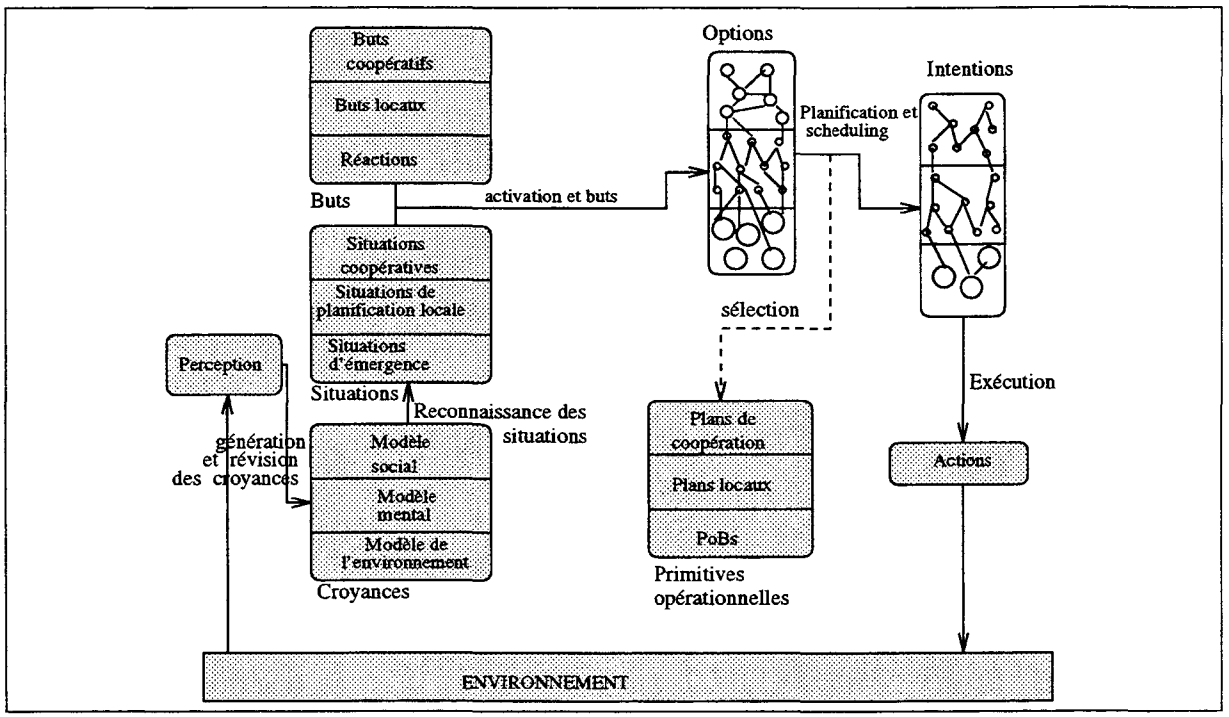

FIG. 3.1 - *Modèle conceptuel d'un agent*

ensemble de buts possibles. Ainsi, les options sont des buts que l'agent peut poursuivre si certaines conditions liées à l'environnement sont satisfaites.

5. Un ensemble d'*intentions* définissant l'état délibératif de l'agent, i.e., les options pour lesquelles l'agent s'est engagé d'accomplir, et la prochaine action que l'agent va exécuter.

6. Un ensemble de primitives opérationnelles liant l'état motivationnel d'un agent à son état délibératif.

Contrairement à l'architecture originelle de Bratman, les composants de l'état mental d'un agent InteRRaP, avec l'exception de la perception, sont structurés en couches. Les *croyances* sont coupées en trois couches: *modèle de l'environnement, modèle mental* et *modèle social.* Le modèle de l'environnement contient des croyances, sous forme d'objets représentant l'environnement de l'agent. Le modèle mental maintient des méta-croyances sur lui même. Le modèle social contient des méta-croyances sur les autres agents.
L'initiation d'une action est souvent déclenchée par des situations spécifiques, i.e., des sous-ensembles particuliers de croyances. Les situations sont des abstractions de classes d'états de l'environnement, qui sont d'un intérêt pour l'agent. Il existe trois classes de situations:

1. Situation comportementale: c'est un sous-ensemble du modèle de l'environnement de l'agent.

2. Situations nécessitant une planification locale: la description est basée conjointement sur le modèle de l'environnement et le modèle mental.

3. Situation nécessitant une planification de la coopération: la description renferme des parties du modèle social de l'agent.

Les buts sont classés en buts réactifs, buts locaux et buts coopératifs. Les buts locaux correspondent à la notion standard du but dans les architectures BDI. Les buts coopératifs sont partagés entre un groupe d'agents. Les buts réactifs diffèrent de la notion commune des buts. Ils dénotent des buts à court terme déclenchés par des événements externes et nécessitant une réaction rapide ou une exécution d'une routine.
Les *primitives opérationnelles* (PO) permettent à l'agent d'effectuer un raisonnement sur la manière de satisfaire certains buts. Les POs généralisent les notions d'opérateurs dans le modèle de planification STRIP [FHN71] et des plans dans les architectures BDI. Le concept de primitives opérationnelles inclut les notions de *modèles de comportement*, ainsi que les plans de coopération. Par conséquent, il permet une exécution flexible des procédures, ainsi qu'une coordination et une synchronisation des actions des différents agents. Comme les primitives opérationnelles sont hybrides et de complexité différente par nature, la structure intentionnelle n'est pas homogène, mais décrite à travers des couches d'abstraction différentes. Néanmoins, il y a absence d'une représentation explicite de l'intention. Ainsi, à un point spécifique dans le temps, la structure intentionnelle d'un agent est représentée par l'état courant de l'exécution:

1. Du modèle actif du comportement de l'agent,
2. De ses plans locaux,
3. Des plans de coopération et de négociation que l'agent s'est engagé à poursuivre.

3.7.2 Aspects fonctionnels de l'agent

Les fonctionnalités de l'agent décrivent les relations fonctionnelles entre les composants mentaux de l'agent. Elles définissent le flux de contrôle dans un agent, en connectant ses perceptions à ses actions. Les relations fonctionnelles sont les suivantes:

1. Génération et révision des croyances: ce processus explique la relation entre les croyances d'un agent et l'état de sa perception courante. Il illustre comment la perception est transformée en croyances, et comment les croyances existantes changent sur la base de la perception.
2. Reconnaissance des situations: ce processus extrait des situations à partir des croyances d'un agent.
3. Activation des buts: en fonction d'un ensemble de situations, l'agent identifie quels sont les buts possibles d'un agent qui sont des options.
4. Planification: ce processus consiste à associer aux buts courants de l'agent un ensemble de primitives opérationnelles pour leur accomplissement.
5. Scheduling: ce processus revient à incorporer les différents plans partiaux dans un plan d'exécution, en tenant compte des compatibilités et des contraintes entre les multiples sous-plans. Ainsi, le processus de scheduling consiste à décider *quand faire quoi.*
6. Exécution: cette étape est responsable d'une implémentation correcte des différents plans et sous-plans spécifiés dans les étapes de planification et de scheduling.

3.7.3 Architecture de contrôle

L'architecture de contrôle montre comment le modèle conceptuel de l'agent est implémenté dans un environnement de contrôle pratique. Elle décrit ses modules de base, la structure d'une couche de contrôle individuelle et le flux de contrôle entre les couches de contrôle.
Un agent InteRRaP est constitué de trois modules: un module interface avec l'environnement (World Interface), un module base de connaissances (Knowledge Base) et un module unité de contrôle (Control Unit). Le premier module fournit des liens communicatifs, sensoriels et d'action avec l'environnement de l'agent. La base de connaissances et l'unité de contrôle sont structurées, chacune, en trois couches. Les couches du module de contrôle sont: *la couche basée sur le comportement, la couche de planification locale et la couche de planification coopérative.* Chaque couche de contrôle est constituée de deux processus appelés respectivement: *processus de reconnaissance des situations* et *processus d'activation des buts.* La base des connaissances est structurée en conséquence en *modèle de l'environnement, en modèle mental,* et *en modèle social.* InteRRaP est une architecture structurée en couches verticales. Le contrôle est déclenché par une couche basée sur le comportement disposant d'un accès immédiat au système sensoriel de l'agent, puis progresse verticalement jusqu'à ce qu'une couche appropriée et *compétente* soit trouvée pour prendre en charge le service demandé. Il existe une importante distinction entre les autres approches structurées en couches, telles que celles de Ferguson [Fer92] et Kaelbling [KR90] où il y a un accès concurrent aux entrées sensorielles et aux sorties permettant à l'agent d'agir. Une différence réside au niveau de la gestion des conflits entre les couches et est basée sur une application appropriée des mécanismes globaux de filtrage et de suppression afin de permettre seulement à une seule couche d'identifier les entrées qui lui sont pertinentes et de supprimer les interactions qui ne lui sont pas parmi les décisions des différentes couches.
Chaque couche de contrôle accède à une portion spécifique de la base de connaissances de l'agent. Cet accès est organisé incrémentallement de telle sorte à ce que chaque couche de contrôle utilise des informations stockées, dans la couche correspondante à sa base des connaissances et dans les couches correspondantes aux bases de connaissances des niveaux inférieurs.

3.8 Résumé

Les architectures BDI sont les plus répandues. Elles ont focalisé beaucoup d'attention et d'efforts de la part des chercheurs experts en multi-agent. L'objectif de ces architectures est motivé par l'idée de caractériser un agent par ses attitudes mentales (croyances, désirs et intentions). Ce modèle a été enrichie par d'autres concepts tels que les buts et les plans qui forment un cadre pratique pour ces architectures. Un modèle formel de l'architecture BDI a été proposé par Rao et Georgeff. Ce modèle inclue la définition des logiques utilisées, la description des croyances, des désirs et des intentions comme des opérateurs modaux. Le modèle proposé est caractérisé, aussi, par une sémantique basée sur les mondes possibles pour ces opérateurs et une axiomatisation définissant les relations et les propriétés des opérateurs BDI. À côté des architectures BDI, les architectures réactives s'appuient sur d'autres postulats qui consistent à affirmer qu'un système est constitué d'un grand nombre d'agents ayant des fonctionnalités très simples de type stimulus/action. L'intelligence du système émerge à partir des interactions des agents réactifs du système. Les architectures hybrides se "placent" entre les architectures BDI et les architectures réactives et tentent d'intégrer les fonctionnalités et les caractéristiques des deux types d'architectures. La plupart de ces modèles d'architectures ont été utilisés avec

succès pour résoudre des problèmes complexes. MAGIQUE est une architecture dont les agents sont cognitifs, i.e., les agents sont guidés par leur buts. En plus, MAGIQUE offre un cadre de coopération global au travers des agents superviseurs. Notre architecture vise ainsi à apporter une contribution complémentaire par rapport à celles décrites dans les architectures présentées dans ce chapitre. Par exemple, MAGIQUE peut supporter un système tableau noir comme HEARSAY-II[EHRLR80] et un système multi-agent constitué d'un ensemble d'agents autonomes. Ainsi, les modules du tableau noir coopèrent via le superviseur du groupe, tandis que les agents autonomes disposent de capacités leur permettant de coopèrer directement.

Chapitre 4

MAGIQUE: une architecture multi-agent hiérarchique

Préambule:

Ce chapitre présente une nouvelle architecture multi-agent hiérarchique, appelée MAGIQUE, dont le but est de supporter des applications complexes mettant en œuvre un grand nombre d'agents. L'architecture est caractérisée par une stratégie de contrôle hiérarchique avec plusieurs niveaux de contrôle. Ainsi, MAGIQUE est caractérisée par deux types d'agents: les agents spécialistes et les agents superviseurs structurés sous forme de niveaux de contrôle.

Une description détaillée des propriétés d'un agent ainsi que du modèle de coopération et de communication entre les agents est donnée. Subséquemment, une discussion entre notre modèle et les autres architectures similaires est présentée.

Enfin, le chapitre se termine par un modèle formel de notre architecture. La formalisation vise un double objectif: (i) servir de cadre formel pour spécifier et analyser les propriétés des agents de MAGIQUE, (ii) servir aussi de cadre pour une programmation flexible des agents.

4.1 Introduction et motivations

L'apparition du paradigme de l'intelligence artificielle distribuée (IAD) et son évolution vers les systèmes multi-agents (SMAs) a ouvert des perspectives prometteuses pour la conception et la réalisation de systèmes complexes moyennant des performances nettement meilleures par rapport à celles offertes par les systèmes d'IA classique. Les SMAs sont étudiés sous trois aspects différents: le premier aspect concerne la formalisation et la spécification qui consistent à donner une représentation formelle des propriétés d'un agent [Woo94] [BEL95] [LLL+95] [GPS90]. Le second aspect a trait aux langages de communication entre agents hétérogènes

dans le cadre des travaux sur l'interopérabilité logicielle [GK94] [MLF95]. Enfin des travaux sur les modèles d'architecture adaptés à la résolution de problèmes dans un contexte multi-agent [Mul96]. Ce dernier aspect décrit comment les agents interagissent et coopèrent pour atteindre leurs buts respectifs. Dans ce cadre, notre travail s'est focalisé sur ce dernier aspect.
Le modèle d'agents autonomes postule que les agents doivent avoir une coopération «horizontale». Dans ce modèle, les agents communiquent par l'envoi de message selon un protocole basé sur les actes de langage. Le modèle est très adapté pour la résolution de problèmes où les agents sont complètement autonomes entre eux et ne nécessitent pas une structure commune de partage d'informations.
En revanche, ce modèle présente une faiblesse pour concevoir des systèmes caractérisés par leurs contraintes globales qui nécessitent d'être explicitement représentées (exemple, le budget de construction d'une maison est de 1.000.000 FR, un système de contrôle industriel doit réagir à un événement critique tous les 10 s, etc...). La limite de ce modèle apparaît aussi quand il y a une forte interdépendance entre les agents se traduisant par un fort recouvrement des domaines d'expertise des agents (un agent a besoin des résultats obtenus par d'autres agents pour progresser). Dans ce cas, il y a une difficulté à représenter le comportement global du système. Finalement, beaucoup de problèmes ne peuvent guère être résolus par des agents isolés et autonomes car ils ne possèdent pas l'expertise, les ressources et les informations nécessaires.

L'approche classique consiste à utiliser un seul agent superviseur pour gérer les problèmes de contrôle et de coordination entre les agents. Néanmoins, cette approche présente deux inconvénients majeurs: (i) toutes les interactions entre les agents passent inéluctablement par l'unique superviseur, ce qui provoque un goulet d'étranglement dans la plupart des cas; (ii) le second inconvénient est que le système est peu fiable car si une faille surgit au niveau du superviseur alors tout le système serait affecté. Afin de pallier à ces problèmes, MAGIQUE [BM97b] [BM97a] représente une alternative consistant à utiliser plusieurs agents superviseurs dans le cadre d'une architecture à plusieurs niveaux hiérarchiques (voir Fig.4.2). Cependant, l'utilisation de plusieurs niveaux hiérarchiques entraîne une complexité au niveau du contrôle et engendre une perte d'efficacité. Dans notre cas, MAGIQUE a un nombre quelconque de niveaux de contrôle, adapté à la complexité de l'application.
L'expertise du domaine est représentée au niveau le plus bas constitué d'*agents spécialistes*. Les propriétés d'un agent spécialiste sont décrites par ses attitudes mentales (croyances et buts).
Les agents spécialistes forment des groupes. Les interactions des agents du groupe sont gérées par un agent particulier appelé *superviseur du groupe*. Un groupe d'agents peut être vu comme un système tableau noir où les spécialistes jouent le rôle de sources de connaissances et le superviseur du groupe joue simultanément les rôles de tableau noir et de mécanisme de contrôle du système tableau noir. Les agents spécialistes peuvent communiquer indirectement via le superviseur, ou directement par l'envoi de messages en utilisant des actes de communication abstraits.
Dans le cas d'applications complexes, mettant en œuvre un nombre important d'agents formant plusieurs groupes, *un superviseur global* coordonne les activités entre les différents superviseurs de groupes. Par exemple, dans le cas d'un projet de construction d'une résidence constituée de plusieurs *buildings*, trois équipes d'agents peuvent travailler en parallèle, une équipe s'occupant des travaux de viabilisation (canalisations, terrassements), une équipe ayant la charge des travaux de réalisation (maçonnerie, menuiseries, finitions), la troisième équipe a pour mission de faire les installations électrique, de chauffage central, etc...

Le système pourrait avoir des contraintes de type: durée et coût de réalisation du projet. Les systèmes d'agents autonomes n'offrent pas un cadre de représentation des contraintes globales contrairement à d'autres architectures telles que les systèmes tableau noir au travers de la structure tableau noir et MAGIQUE au travers des superviseurs.

Les systèmes complexes doivent être caractérisés par une forte dynamicité. Les agents du modèle sont dotés d'une stratégie de contrôle adaptative. L'agent est capable de s'adapter dynamiquement à une situation particulière. L'architecture proposée est hybride, elle se situe entre le modèle d'agents autonomes et le modèle tableau noir.

4.2 Présentation de MAGIQUE

Les agents spécialistes sont cognitifs car ils sont dotés de capacités de perception, de raisonnement, de planification et d'action. Un agent spécialiste reçoit de la connaissance envoyée par le superviseur de son groupe, ou à la suite du changement de son environnement après l'exécution d'une ou de plusieurs actions. Par exemple, le changement de température d'un four dans une centrale nucléaire est perçu par un agent robot comme un événement important qui doit être pris en compte immédiatement. Un agent spécialiste peut être appelé à s'adapter à une situation en changeant ses règles de raisonnement. Sa base de croyances courante devient obsolète, l'agent charge une nouvelle base des croyances afin de faire face à la nouvelle situation. Un robot peut être appelé à quitter son environnement et s'intégrer dans un autre groupe d'agents pour remplacer un autre robot tombé en panne par exemple. Ainsi, il doit pouvoir s'adapter au nouvel environnement, d'ou la nécessité de charger une nouvelle base de croyances adaptée à cet environnement.
L'agent se trouve donc caractérisé par plusieurs bases de règles de croyances. Une base de règles des croyances est un ensemble de règles dont chacune est une conjonction d'atomes constitués de prédicats logiques de premier-ordre. Il dispose d'une mémoire locale dans laquelle il maintient l'état de son environnement. En terme d'IA classique, cette mémoire locale est une base de faits. Dans le cas de notre modèle, la mémoire locale contient des faits Prolog.

Un agent spécialiste est caractérisé par les attributs suivants: une mémoire locale, un ensemble de bases de règles de croyances, un ensemble de capacités, un ensemble de bases de règles de coopération, une fonction de calcul des priorités entre les capacités (voir Fig.4.2). Les capacités de l'agent représentent l'ensemble des actions qu'un agent est capable d'exécuter.

4.2.1 Croyances

Elles représentent l'expertise ou la connaissance de l'agent. Elles constituent une représentation de l'environnement de l'agent. Dans l'implémentation courante du modèle, les croyances sont représentées par un ensemble d'atomes logiques de premier-ordre.
Les croyances proviennent de quatre sources différentes:

- De l'application de la base des croyances.
- De la perception du monde réel.
- Du superviseur du groupe.
- De l'exécution d'une action privée.

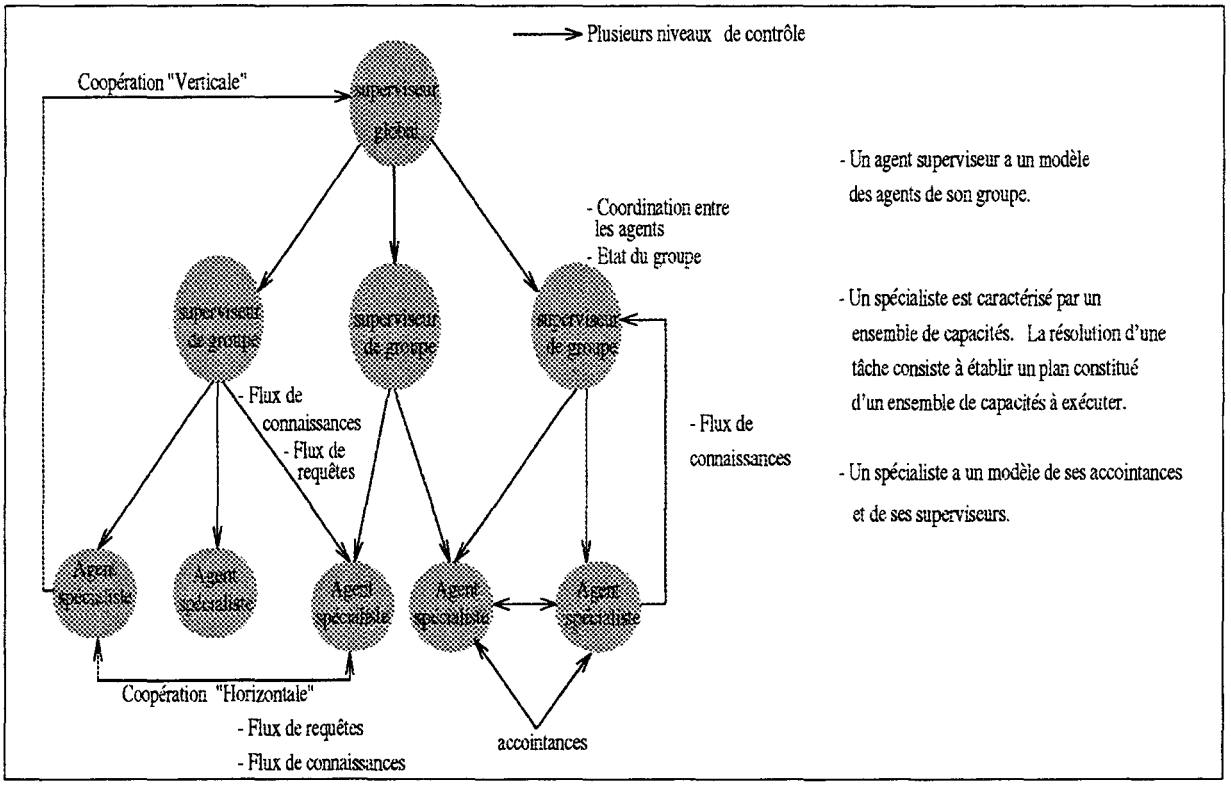

FIG. 4.1 - *Architecture du système*

- d'un autre agent spécialiste.

La perception du monde réel consiste à capter une connaissance à partir de l'environnement dans lequel évolue l'agent. La connaissance perçue est un fait Prolog. Elle est supposée être "fraîche" et est donc toujours prise en compte. A chaque fois qu'un agent perçoit son environnement, une fonction de maintien de la cohérence supprime de la base des croyances toutes les croyances qui sont incompatibles avec la nouvelle information. La base des croyances est représentée sous forme d'un graphe de dépendances entre les croyances. Nous n'allons pas détailler cette fonction dans le cadre de ce travail.
Le superviseur du groupe peut informer ses agents qu'une nouvelle croyance, les concernant, est valide. La dernière source correspond à l'exploitation des ressources internes de l'agent. Par exemple, un agent secrétaire peut consulter l'agenda de disponibilité de son usager. Un agent peut consulter une base de données et connaître une nouvelle information. Cette opération n'est pas visible des autres agents. Le résultat d'une telle action peut être considéré comme une nouvelle croyance.

4.2.2 La base de règles de croyances

La base de règles des croyances détermine comment de nouvelles croyances sont générées à partir des anciennes. Elle représente la dimension raisonnement de l'agent car elle lui permet de raisonner sur ses croyances. Cette activité de raisonnement engendre un changement au

sein de l'état mental de l'agent (changement de la base des croyances) dont la conséquence est l'action sur l'environnement afin d'atteindre les buts de l'agent.
Une règle de croyances est une règle de production au sens standard de l'IA: elle a une partie condition et une partie conclusion. Cette dernière est déclenchée si la condition est vérifiée. Dans l'implémentation courante, les règles sont appliquées en chaînage avant.
L'existence de plusieurs bases de règles des croyances permet à l'agent de s'adapter à des environnements disjoints et variés. Cependant, une seule base de règles des croyances est active à un moment donné. Ainsi, l'architecture dispose d'un programme, appelé *filtre*, lui permettant de sélectionner, en fonction du résultat de son activité de perception, la base de règles des croyances la plus adaptée aux changements de l'environnement. La multiplicité des bases de règles de croyances confère au système un comportement adaptatif.

4.2.3 Base de règles de coopération

La base de coopération détermine comment les connaissances de contrôle, nécessaires pour la coopération, sont générées par les agents spécialistes. Cette base est similaire à la base des croyances en ce qui concerne la partie condition; la partie conclusion est une action qui consiste à générer un fait et à l'envoyer, au superviseur du groupe si le fait affecte la coopération au niveau du groupe (dans ce cas, le fait est appelé méta-connaissance), où au superviseur global si le fait pourrait affecter l'état global du système (le fait est appelé dans ce cas méta-méta-connaissance). Ainsi, l'agent spécialiste participe à la coopération en filtrant la connaissance de contrôle, qui relève du domaine d'intérêt du groupe ou du système entier, pour la focaliser vers son superviseur du groupe où vers le superviseur global. La règle de coopération est déclenchée si sa condition est unifiée avec un ensemble de croyances de la base des croyances.

4.2.4 Les capacités

Elles représentent les actions que l'agent est capable d'exécuter. Elles sont aussi appelées le savoir-faire de l'agent. Elles sont représentées sous forme de règles de systèmes experts de la forme: «priorite» «condition» «action». Un coefficient de priorité est calculé dynamiquement, en fonction de l'état de l'agent, et est attribué à l'action à exécuter.
Les capacités déclenchables sont exécutées selon l'ordre de leur priorité. Les actions exécutables durant un cycle de fonctionnement d'un agent constituent le plan courant de l'agent. A chaque cycle, une fonction calcule dynamiquement la priorité à attribuer à une action déclenchable. Cette fonction prend comme entrée une liste de capacités, attribue des priorités aux actions correspondantes en fonction de l'état interne de l'agent et rend une liste d'actions ordonnées.

4.2.5 Cycle d'exécution d'un agent spécialiste

Le comportement d'un agent spécialiste durant un cycle d'exécution est déterminé par l'algorithme de scheduling suivant:

1. *Percevoir l'environnement.*
2. *Modifier les croyances à travers une fonction de maintien de la vérité;*

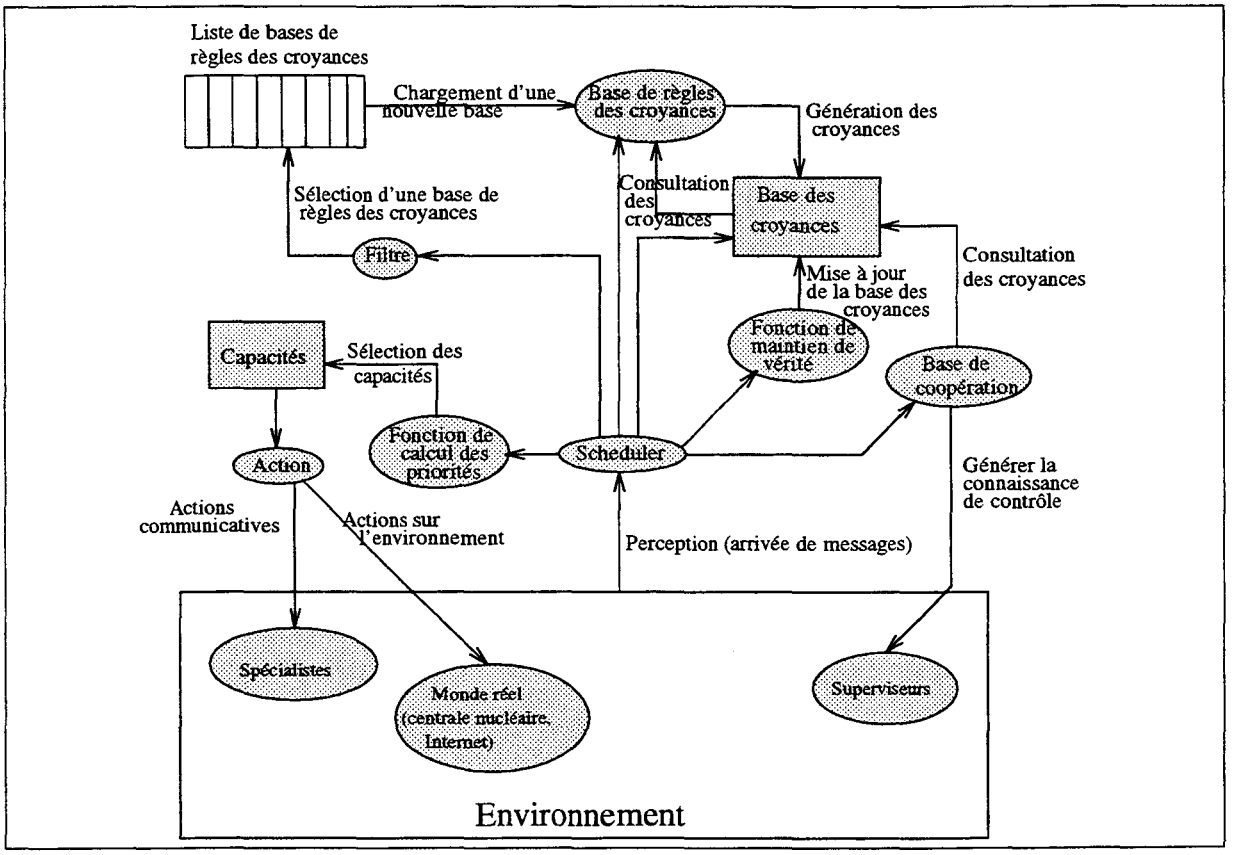

FIG. 4.2 - *Modèle conceptuel d'un spécialiste*

3. Si une base de croyances est plus adaptée que la base courante est satisfaite alors la charger avec la base de coopération correspondante;
4. Appliquer la base de croyances courante en chaînage avant;
5. Appliquer la base de coopération;
6. Envoyer aux superviseurs la connaisance nécessaire pour le contrôle;
7. Sélectionner les capacités satisfaites;
8.Appliquer la fonction de calcul des priorités entre les capacités;
9. Exécuter les actions déclenchables.

Le comportement d'un spécialiste est une boucle qui consiste à réagir aux changements de l'environnement, à raisonner sur ses croyances, à sélectionner un ensemble de capacités et à agir sur l'environnement.

Un agent superviseur est caractérisé par les attributs suivants: un ensemble de bases de règles de croyances, la base des croyances et un ensemble de méta-capacités (voir Fig.4.3). La base de règles des croyances est semblable à celle de l'agent spécialiste. La base de croyances du superviseur du groupe est vue comme un tableau noir du groupe, par contre celle du superviseur global est vue comme un tableau noir global du système. Les agents spécialistes coopèrent via le tableau noir de leur superviseur du groupe, ou via le tableau noir du superviseur global.

L'existence de plusieurs bases de règles des croyances permettent au superviseur d'avoir un

comportement adaptatif. En fonction du contenu de sa base des croyances, le superviseur peut adopter un mode de raisonnement adapté. La sélection de ces bases de règles des croyances se fait à partir de l'identification de configurations particulières des croyances par le biais d'un filtre de reconnaissance. Ainsi, le superviseur peut adopter un mode de raisonnement spécifique à une situation particulière. Les croyances d'un superviseur de groupe proviennent de deux sources: les agents spécialistes ou les superviseurs appartenant au niveau immédiatement supérieur, par contre celles du superviseur global proviennent uniquement des agents spécialistes.

4.2.6 Les méta-capacités

Ce sont les capacités du superviseur du groupe. Elles sont similaires aux capacités des agents spécialistes.
Elles sont de la forme «priorite» «condition» «meta-action». La partie méta-action est constituée d'un ensemble d'actions qui seraient exécutées par les agents du groupe. Si la condition est satisfaite, ces actions sont sélectionnées par le superviseur du groupe et sont envoyées aux agents spécialistes concernés. Elles sont vues par l'agent spécialiste comme des actions sélectionnées qui seraient déclenchées si leur condition est satisfaite au niveau de l'agent spécialiste. La méta-capacité n'est pas seulement une opération de sélection des actions au niveau des agents du groupe, elle peut être une action communicative consistant à informer un agent du groupe qu'une croyance est valide. L'ensemble des méta-actions constitue le plan du superviseur du groupe. La $meta - capacite^n$ appartient à l'ensemble des capacités du superviseur de niveau n.

4.2.7 Cycle d'exécution d'un superviseur

Le comportement d'un agent superviseur durant l'exécution est spécifié à travers le cycle de scheduling suivant.

1. Percevoir l'environnement.
2. Modifier les croyances à travers une fonction de
maintien de la cohérence;
3. Si une base de croyances, plus adaptée que la base courante,
est satisfaite alors la charger;
4. Appliquer la base de croyances courante en chaînage avant;
5. Sélectionner les méta-capacités satisfaites;
6. Appliquer la fonction de calcul des priorités entre les méta-capacités;
7. Exécuter les méta-actions déclenchables.

À travers les cycles de fonctionnement des agents, nous remarquons que les agents peuvent avoir une communication horizontale ou verticale.

4.2.8 Coopération horizontale

La communication horizontale caractérise les agents spécialistes. Ces derniers utilisent des actes de communication abstraits appelés actes de langage [Sea69] [Bra92]. Dans le cas de MA-

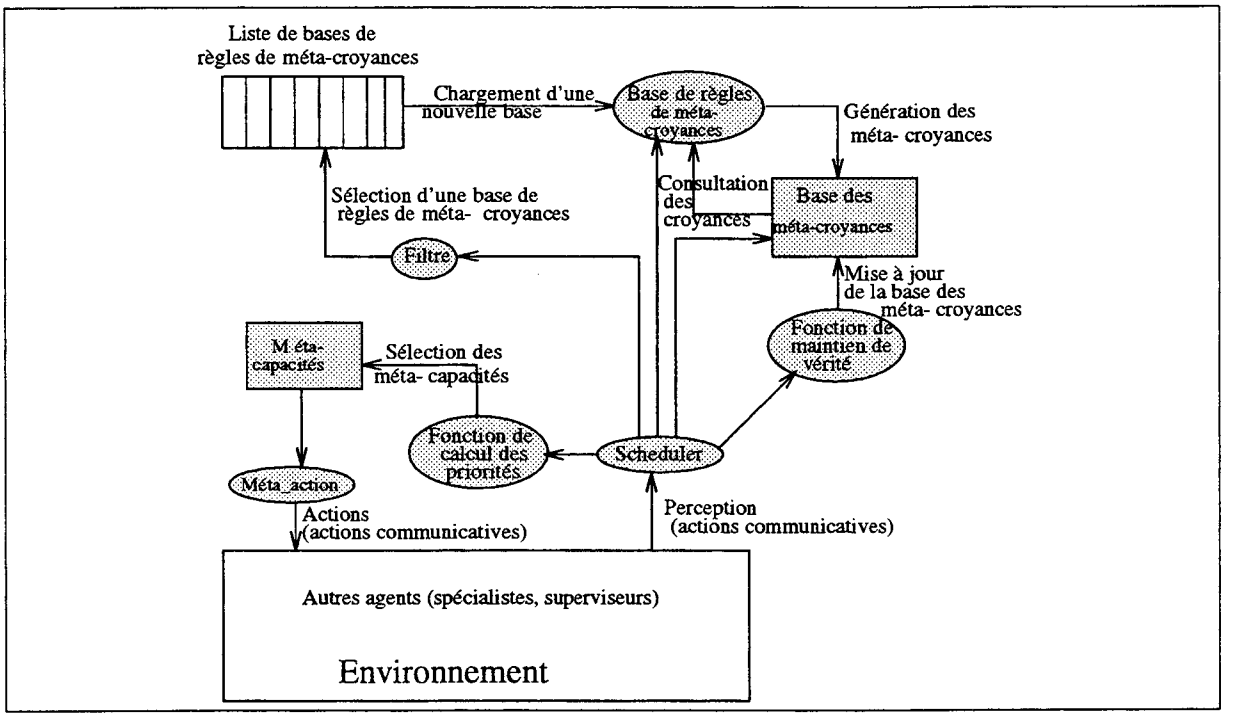

FIG. 4.3 - *Modèle conceptuel d'un superviseur*

GIQUE, les spécialistes utilisent deux primitives de communication appelées respectivement: *informer* et *requête*. La primitive *informer* permet d'informer un autre agent accointance qu'une croyance est valide (voir Fig.4.4). La syntaxe est la suivante:

informer(Spécialiste, Croyance);
Spécialiste $\in$ à l'ensemble des spécialistes.

L'exécution de cette action entraîne un changement dans l'état mental de l'agent accointance se traduisant par l'exécution d'un cycle de scheduling. Ce dernier consiste à rajouter la croyance à la base des croyances de l'agent destinataire, puis à saturer la base de règles des croyances et enfin exécuter les actions déclenchables.

La seconde action de communication consiste à envoyer une requête à un autre agent spécialiste afin de solliciter un service. La syntaxe de la primitive est la suivante:

requête(Agent, Service(Liste_Parametres));

À la différence de l'approche objet basée sur l'envoi de message sous forme d'appel procédural entre objets, l'approche multi-agent autorise un agent à soumettre une requête pour demander un service auprès d'un autre agent. Nonobstant, le service risque de ne pas être rendu dans l'immédiat faute de ressources, comme il peut être rendu par un autre agent compétent et ayant les ressources nécessaires. Le traitement de la requête soumise peut nécessiter l'exécution d'un plan d'actions tenant compte de l'état mental de l'agent appelé à rendre ce service. Cette abstraction au niveau des échanges de communication entre agents marque

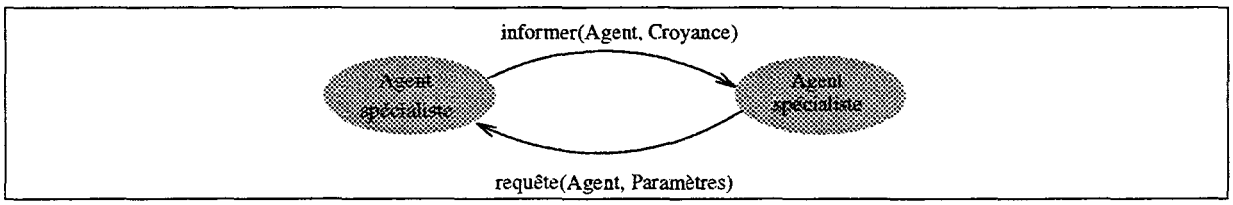

FIG. 4.4 - *Coopération entre spécialistes*

une différence de taille entre l'approche multi-agent où l'agent est caractérisé par la notion d'autonomie, qui suppose une encapsulation du comportement, et l'approche objet caractérisée par la notion d'encapsulation des données et des méthodes. Les aspects autonomie et encapsulation sont détaillés à la section 4.5.

Ce mode de coopération caractérisent aussi les systèmes d'agents autonomes qui s'appuient généralement sur un protocole classique de type *request, inform*, à l'image de Agent_0 [Sho93].

En revanche, ce type de coopération n'existe pas dans les architectures tableau noir car les agents n'ont pas de modèle de leur environnement. Les agents d'un système tableau noir s'ignorent mutuellement. Toutefois, ils coopèrent indirectement via la structure tableau noir.

4.2.9 Coopération verticale

L'interaction entre les agents appartenant à des niveaux de contrôle hiérarchiques est matérialisée par une communication dite *verticale*. La communication peut se faire entre les spécialistes, appartenant au niveau le plus bas de la hiérarchie, et les superviseurs des différents niveaux de contrôle de la hiérarchie. Les spécialistes filtrent la connaissance qui relève du domaine d'intérêt du groupe et la communique à un superviseur approprié en exécutant une primitive de communication appelée **informer**. Cette représentation de la connaissance, qui relève du domaine d'intérêt du groupe, constitue la dimension sociale de l'agent qui a un modèle de son groupe. La syntaxe de l'action communicative est la suivante:

informer(Superviseur, Croyance);
Superviseur $\in$ à l'ensemble des superviseurs.

L'action communicative du spécialiste est perçue par le superviseur comme un événement de perception. Par conséquent, la perception de cet événement a pour effet de déclencher un cycle d'exécution du superviseur. De la même manière, les superviseurs peuvent exécuter une action communicative destinée à informer un agent du groupe qu'une croyance est valide (voir Fig.4.5). La syntaxe est la suivante:

informer(Agent, Croyance);
Agent $\in$ à l'ensemble des agents.

Le superviseur peut aussi sélectionner une méta-capacité (ou une capacité) au niveau d'un agent de son groupe. Cette sélection donnera à cette capacité le statut de capacité *déclenchable*. En d'autres termes, dès que les conditions de satisfiabilité sont réunies, elle serait exécutée. La syntaxe est la suivante:

sélectionner(Agent, (meta-)capacité);
Agent $\in$ à l'ensemble des agents.

Ce modèle de coopération est similaire à celui que l'on trouve dans les architectures tableau noirs. Ces dernières sont caractérisées par une entité de contrôle qui scrute les connaissances de contrôle des modules du système et déterminent parmi ces modules ceux qui sont déclenchables afin qu'ils soient activés dans le prochain cycle du système. La différence avec MAGIQUE, c'est que les agents spécialistes sont cognitifs, par conséquent, ils décident de leur propre "volonté" de filtrer la connaissance de contrôle et de l'envoyer à l'agent superviseur. L'application des règles de la base de coopération d'un agent spécialiste dépend de son état interne (sa base des croyances).

Ce type de coopération n'existe pas dans les systèmes d'agents autonomes où le contrôle est parfaitement distribué et où les agents du système sont totalement autonomes.

4.2.10 Contrôle hiérarchique

L'activité de contrôle consiste à permettre à un agent de décider des actions à exécuter à un moment donné. L'une des caractéristiques fondamentales de MAGIQUE est son architecture de contrôle qui est hiérarchique. Cette caractéristique vise à permettre au système de supporter des applications complexes mettant en œuvre un grand nombre d'agents. Les superviseurs ont pour rôle de coordonner les activités d'un groupe d'agents. Ainsi, ils ont un modèle de leur groupe respectif. Dans le cas de MAGIQUE, le contrôle est déterminé par le cycle de scheduling d'un agent. Il convient de souligner que pendant l'exécution d'un cycle, l'agent n'est pas interruptible. En d'autres termes, les événements qui surviennent pendant l'exécution d'un cycle sont pris en compte au cours du prochain cycle. Ainsi, le processus de scheduling a pour tâche de scruter les événements qui lui parviennent, d'une manière asynchrone, de son environnement et d'invoquer les différents composants de l'agent en tenant compte de l'état mental représenté par la base des croyances de l'agent. L'avantage avec ce type d'architecture réside dans la possibilité pour un agent superviseur d'appréhender l'état global de son groupe, et pour le superviseur global d'appréhender l'état global du système via les contraintes globales qui caractérisent ce dernier et d'autres connaissances partagées par tous les agents du système. Les spécialistes ont une dimension sociale, alors que les superviseurs sont spécialisés dans la coordination des activités entre un groupe d'agents. Finalement, nous pourrons dire que le contrôle dans MAGIQUE est hybride car il n'est ni totalement distribué ni encore moins centralisé.

4.3 MAGIQUE: une architecture multi-tableaux noirs

L'utilisation de plusieurs agents superviseurs au travers des multiples niveaux de contrôle confère au système un comportement multi-tableaux noirs (voir Fig.4.6). La structure de tableau noir est représentée par la base de croyances du superviseur qui constitue un cadre de coopération entre un groupe d'agents. La base des croyances du superviseur joue aussi le rôle de solution partielle du système. L'état global du système est déterminé par la conjonction des états des multiples tableaux noirs représentés au travers des différents superviseurs. Ainsi, l'activité de contrôle du système se trouve répartie entre un ensemble d'entités de contrôle. Le mécanisme de contrôle est centralisé car le superviseur réagit aux changements du tableau noir et active les agents de son groupe. Il est similaire à l'approche retenue dans le système

HEARSAY-II [EHRLR80], sauf que dans MAGIQUE, il y a plusieurs points de contrôle. La différence avec les systèmes tableaux noirs réside au niveau de la génération de la connaissance du tableau. Dans les architectures tableaux noirs, la connaissance du tableau est générée exclusivement par les sources de connaissances. En revanche, dans le cas de notre architecture, la connaissance du tableau est générée par les agents du groupe et le superviseur via l'application de sa base des règles de croyances. Cette différence entre MAGIQUE et le modèle du tableau noir résulte du fait que les agent de MAGIQUE sont cognitifs, tandis que les agents d'un système tableau tableau noir sont réduits à des sources de connaissances ignorant l'existence des autres agents. La multiplicité des points de contrôle, ainsi que la possibilité de supporter des systèmes à tableaux noirs confère à l'architecture MAGIQUE la capacité de supporter des applications complexes moyennant un contrôle efficace. Un autre élément qui améliore l'efficacité du système concerne l'utilisation de plusieurs tableaux noirs permettant d'éviter les goulets d'étranglement engendrés par les multiples accès à une seule structure tableau noir. Cette dernière approche a été utilisée aussi dans le système ATOME [LMH87a].

4.4 Les avantages du modèle

Dans cette section, nous discutons les avantages de MAGIQUE.

- Conception et développement d'applications complexes mettant en œuvre un grand nombre d'agents. Ces derniers ont une structure interne homogène et sont rassemblés en groupes. En plus du fait que l'approche multi-agent est bien adaptée pour gérer la complexité des systèmes, notre architecture caractérisée par son contrôle hiérarchique présente un cadre de coordination et de coopération de plusieurs groupes d'agents cognitifs.

- Le contrôle est efficace car il est hiérarchique et est basé sur plusieurs agents superviseurs. Chaque agent exécute une partie de contrôle à travers sa dimension de contrôle (décider des actions à exécuter en fonction de son état interne, de celui de son environnement et des connaissances qu'il détient sur ses accointances). La connaissance de contrôle du système est représentée explicitement au niveau des agents superviseurs. En plus, la multiplicité des points de contrôle au travers des superviseurs travaillant en parallèle et communiquant d'une manière asynchrone, confère au système une bonne efficacité.

- Le modèle supporte toutes les applications tableau noir, ainsi que les applications basées sur des agents autonomes. De ce fait, il couvre un large éventail d'applications. Pour les applications tableau noir, les agents de MAGIQUE n'auront pas à interagir avec leur accointances. Dans ce cas, le cycle de l'agent consiste à capturer une information en provenance du superviseur et à accéder au tableau noir en exécutant une règle de la base de coopération. En revanche, pour les applications d'agents autonomes, le cycle de l'agent consiste à percevoir son environnement (recevoir une information de ses accointances ou de son environnement réel), raisonner sur cette information et agir sur l'environnement. Dans ce cas, la base de coopération n'est pas utilisée.

- Représentation explicite des contraintes globales du système. Une information commune à plusieurs agents peut être représentée au niveau du superviseur. En plus, l'architecture permet de résoudre des problèmes caractérisés par une solution globale du système. Le

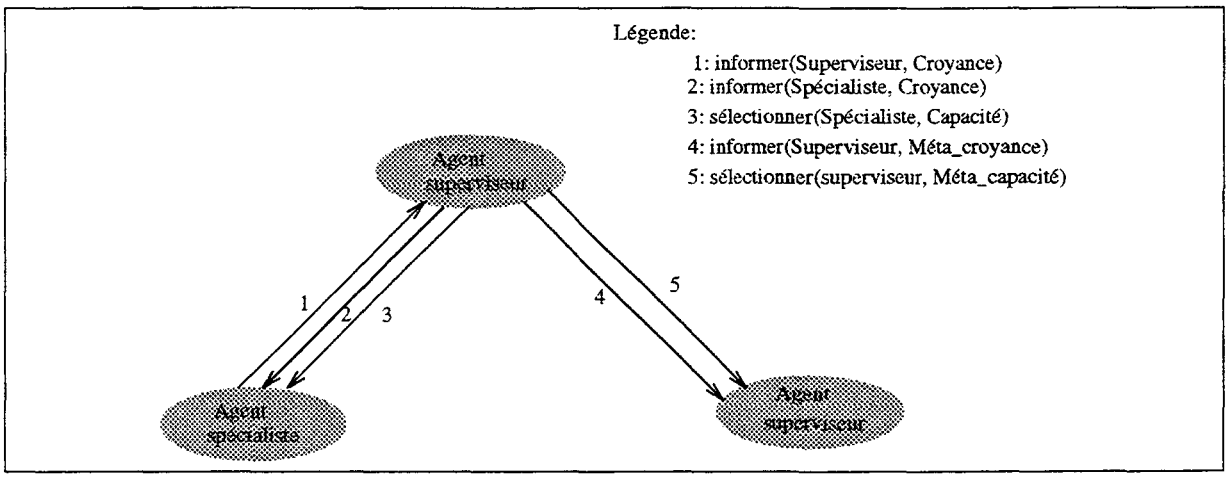

FIG. 4.5 - *Coopération verticale*

problème qui consiste à appréhender le comportement global du système constitue une sérieuse limite des systèmes d'agents autonomes.

- Le modèle est générique. Il présente l'avantage d'être instancié dans plusieurs domaines d'applications. Cette propriété de généricité fait que le modèle peut être réutilisé par d'autres personnes qui souhaitent résoudre des problèmes complexes nécessitant, un grand nombre d'agents ayant une forte granularité fonctionnelle.

- En terme de flexibilité, les agents de MAGIQUE sont sociaux et pro-actifs. La propriété de pro-activité est la capacité des agents spécialistes d'exhiber un comportement opportuniste pour atteindre leurs propres buts. Toutes les actions exécutées par l'agent spécialiste dépendent de l'état interne de l'agent et visent à satisfaire un but donné. Concenant la dimension sociale du spécialiste, MAGIQUE offre deux types de coopération: la coopération horizontale (interaction directe entre spécialistes) et la coopération verticale (interaction indirecte via les superviseurs).

4.5 Autonomie ou semi-autonomie?

Alors que la notion d'autonomie est fondamentale dans un système multi-agent, les agents de MAGIQUE, compte tenu de l'aspect hiérarchique de l'architecture et de ses fondements, garantit les propriété d'autonomie et de semi-autonomie. La propriété d'autonomie est la capacité d'un système à agir sans l'intervention directe d'un humain ou encore des autres agents [JW98]. Cette propriété implique aussi que le système doit avoir un contrôle sur ses propres actions et sur son état interne. Afin de mettre la lumière sur cette notion d'autonomie, il est important d'établir une analogie entre le concept d'autonomie qui caractérise les agents et le concept d'encapsulation pour les systèmes orientés objets. Un objet encapsule son état. Il dispose d'un contrôle sur son état dans le sens où il peut être accédé ou modifié que par le biais des méthodes que l'objet fournit. De la même manière, les agents encapsulent leur état. Néanmoins, en plus de leur état, les agents encapsulent aussi leur comportement. Les objets n'encapsulent pas leur comportement car ils n'ont aucun contrôle sur l'exécution de leurs méthodes. Par exemple, si un objet A envoie un message m à un objet B, ce dernier

n'a aucune garantie si le message m s'est bien exécuté ou non. Dans ce sens, nous dirons que l'objet B n'est pas autonome. En revanche, un agent a exactement ce type de contrôle sur les actions qu'il exécute. Finalement, nous disons que les agents invoquent des requêtes au lieu des méthodes.
Les agents de MAGIQUE respectent la propriété d'autonomie dans le sens où ils disposent de mécanismes propres qui leurs permettent de percevoir leur environnement, de raisonner sur leurs croyances et de décider, en fonction de leur état interne, des actions à exécuter. Ainsi, l'interaction horizontale entre les agents spécialistes engendrent un comportement autonome de ces derniers car les messages échangés entre les spécialistes sont, soit le message "informer" qui induit une mise à jour de l'état interne de l'agent destinataire, soit, le message "requête" dont la satisfaction est le résultat d'une activité "interne" de l'agent se traduisant par l'établissement d'un plan constitué d'un ensemble d'actions à exécuter. L'exécution de ces actions n'est pas visible de l'extérieur, par conséquent les autres agents ignorent la manière dont la requête est traitée.
Nonobstant, un agent superviseur peut sélectionner un ensemble de capacités au niveau d'un agent de son groupe. À la suite d'une configuration particulière du tableau noir du superviseur, le superviseur peut demander à certains des agents de son groupe de s'adapter à cette situation. Cette demande s'exprime par une opération de sélection de certaines capacités à exécuter au niveau de ces agents. Résultat, cette opération de sélection remet en cause le contrôle qu'a un agent sur ses actions; d'où la *semi-autonomie*. La semi-autonomie est nécessaire dans des architectures comme MAGIQUE car les agents spécialistes ont leur propre vision de l'environnement alors que le superviseur du groupe a une vision plus globale. Ainsi, un spécialiste n'a pas accès à une connaissance qui ne lui est pas locale. Dans cette situation, le superviseur peut orienter le processus de résolution vers une direction ou une autre en fonction de certaines contraintes globales du système et de l'état global du processus de résolution. Cette orientation du processus de résolution se traduit par l'exécution par des spécialistes de plans établis a priori par les superviseurs de groupe.
Finalement, la coopération horizontale engendre un comportement autonome des spécialistes, tandis que la coopération verticale suscite un comportement semi-autonome de ces spécialistes. Nous pouvons conclure que les notions d'autonomie et de semi-autonomie sont complémentaires dans le cas de notre architecture.

4.6 MAGIQUE: une structure organisationnelle

L'organisation dans un système multi-agent suppose l'existence d'un ensemble d'entités formant un "tout" et engagées dans une activité convergente. Elle suppose aussi un certain ordre entre des entités éventuellement hétérogènes, concourant à la cohérence de l'ensemble du système. Morin [Mor77] a défini la notion d'organisation comme suit:

« *Une organisation peut être définie comme un agencement de relations entre composants ou individus qui produit une unité, ou système, dotée de qualités inconnues au niveau des composants ou individus. L'organisation lie de façon interrelationnelle des éléments ou événements ou individus divers qui dès lors deviennent les composants d'un tout. Elle assure solidarité et solidité relative, donc assure au système une certaine possibilité de durée en dépit de perturbations aléatoires.* »

D'après J. Ferber [Fer95], une organisation est caractérisée par sa structure organisationnelle qui est représentée sur un plan abstrait, par une classe d'organisation. En revanche, l'organisation concrète est une instanciation, au sens des langages objets, possible d'une structure organisationnelle.
P. Delattre [Del68] a défini l'organisation par la double donnée:

- d'un ensemble de classes d'agents caractérisés par les rôles affectés aux agents.

- de l'ensemble de relations abstraites existant entre ces rôles.

L'obtention d'une organisation concrète est le résultat de l'ajout d'un ensemble d'agents appartenant aux différentes classes plus la description des relations concrètes qui existent entre les agents. En d'autres termes, il suffit d'instancier l'organisation abstraite.
Si l'on effectue une comparaison avec MAGIQUE, on pourra dire que la structure organisationnelle est définie par les classes *spécialiste* et *superviseur* qui caractérisent l'architecture. Ces deux classes constituent le modèle générique minimal sur lequel repose l'architecture MAGIQUE. L'instanciation des ces deux classes d'agents donne naissance à une application de MAGIQUE et constitue une organisation concrète. Les rôles affectés aux agents sont décrits au travers des propriétés qui caractérisent les agents (attributs, capacités, etc...); par contre, les relations entre ces rôles sont représentées par un protocole de communication entre agents.
L'intérêt de la notion d'organisation est de pouvoir intégrer à la fois la notion d'agent et celle de système multi-agent. Un module désigne les unités de niveau inférieur considérées comme indécomposables et la notion de système multi-agent général désigne le niveau supérieur. Finalement, les agents sont donc avant tout des entités individuelles alors que les systèmes multi-agent sont essentiellement des entités collectives. Pourtant, un agent pouvant être lui-même composé d'un ensemble d'agents sera considéré à un certain niveau comme une organisation individuelle et, au niveau inférieur, comme une organisation collective.
L'analyse d'une organisation doit être structurée selon trois volets:

- L'analyse fonctionnelle: elle permet de décrire les fonctions d'une organisation multi-agent dans ses différentes dimensions. Elle consiste à identifier les principales fonctions que les agents de l'organisation doivent remplir. Dans ce cadre, une organisation peut être perçue comme un ensemble de rôles dont chacun est défini par l'ensemble des propriétés que les agents ayant ce rôle expriment dans cette organisation.

- L'analyse structurale: elle identifie les diverses formes d'organisation possibles et détermine quelques paramètres structuraux essentiels. Elle a pour but d'attribuer un ordre à l'ensemble des interactions possibles entre agents en identifiant les relations abstraites qui les relient, et la manière dont elles évoluent dans le temps.

- Les paramètres de concrétisation: ces paramètres traitent du passage d'une structure organisationnelle à une organisation concrète et soulève le problème ayant trait à la réalisation effective d'une organisation multi-agent.

	Tableau noir	Systèmes d'agents autonomes	MAGIQUE
contrôle	centralisé	distribué	hybride
Granularité	faible	forte	forte
Accointances	non	oui	oui
Communication	indirecte	directe	directe/indirecte
Environnement	partagé	local	local/partagé
Perception	oui	oui	oui

TAB. 4.1 - *Comparaison de MAGIQUE avec les autres architectures*

4.7 Comparaison

L'architecture MAGIQUE incarne un nouveau modèle d'architecture intermédiaire au modèle du tableau noir et aux systèmes d'agents autonomes. Afin de bien situer MAGIQUE vis à vis des modèles précédemment évoqués, il est important de discuter les différences et les similitudes qui caractérisent les trois modèles.

La première propriété est le *contrôle* dont le rôle est de permettre au système de décider des meilleures actions à exécuter à un moment donné afin d'atteindre certains buts, tout en garantissant un comportement cohérent. Pour les architectures tableaux noirs, le contrôle est soit centralisé à l'image de HEARSAY-II, soit hiérarchique à l'image d'ATOME où encore basé sur le tableau noir comme c'est le cas pour le système BB-1 [HR85]. Il convient de souligner que le contrôle dans les architectures tableau noir est concentré, soit dans une seule entité (contrôle procédural centralisé), soit dans plusieurs entités qui peuvent être structurées en niveaux de contrôle hiérarchiques (contrôle hiérarchique) où non (contrôle à base de tableau noir). Les agents sont des "sourds-muets" car ils ne se connaissent pas mutuellement et n'ont donc pas conscience de leur environnement respectif. En revanche, dans les architectures parfaitement distribuées, il n'existe aucune entité de contrôle. Les agents sont dotés de capacités élaborées, leur permettant de contrôler leurs activités et d'interagir facilement avec leur environnement. MAGIQUE combine le contrôle hiérarchique d'un système tableau noir comme ATOME, connu par son contrôle efficace, et celui des agents complètement autonomes dépourvu de tout cadre de contrôle global nécessaire pour appréhender le comportement global du système. Par conséquent, MAGIQUE a un contrôle hybride.

La seconde propriété se rapporte à la *granularité* des agents. cette dernière exprime la complexité fonctionnelle des agents. Il est clair que les agents d'un système tableau noir présentent une faible granularité car ils sont réduits à de simples sources de connaissances structurées sous forme de paquets de règles dont le déclenchement engendre des accès à la structure tableau noir afin de créer, de modifier ou d'effacer des informations dans le tableau. En revanche, la granularité des agents des systèmes d'agents autonomes et de ceux de MAGIQUE est forte car les agents présentent une structure complexe. Cette complexité s'explique par les capacités de perception et de raisonnement des agents leur permettant d'avoir une meilleure représentation de leur environnement, ainsi que leurs capacités d'action et d'interaction avec leurs accointances afin d'accomplir des tâches complexes.

La notion d'accointance n'existe pas dans les architectures tableau noir vu que les sources

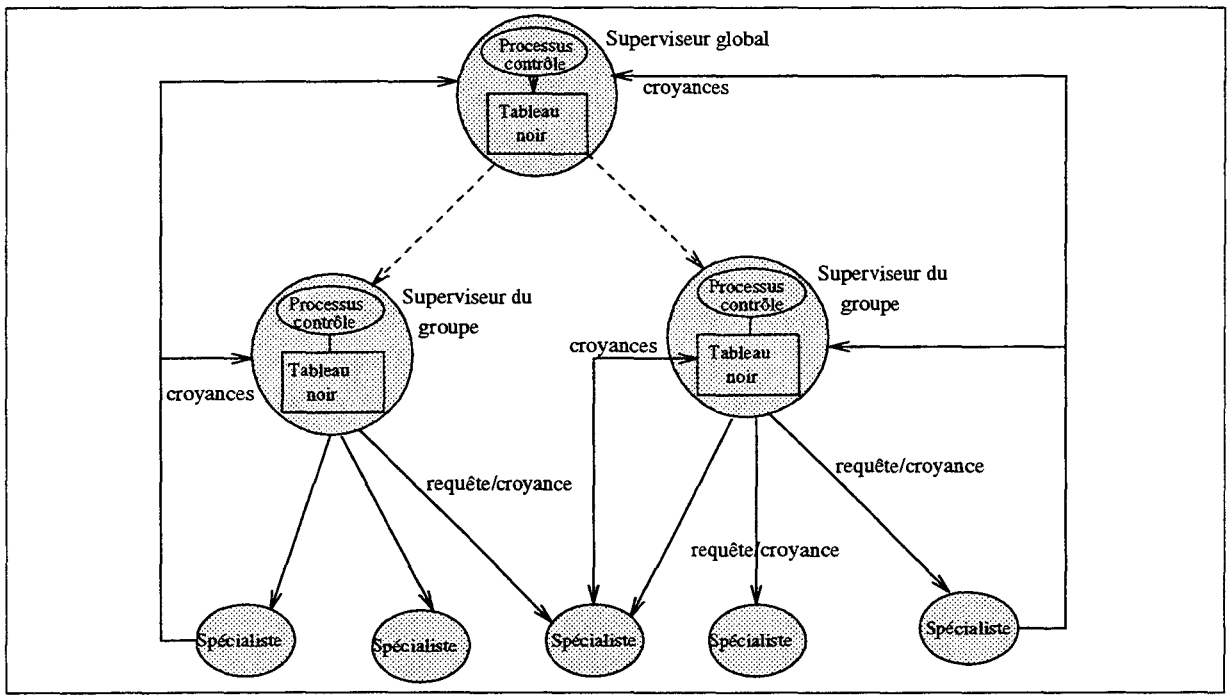

FIG. 4.6 - *MAGIQUE: Une architecture multi-tableaux noirs*

de connaissances ne "voient" que la structure tableau noir et ignorent totalement l'existence des autres sources de connaissances. En revanche, la notion d'accointance est fondamentale dans le cas de MAGIQUE et des systèmes d'agents autonomes. Dans ce dernier cas, les agents ont un modèle de représentation explicite de leur accointances. Ainsi, un agent "connaît" les services que ses accointances sont capables de rendre. La différence entre les systèmes d'agents autonomes et MAGIQUE est d'ordre architecturale car elle réside au niveau de l'existence d'un nouveau type d'agents, appelés superviseurs, dédiés exclusivement à la gestion de la coordination des activités entre un groupe d'agents spécialistes.

Une autre propriété fondamentale pour les architectures multi-agents concerne l'environnement. En ce qui concerne les architectures tableau noir, la notion d'environnement se confond avec l'état global du système représenté à travers l'état courant du tableau noir. Ainsi, on peut dire que l'environnement des sources de connaissances est partagé. À l'inverse, cette notion est extrêmement importante pour MAGIQUE et les systèmes d'agents autonomes. L'environnement d'un agent peut être le monde réel, un utilisateur, une collection d'agents, Internet, etc...
En ce qui concerne les systèmes d'agents autonomes, chaque agent a son propre environnement. Les environnements des agents sont différents, mais peuvent se chevaucher.
MAGIQUE opère une synthèse des deux environnements qui caractérisent les modèles cités en fournissant un environnement local propre pour chaque agent spécialiste et un environnement global partagé par un groupe d'agents spécialistes. Cet environnement global est représenté au niveau de la base des croyances du superviseur.

L'activité de perception est une propriété fondamentale aussi bien pour les systèmes d'agents autonomes que pour notre architecture. En effet, l'activité de raisonnement des

agents s'appuie essentiellement sur le processus de perception dont l'effet est de modifier la base des croyances représentant l'environnement de l'agent. En ce qui concerne le modèle du tableau noir, la notion de perception n'existe pas car les agents ne font qu'accéder au tableau noir après être activés par le mécanisme de contrôle.

4.8 Discussion

Ce modèle d'architecture se place dans le cadre d'une multitude de travaux effectués dans le domaine des systèmes multi-agents. Notre architecture peut être comparée au système tableau noir ATOME [LMM+87] dont le modèle de contrôle est structuré en niveaux hiérarchiques (tâches et stratégie). Ainsi, la similitude la plus importante se rapporte à l'architecture de contrôle qui est hiérarchique. La connaissance de contrôle est représentée de manière explicite au travers d'entités spécialisées dans l'activité de coordination entre les agents du système. Les agents de MAGIQUE ont des capacités plus élaborées par rapport à celles des agents d'ATOME. La différence fondamentale réside au niveau de la granularité, exprimée en terme de complexité des fonctionnalités des agents d'ATOME et de ceux de MAGIQUE. En effet, les agents de MAGIQUE ont les capacités de perception et de raisonnement sur leur environnement, alors que ceux d'ATOME sont réduits à de simples structurations de la connaissance sous forme de paquets de règles, dépourvus de toute activité de perception ni encore de représentation de l'environnement. Par exemple, dans MAGIQUE, un spécialiste peut soumettre une requête à son accointance (un autre spécialiste) pour demander un service. En revanche, un agent d'ATOME ignore tous les autres agents du système.

MAGIQUE peut aussi être comparée au système GRATE [Jen95][JML+92], prototype du système ARCHON[1] [WJM94], utilisé dans la distribution d'électricité, dont l'architecture est structurée en plusieurs modules: un module générique de coopération, un module d'évaluation de la situation, un module de contrôle et un module d'application. Le contrôle dans le système GRATE est parfaitement distribué et est représenté au niveau de chaque agent par sa dimension "module de contrôle". Le modèle de coopération est représenté par le module de coopération. Dans MAGIQUE, il existe deux types de coopérations: la coopération horizontale qui consiste à informer un autre agent qu'une connaissance est valide (cet acte de communication induit un changement dans l'état mental de l'agent destinataire), ou àlui soumettre une requête pour demander un service. En ce qui concerne la coopération verticale, le module de contrôle de GRATE est similaire à la base de coopération d'un agent spécialiste de MAGIQUE.

Shoham [Sho90][Sho93] a proposé un nouveau paradigme: la Programmation Orienté Agent (POA) et a développé l'interpréteur Agent_0. Les agents sont autonomes et sont représentés par leurs attitudes mentales. Le cycle de fonctionnement d'un agent consiste à percevoir son environnement, à modifier son état interne puis à exécuter les actions déclenchables. Le paradigme précédent a été repris dans [Gau95] pour donner naissance à un environnement de programmation orienté agent, donnant la possibilité à l'utilisateur de choisir la forme la plus appropriée pour spécifier les comportements d'un agent. Il faut noter que le contrôle des agents supportés par Agent_0 est complètement distribué, alors qu'il est hiérarchique dans le cas de MAGIQUE. En revanche, le modèle de communication dans le cas de MAGIQUE et

1. ARchitecture for Cooperative Heterogeneous ON-line systems

de Agent_0 est basé sur des actes de communication abstraits.

Le système DVMT [2] [LC83] simule un réseau d'agents participant à la régulation du trafic automobile. Chaque agent est un "solver" qui analyse des données capturées à partir des véhicules, afin de les identifier et de les localiser dans un espace à deux dimensions. Chaque solver est un agent ayant une architecture tableau noir dont les niveaux du tableau et les sources de connaissances contiennent de la connaissance appropriée pour le contrôle du trafic de véhicules. DVMT est une architecture qui illustre la coopération de plusieurs systèmes tableau noir. La différence avec MAGIQUE est qu'un groupe d'agents avec leur superviseur peuvent se comporter comme un système tableau noir, alors que chaque agent DVMT peut être vu comme un système tableau noir.

Tambe et Rosenbloom [TR95a][TR95b] ont focalisé leur efforts sur la simulation de SMAs où des agents "poursuivent" d'autres agents dans un environnement temps réel et dynamique. Ils ont examiné les implications pour les architectures basées sur les agents. Dans l'application précédente, les agents ont des comportements conflictuels, alors que les agents de notre architecture sont coopératifs.

Le modèle CONTRACT NET [DS83] est similaire à MAGIQUE concernant l'aspect hiérarchique de l'architecture. Le protocole consiste à négocier des contrats entre un agent *manager* chargé de faire un appel d'offre et un groupe d'agents spécialistes répondant à l'offre du manager. Un agent ne sait pas comment les autres agents vont accomplir la tâche qui leur est soumise, ni comment le manager exploite les résultats rendus par l'agent qui a exécuté la tâche. Il faut signaler que les agents interagissent seulement avec l'agent manager. Ils s'ignorent mutuellement, alors que dans MAGIQUE les agents ont des capacités pour coopérer directement sans passer par le superviseur. Le modèle CONTRACT NET est caractérisé essentiellement par son protocole de négociation, alors que MAGIQUE se distingue par son modèle de coopération entre agents.
Néanmoins, notre architecture ne prend pas en compte les aspects temps réel. À ce titre, une extension de MAGIQUE peut être envisagée ultérieurement, comme c'est le cas pour des systèmes ATOME-TR [LB92] et REACT [Lal92].

4.9 Problèmes visés par MAGIQUE

4.9.1 Résolution de nouveaux types de problèmes

Certains types de problèmes sont fondamentalement plus difficiles à concevoir et à implémenter que d'autres. La classe générale la plus simple des systèmes software concerne les systèmes fonctionnels. Ce type de systèmes prennent une entrée, exécutent une fonction sur cette entrée et renvoient un résultat en sortie. Les compilateurs constituent un bon exemple des systèmes fonctionnels. En revanche, les systèmes réactifs qui nécessitent une interaction constante avec l'environnement sont plus difficiles à concevoir et à implémenter correctement. Les systèmes de contrôle de processus, les systèmes d'exploitation des ordinateurs, les systèmes de gestion des réseaux d'ordinateurs sont des exemples illustratifs des systèmes réactifs. Il est clair que les systèmes réactifs sont reconnus pour être parmi les types de systèmes les plus complexes à concevoir et à implémenter [Pnu86], ce qui a concentré beaucoup d'efforts

2. Distributed Vehicle Monitoring Testbed

pour développer des outils software, des langages de programmation et des méthodologies pour la gestion de cette complexité, sans pour autant avoir eu un succès grandiose. Néanmoins, pour certains types de systèmes réactifs, même l'utilisation des techniques et outils les plus spécialisés n'a pas donné les résultats escomptés. Conséquemment, de nouvelles techniques sont incontournables pour concevoir et développer ce type de systèmes. Ces systèmes peuvent être divisés en trois classes:

- Les systèmes ouverts.
- Les systèmes complexes.
- Les systèmes omniprésents.

4.9.2 Les systèmes ouverts

Un système ouvert est un système dont la structure peut changer dynamiquement. Les caractéristiques d'un tel système résident au niveau de ses composants qui ne sont pas connus d'avance, qui peuvent changer dans le temps et qui peuvent être hautement hétérogènes (dans le sens où ils peuvent être implémentés par des personnes différentes, à des moments différents, en utilisant des langages différents). L'exemple le plus connu des environnements software ouverts demeure Internet - un réseau d'ordinateurs couplés et ouverts d'une taille et d'une complexité très étendues. La conception et le développement d'outils software pour exploiter l'énorme potentiel offert par Internet ainsi que ses retombées en terme de besoins de technologies software est l'un des "challenges" auquel sont confrontés les scientifiques ces dernières années. Internet peut être considéré comme une ressource d'information distribuée et large, avec des nœuds sur le réseau, conçus et implémentés par des individus et organisations différents. Tout système informatique connecté au réseau Internet doit être capable d'interagir avec ces organisations et autres masses d'informations, sans qu'il y ait une quelconque intervention en provenance des utilisateurs. Des fonctionnalités de ce type exigent l'utilisation de techniques basées sur la négociation, la coopération que l'on retrouve dans l'approche multi-agent [BG88]. Dans ce contexte, MAGIQUE peut jouer un rôle important pour mettre en place des services entre objets répartis et hétérogènes. Les superviseurs joueront le rôle d'interfaces de communication entre les objets. L'hétérogénéité se traduit par une interface générique implémentée sous forme de primitives à invoquer au niveau du superviseur. Ces primitives vont établir une correspondance entre une requête soumise par un agent et le service correspondant ainsi que les agents compétents pour traiter cette requête.

4.9.3 Les systèmes complexes

Parmi les outils les plus puissants pour prendre en compte la complexité dans les systèmes software, nous pouvons citer la *modularité* et l'*abstraction*. En effet, la notion d'agent fournit un outil puissant pour concevoir et implémenter des systèmes modulaires. Il est évident que si le domaine du problème est particulièrement complexe, large et incertain alors il est raisonnable de "penser" le problème en terme de composants modulaires dont chacun est spécialisé dans la résolution d'un aspect particulier de ce problème. Dans des situations où les problèmes sont interdépendants, les agents du système doivent coopérer les uns avec les autres afin de garantir que cette interdépendance soit bien gérée. Dans ce type de systèmes, l'approche multi-agent signifie que le système dans sa plénitude peut être divisé en un ensemble

de petits composants simples, facile à développer et à maintenir et qui soient spécialisés dans la résolution des sous-problèmes qui leur sont affectés. Cette décomposition permet à un agent d'utiliser le paradigme le plus approprié pour résoudre son problème particulier, plutôt que d'être forcé d'adopter une approche uniforme et commune qui représente un compromis pour tout le système, pour en fin de compte perdre toutes ses performances. La notion d'agent autonome fournit une abstraction intéressante, de la même manière que les procédures, les types de données abstraits et plus récemment les objets, fournissent une abstraction. Elle permet au concepteur de conceptualiser un système complexe en une société d'agents "solvers" coopératifs et autonomes. En plus de tous les avantages qui caractérisent l'approche multi-agent pour supporter des systèmes complexes, MAGIQUE fournit un cadre de coopération au travers des agents superviseurs dont le rôle est de coordonner les interactions entre un groupe d'agents. L'organisation des agents en groupes a pour but de supporter des applications complexes mettant en œuvre un grand nombre d'agents. Finalement, en plus de la pertinence de l'approche agent pour concevoir des systèmes complexes, MAGIQUE offre un cadre de coopération hiérarchique capable de supporter facilement des systèmes d'une très grande complexité. Par exemple, MAGIQUE peut supporter une communication entre un système comme ATOME (système tableau noir) et un système multi-agent comme RETSINA [SZ96]. Dans ce cas, MAGIQUE aura 4 niveaux hiérarchiques dont trois niveaux sont réservés pour supporter le système ATOME, et un superviseur global (quatrième niveau) pour établir une communication avec les agents autonomes de l'architecture RETSINA représentés par des agents spécialistes.

4.9.4 Les systèmes omniprésents

Malgré les multiples innovations dans le domaine de la conception d'interfaces homme-machine durant les deux précédentes décennies et l'explosion des interfaces utilisateur basées sur Windows, les utilisateurs trouvent certains logiciels difficiles à utiliser. L'une des raisons de ces difficultés au niveau de l'utilisation réside au fait que l'utilisateur doit décrire chacune des étapes qui doivent être exécutées pour résoudre un problème, y compris des détails de la couche la plus basse du logiciel. En conséquence, si les applications logicielles courantes doivent être fortement utilisées par de tels utilisateurs, alors il convient de "repenser" la nature de l'interaction entre l'utilisateur et la machine. Ainsi, on se retrouvera dans une situation de "partenariat" entre la machine et l'utilisateur. Dans ce cadre, la machine ne se comporte guère comme un récepteur de descriptions de tâches à exécuter, mais doit coopérer avec l'utilisateur afin d'accomplir ses buts. Afin de garantir ces fonctionnalités, une application doit être:

- Autonome: malgré une spécification vague et imprécise, l'application doit déterminer comment le problème est mieux résolu, et le résout sans qu'il y ait une intervention constante de l'utilisateur.

- Pro-active: l'application fournit un ensemble de suggestions à l'utilisateur, plutôt que d'attendre à ce que l'utilisateur lui "dicte" ce qu'il faut faire dans la prochaine étape.

- Réactive: elle doit tenir compte des changements des besoins des utilisateurs, ainsi que des variations dans l'environnement.

- Adaptative: l'application doit connaître les préférences de l'utilisateur et exhibe des actions d'interaction afin de prendre en compte cet aspect.

Toutes les propriétés décrites ci-dessus caractérisent l'application de notre architecture à la navigation d'un client dans un magasin virtuel en tenant compte de son profil. Les détails de l'application sont présentés dans le chapitre 6.10.

4.10 Limites de MAGIQUE

Bien que MAGIQUE présente un certain nombre de propriétés intéressantes qui le rendent adapté à la résolution de certains problèmes complexes, il souffre de certaines limites, notamment l'aspect temps réel qui n'est pas pris en compte. Ainsi, MAGIQUE n'intègre pas des fonctionnalités temps réel afin de supporter des situations nécessitant une réaction en un temps acceptable. Le problème résulte du fait qu'il n'existe aucune contraintes temporelle sur la longueur du cycle d'exécution d'un agent. Ainsi, la satisfaction des buts nécessitant une prise en compte immédiate ne s'accommode pas d'une architecture d'agents cognitifs. Ces derniers n'ont aucun modèle réactif pour réagir en temps réel aux changements de l'environnement. Une extension de ce travail est souhaitable afin d'intégrer des fonctionnalités réactives dans notre architecture.

4.11 Structure syntaxique des agents de MAGIQUE

Dans cette section, nous présentons la structure syntaxique d'un agent en spécifiant ses propriétés. Bien que notre architecture ne contient que 3 niveaux hiérarchiques (niveau spécialiste, niveau superviseur du groupe et niveau global), nous généralisons la formalisation du modèle à n niveaux hiérarchiques.

4.11.1 Croyances et base de croyances

Les croyances sont des atomes de la logique de premier-ordre. Elles sont similaires à des faits Prolog. La représentation d'une croyance nécessite la définition de deux ensembles: un ensemble de termes *Term* et un ensemble de symboles de prédicats *Pred*. Un terme est constitué de variables et de constantes.
Soient *Const* l'ensemble des constantes et *Var* l'ensemble des variables.

$$\mathbf{Const} = \{\mathbf{a, b, c,...}\} \qquad \mathbf{Var} = \{\mathbf{X, Y, Z,...}\}$$
$$\mathbf{Term} = \{\mathbf{Const} \cup \mathbf{Var}\} \qquad \mathbf{Pred} = \{\mathbf{p,q,r,...}\}$$

Une croyance est un atome défini sous forme d'une paire: symbole de prédicat et une liste de termes. Elle peut aussi être représentée par un objet structure ayant les attributs **Sym_pred** et **List_term**. Le symbole * signifie une répétition de 1 ou plusieurs items.

Croy:: Sym_pred: Pred;
List_term: Term*;

Finalement, une croyance est l'application d'un symbole de prédicat à une liste de termes.
Un agent est capable de raisonner en appliquant sa base de règles de croyances à leur base

des croyances. Conceptuellement, une règle de croyance correspond à une formule de premier ordre de la forme:

$$\forall \overline{x}, \varphi(\overline{x}, \overline{a}) \Rightarrow \psi(\overline{y}, \overline{b})$$

où $\overline{x} = x_1, ..., x_m$ et $\overline{y} = y_1, ..., y_n$ sont des tuples de variables tels que $\{y_1, ..., y_n\} \subseteq \{x_1, ..., x_m\}$, $\overline{a} = a_1, ..., a_o$ et $\overline{b} = b_1, ..., b_p$ sont des tuples de constantes, et $\varphi(\overline{x}, \overline{a})$ et $\psi(\overline{y}, \overline{b})$ sont des atomes.

Cette forme de règle est très similaire aux règles des systèmes d'IA classique. Nous supposons que les conditions peuvent contenir seulement des conjonctions; nous ne considérons pas les disjonctions ni les négations. Ces dernières nous permettent d'éviter les problèmes de négation et d'échec.
Une règle de croyance contient deux parties: la partie antécédent et la partie conséquence. L'antécédent est une conjonction d'atomes, par contre, la conséquence est représentée par un seul atome. Nous définissons les types *Cond* et *Conc*.

Cond = Atome*;
Conc = Atome;

Une règle de croyance est représentée de la manière suivante:

RCroy :: ant : Cond;
cons: Conc;

Finalement, nous définissons une fonction **saturer** permettant d'appliquer en chaînage avant une base de règles des croyances.

SATURER = BR-Croy × Croy ⟶ Croy;

Cette fonction prend en entrée un ensemble de règles de croyances et un ensemble de croyances et retourne un ensemble de croyances qui résultent de la saturation des règles de la base.
La fonction de maintien de la vérité prend la nouvelle croyance, la base des croyances, supprime toutes les croyances incompatibles avec la nouvelle croyance et retourne un nouvel ensemble des croyances.

MVC = Croy ⟶ Croy;

4.11.2 Base de règles de coopération

La connaissance de coopération est organisée sous forme de règles similaires à celles utilisées dans les systèmes experts. Ces règles sont de la forme *si condition alors action.* Elles sont définies comme suit:

BR-Coop :: Cond : Cond;
Conc : informer(Agent, Croyance);

Croyance : Croy;
Agent : Specialiste | Superviseur;

Une règle de coopération est de la forme *si condition alors action.*

si Cond alors informer(Agent, Croyance);

Si la condition **Cond** est vérifiée alors informer l'agent **Agent** que la croyance **Croyance** est valide.

4.11.3 Capacités

Une capacité est un triplet contenant (i) une partie priorité, (ii) une partie but et (iii) une partie action. La partie but représente ce que l'agent doit croire pour que la capacité soit satisfaite; La partie priorité représente le degré d'importance de la capacité par rapport aux autres capacités déclenchables; la partie action représente ce que doit exécuter l'agent si la capacité est satisfaite. Les capacités caractérisent l'agent spécialiste. Elles représentent le savoir faire de l'agent.
Soit *ACT* l'ensemble des actions qu'un agent est capable d'exécuter.
Une capacité peut être représentée comme suit:

CAPACITE :: but : Cond;
act : ACT;
priorite : N $\bigcup$ Var;

La partie variable du champ priorité de la capacité est unifiée avec l'état courant de l'agent (représenté par la base des croyances) par une fonction de calcul de priorité. Cette fonction prend en entrée un ensemble de croyances, une capacité et retourne un entier. La fonction de priorité dépend de l'application. Elle est définie par l'utilisateur.

$$\mathbf{Pr = BCroy \times ACT \longrightarrow N;}$$

4.11.4 Méta-capacités

La capacité du superviseur est similaire à celle de l'agent spécialiste; elle contient trois parties: (i) la priorité, (ii) le but, (iii) la méta-action^n (Dans notre cas, $n = 2$; le premier niveau est constitué des superviseurs de groupes, le second niveau contient le superviseur global).
En fonction de l'état courant du groupe, un superviseur peut *adopter* un plan constitué par une ou plusieurs méta-capacités. L'adoption de ce plan par le superviseur consiste à exécuter une action communicative (**informer**) pour informer un agent du groupe qu'une croyance est valide ou à **sélectionner** une méta-capacité au niveau de l'agent du groupe. Dans ce dernier

cas, les méta-capacités seront déclenchées une fois que leur condition est vérifiée au niveau des agents du groupe. Dans une application de robotique par exemple, des robots représentés par des agents sont capables d'augmenter la production. Un superviseur exécute une meta-capacité consistant à sélectionner des méta-capacités au niveau des agents de son groupe. L'exécution des méta-capacités aux niveau des agents du groupe va permettre, par exemple, aux robots de travailler à un rythme plus élévé.
Il existe trois types d'actions du superviseur:
(i) Informer un agent de son groupe qu'une croyance est valide.
(ii) Sélectionner une méta-capacité au niveau d'un agent superviseur appartenant à son groupe.
(iii) Sélectionner une capacité au niveau d'un agent spécialiste appartenant à son groupe.
Soit *Agent* l'ensemble des agents spécialistes du système. Nous définissons d'abord le type $META-ACT$.

META-ACT^n :: informer(Agent, Croyance) |
selectionner(Superviseur, Meta-cap^{n-1}) |
selectionner(Specialiste, Capacite);

Un agent peut être soit un spécialiste soit un superviseur.

Agent :: Specialiste | Superviseur;

Agent peut être un agent spécialiste (cas ou $n = 1$), ou un agent superviseur (cas où n>1).
Agent $\in$ **Liste_des_Agents_du_Groupe**

Une méta-capacité est définie de la manière suivante:

META-CAP^n :: but : Cond;
priorite : N $\bigcup$ Var;
liste-meta-act : META-ACT^n*;

4.11.5 Modélisation d'un agent

Un agent spécialiste est une structure définie par les attributs suivants:

Spécialiste :: base_croy : BCroy;
liste-bregles-croy : BR-Croy*;
liste-rcoop : BR-Coop*;
liste-cap : CAPACITE*;
mvc : MVC;
pr : Pr;
saturer : SATURER;

Un agent superviseur est représenté par une structure ayant les attributs suivants:

Superviseur :: base_croy : BCroy;
liste-bregles-croy : BR-Croy*;
liste-meta-cap : META-CAP*;
mvc : MVC;
pr : Pr;
saturer : SATURER;

Finalement, le développement d'une application sous MAGIQUE consiste à spécifier les attributs définis dans les structures précédentes et à définir les capacités ainsi que les méta-capacités des différents agents.

4.12 Conclusions et perspectives

Nous avons proposé une architecture multi-agent hiérarchique et hybride dont le but est de modéliser des applications complexes caractérisées par des contraintes globales moyennant un contrôle efficace. Les agents sont représentés par leurs attitudes mentales. Le superviseur représente un cadre de coopération d'un ensemble d'agents et permet d'adopter un plan de groupe en sélectionnant des (méta-)capacités au niveau des agents du groupe. À travers ce travail, nous pourrons dire que les systèmes d'agents autonomes et les modèles tableau noir ne sont pas antinomiques, par conséquent, ils peuvent être considérés comme étant des modèles complémentaires. Notre architecture a été instanciée pour supporter plusieurs applications notamment les systèmes d'inférence répartis, la simulation d'un ensemble de robots qui coopèrent en vue d'explorer un territoire inconnu, où encore la navigation dans un magasin virtuel d'un client à distance caractérisé par son profil. Ces applications sont présentées en détails dans les chapitres ultérieurs.
Dans le futur, nous espérons développer l'aspect maintien de vérité qui n'a pas été traité dans le cadre de ce travail. L'agent doit avoir un outil qui lui permet de maintenir la consistance de sa base des croyances. Les croyances partagées ainsi que les contraintes globales du groupe d'agents ou du système sont représentées au niveau des superviseurs. Ainsi, nous pensons focaliser nos efforts sur la conception d'un système de maintien de vérité distribué (DATMS), où chaque agent est perçu comme un solver capable de prendre en compte l'aspect incertitude qui caractérise souvent l'information dans la réalité. Le DATMS est le résultat de la coopération de plusieurs systèmes de maintien de vérité appelés ATMS[3][Kle86a] [Kle86b] dont chacun est associé à un agent du système. Les agents auront la capacité d'exploiter des informations incertaines et incomplètes en les représentant sous forme d'hypothèses. Des problèmes d'efficacité liés à l'exploration de plusieurs contextes en parallèle méritent d'être examinés. À la fin de chaque cycle d'exécution, les agents identifieront les différentes solutions partielles qu'ils ont obtenues, et relancent le processus de résolution si le but n'est pas encore atteint. le processus va se réitérer jusqu'à l'obtention de la solution globale du système.
Une autre piste d'exploration qui n'a pas été traitée par les experts en multi-agent, concerne le problème de la révision des croyances de l'agent et la nécessité d'intégrer un mécanisme de raisonnement qui prend en compte l'aspect révision [Gär88]. Ce dernier aspect permet de modéliser des situations dynamiques de l'environnement, changeant dans le temps. Il s'agit de proposer un modèle de révision qui puisse permettre à un agent de représenter d'une manière "réaliste" son environnement et d'avoir des capacités de perception adaptées pour ce

3. Assumption-based Truth Maintenance System

type d'environnements. Un autre aspect qui mérite d'être exploré concerne l'utilisation des niveaux de contrôle hiérarchiques de l'architecture comme des méta-niveaux pour l'apprentissage dans un environnement multi-robot. Dans ce cadre, si un robot se retrouve dans un état indéterministe, il peut interagir avec ses superviseurs afin de décider des actions à exécuter et passer dans un état déterministe. L'objectif est d'utiliser l'architecture comme cadre pour un modèle d'apprentissage des agents à plusieurs niveaux d'abstraction.

Chapitre 5

Implémentation et exemples d'instanciation

Préambule:

L'objectif de ce chapitre est de présenter les aspects implémentation de notre architecture à travers le développement d'une plate-forme pour prototyper des systèmes multi-agents. Le chapitre décrit la manière dont la plate-forme est implémentée en IC-Prolog et fournit une batterie d'exemples simples pour illustrer l'instanciation du système MAGIQUE. L'idée sur laquelle repose ces exemples s'articule autour de la coopération d'un ensemble de moteurs d'inférence via des agents superviseurs caractérisés par une panoplie de capacités de communication.

Ainsi, un aperçu bref du langage IC-Prolog est donné, suivi par une description du noyau du langage objet que nous avons implémenté. Des détails d'implémentation de certaines primitives importantes et inhérentes à la plate-forme sont aussi fournis. Subséquemment, un premier exemple montre un système d'inférence propositionnel réparti dont la connaissance du domaine d'expertise est répartie exclusivement entre les agents spécialistes représentés au niveau des feuilles de la hiérarchie.

Le second exemple illustre un moteur d'inférence réparti où tous les agents du système (spécialistes ou superviseurs) sont dotés de capacités d'inférence.

Le dernier exemple introduit la notion de raisonnement flou au niveau des systèmes d'inférence. Ainsi, plusieurs types de moteurs d'inférence (moteur d'inférence en chaînage avant, moteur d'inférence en chaînage arrière et moteur d'inférence en logique flou) coopèrent mutuellement. Les superviseurs coordonnent les interactions et calculent les coefficients de vraisemblance des nouvelles informations inférées.

5.1 Introduction

L'une des idées ayant poussé les experts en IA à adopter le paradigme multi-agent comme une nouvelle approche de conception et de développement de systèmes réside au niveau du besoin de faire coopérer des systèmes experts déjà existants. Ainsi, malgré le succès engendré par certains systèmes experts dans les milieux industriels [Sho76][BS84], l'approche centralisé s'est révèlée insuffisante pour développer des sytèmes complexes, performants et robustes. Le paradigme multi-agent constitue l'artefact le plus approprié pour répondre à l'exigence de la communication entre les multiples systèmes experts existants dont les fonctionnalités leur sont propres, et dont l'architecture n'est pas conçue de telle façon à prévoir une coopération ultérieure avec d'autres composants logiciels similaires. La naissance de cette nouvelle approche s'est manifestée par la conception et le développement de plusieurs environnements pour supporter des applications multi-agents dont les plus connus sont exposés dans [CC94][Chu92][FB88]. Alors que le langage APRIL [1] (voir la section B.5), orienté vers l'implémentation de systèmes multi-agents, manipule uniquement des processus Unix, le langage IC-Prolog est caractérisé par l'aspect multi-threadé qui lui confère une nette efficacité par rapport aux environnements à base de processus lourds.
Vu l'aspect déclaratif, la puissance d'unification, ainsi que les capacités d'inférence qui caractérisent le langage Prolog, nous avons proposé une plate-forme implémentée en IC-Prolog. L'objectif visé est de fournir un support de programmation permettant de prototyper facilement et rapidement un système multi-agent. La plate-forme tire profit de tous les avantages d'un langage d'IA, des aspects multi-threadé nécessaires pour une communication asynchrone et transparente entre agents distribués, ainsi que les aspects objets comme la modularité, l'encapsulation et l'héritage qui sont d'un grand intérêt pour l'implémentation des systèmes multi-agents. Le choix d'un modèle de base méta-circulaire réflexif permet de créer de manière triviale un nombre quelconque de niveaux de contrôle hiérarchiques. La classe *meta_super* est une méta-classe qui se charge d'instancier des classes dont le but est de créer des agents superviseurs.
Ainsi, le langage conçu supporte des objets concurrents caractérisés par leur boîte aux lettres. La boîte aux lettres est gèrée par un thread dont la tâche consiste à détecter l'arrivée d'un message asynchrone, l'exécuter en créant un nouveau thread pour l'exécution, et enfin se bloquer à nouveau derrière sa boîte aux lettres pour attendre le prochain message à traiter. Le noyau offre un haut niveau d'abstraction en matière de communication entre les agents 5.1. Un agent n'a pas besoin de savoir où sont logés les autres agents accointances pour commniquer avec eux. L'envoi de message se fait normalement comme dans le cas des langages objets, sauf que le noyau se chargera d'identifier la location physique de l'agent avant d'effectuer une requête au serveur de boite aux lettres qui se chargera d'acheminer le message vers l'agent destinataire. Le serveur de boîte aux lettres est une entité centralisée qui gère l'ensemble des boîtes aux lettres créées ainsi que les opérations de communication de base entre les threads

1. APRIL fait partie du programme Européen IMAGINE- projet ESPRIT

[CC93]. Par conséquent, il dispose d'une vision globale de l'ensemble des boîtes aux lettres créées ainsi que des threads qui les gèrent, et offre une transparence en terme de location des objets sur les différents nœuds du réseau.

5.2 Le langage IC-Prolog

IC-Prolog est un nouveau système développé à l'Imperial College. Il résulte du projet IMAGINE dont le but est de construire des systèmes multi-agents dans lesquels une collection d'agents "solvers" semi-autonomes coopèrent et coordonnent leurs actions afin de résoudre des problèmes ou d'atteindre un but commun. Les agents sont distribués sur des machines différentes reliées par un réseau. IC-Prolog est caractérisé par un certain nombre de propriétés qui ne sont pas disponibles dans les systèmes Prolog standards. Parmi ces propriétés, nous pouvons citer la manipulation de threads multiples, les primitives de communication de haut niveau et une extension orienté objet. Toutes ces caractéristiques font que de nouveaux domaines d'application peuvent tirer profit des avantages de la programmation logique. À cet égard, les systèmes basés sur la connaissance distribuée, les systèmes experts coopératifs et les systèmes multi-agents sont des applications pouvant être implémentées facilement à base du langage IC-Prolog. Comme ce dernier est destiné au prototypage des systèmes multi-agents utilisant la programmation logique, il est clair que les aspects communication et concurrence sont importants.
Ainsi, la «concurrence» est la capacité de manipuler plusieurs problèmes simultanément. Par exemple, il est indésirable de monopoliser un serveur de bases de données pour traiter une requête complexe alors que plusieurs requêtes simples sont en attente pour être traitées. À cet égard, un seul thread de contrôle est insuffisant, d'où le recours au parallèlisme.
La «communication» repose sur la capacité d'envoyer ou de recevoir des messages entre threads et au travers d'un réseau, afin d'échanger des données entre agents. Le langage fournit des *pipes* pour permettre la communication entre threads locaux. Le modèle de communication est basé sur la notion de boîte aux lettres (*mailboxes*) qui garantit une communication asynchrone entre agents.
En ce qui concerne l'aspect multi-threadé, et à la différence des systèmes Prolog traditionnels qui ont un seul thread de contrôle ne permettant guère une exécution concurrente, IC-Prolog permet à plusieurs programmes indépendants de s'exécuter en pseudo-parallèle en utilisant de multiples threads [CMG82]. Pour des raisons d'efficacité, tous les threads s'exécutent dans un seul processus Unix dont on a besoin pour effectuer son propre scheduling. Chaque thread contient son propre contexte rangé dans une pile, un ensemble de registres WAM[2] et d'autres informations liées à la gestion du thread.
Pour gérer les threads, IC-Prolog fournit trois primitives *fork, suspend* et *resume*. La primitive *fork* permet de créer un nouveau thread, la primitive *suspend* permet de suspendre un thread actif et la primitive *resume* active un thread bloqué.
Le modèle de communication entre agents distribués sur un réseau de stations est basé sur le concept de boîte aux lettres. IC-Prolog fournit trois primitives pour gérer une communication à base de boîte aux lettres. Avant d'envoyer un message à son accointance, l'agent émetteur doit connaître l'adresse de la boîte aux lettres de l'agent destinataire. Pour créer une boîte aux lettres, IC-Prolog fournit la primitive $mbx\_create(-Id)$ qui retourne un numéro identificateur de la boîte aux lettres. Il fournit aussi un outil permettant d'associer une boîte aux

2. Warren Abstract Machine

lettres à un agent au travers de la primitive $mbx\_bind(+Id, +Nom\_agent)$. Les primitives permettant de déposer (respectivement de retirer) un message dans (respectivement de) la boîte aux lettres sont:

$mbx\_send(+Id, +Message)$
$mbx\_receiv(+Id, -Message)$

Il existe aussi une primitive dont le rôle consiste à récupérer l'adresse de la boîte aux lettres d'un agent donné. La syntaxe est la suivante:

$mbx\_getid(+Agent, -Id)$

5.3 Noyau de la plate-forme

La distribution des agents sur un réseau répond à un double souci , celui de prendre en compte les aspects distribués qui sont inhérents aux systèmes multi-agents et un autre concerne l'efficacité en permettant aux agents d'une application de s'exécuter en parallèle sur des machines différentes et de communiquer d'une manière asynchrone. Le code présenté dans cette section illustre l'implantation d'un noyau de notre plate-forme distribuée à base d'un modèle de communication basé sur la notion de boîte aux lettres. Il décrit comment les notions de base inhérentes aux langages objets, notamment l'envoi de messages et les méthodes de création d'objets et de classes, sont implantées en IC-Prolog. Le mécanisme d'envoi de messages est représenté par le symbole ::. La définition de la liste des attributs et de celle des méthodes se fait par le biais d'attributs caractérisant la classe *object*, et appelés respectivement **att** et **meth**.

Dans cette section, nous présenterons les détails d'implémentation de la méthode *new* du modèle de base de notre plate-forme afin qu'elle soit invoquable à partir d'un site quelconque du réseau. L'objectif visé est de permettre à deux objets quelconques du système, localisés sur des sites différents de s'envoyer mutuellement des messages sans se soucier de la location physique d'un objet par rapport à l'autre. Le noyau de base constitue ainsi une couche d'abstraction transparente qui fournit un modèle de communication entre agents répartis sur des sites différents d'un réseau de stations Unix (voir Fig.5.1).

```
/*************************************
Noyau de base du
langage orienté objets
*************************************/
:- set_path('/home/smac/bensaid/these/sources/magique_dist').

:- ensure_loaded(mailbox). /* chargement du module de gestion des boîtes aux
                              lettres */

:- [util]. /* chargement du fichier contenant des prédicats utiles */
:- dynamic obj/3 .
:- retractall(obj(_,_,_)).
```

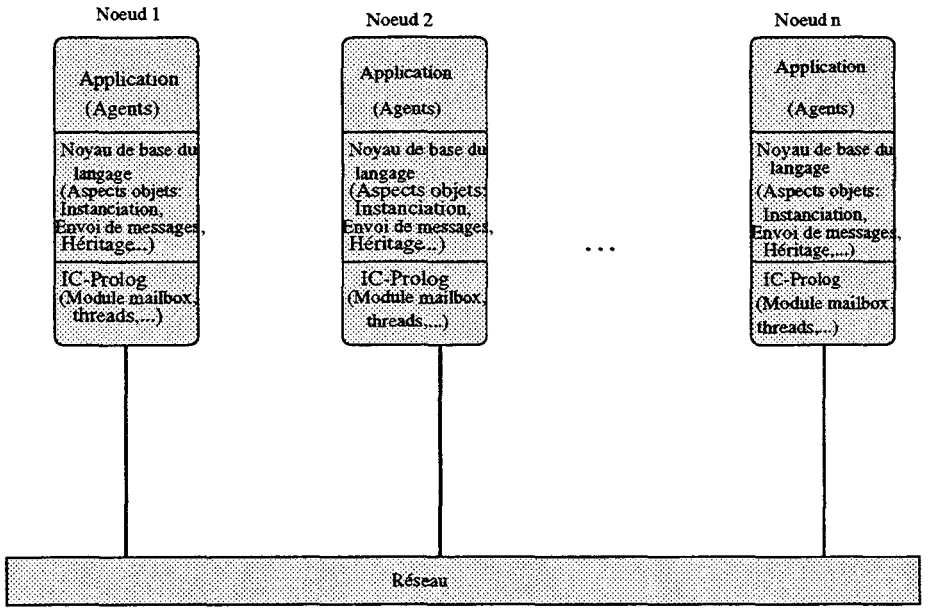

FIG. 5.1 - *Le modèle du langage Objet*

```
:- op(800,xfx,'::').
:- op(750,xfx,as).
```

La méthode new

```
'::'(Object,Message) :- obj(Object,_,_),Message='new'(Obj),
                        mbx_create(Mbx),mbx_bind(Mbx,Obj),
                        basicnew(Object,Obj),
                        fork(retirer_mess(Mbx)),!.

'::'(Object,Message) :- Message='new'(Obj),
                        mbx_getid(Object,_),
                        basicnew(Object,Obj),
                        mbx_create(Mbx),mbx_bind(Mbx,Obj),
                        fork(retirer_mess(Mbx)),!.

'::'(Object,Message) :- obj(Object,_,_),Message='new'(Obj,sc,Classe),
                        obj(Classe,path,Path),add_item(Obj,Path,Npath),
                        basicnew(Object,Obj,sc,Classe,Npath),
                        mbx_create(Mbx),mbx_bind(Mbx,Obj),
                        fork(retirer_mess(Mbx)),!.

'::'(Object,Message) :- obj(Object,_,_),Message='new'(Obj,sc,Classe),
                        \+ obj(Classe,_,_),
                        mbx_getid(Classe,Adr),mbx_create(Bal),
                        mbx_send(Adr,rech_path(Classe,Bal)),
```

```
                        mbx_recv(Bal,L),mbx_close(Bal),
                        add_item(Obj,L,Path),
                        basicnew(Object,Obj,sc,Classe,Path),
                        mbx_create(Mbx),
                        mbx_bind(Mbx,Obj),
                        fork(retirer_mess(Mbx)),!.

'::'(Object,Message) :- Message='new'(Obj,sc,Classe),
                        mbx_getid(Object,_),
                        obj(Classe,path,Path),
                        add_item(Obj,Path,Npath),
                        basicnew(Object,Obj,sc,Classe,Npath),
                        mbx_create(Mbx),mbx_bind(Mbx,Obj),
                        fork(retirer_mess(Mbx)),!.

'::'(Object,Message) :- Message='new'(Obj,sc,Classe),
                        mbx_getid(Object,_),
                        \+ obj(Classe,_,_),
                        mbx_getid(Classe,Adr),
                        mbx_create(Bal),
                        mbx_send(Adr,rech_path(Classe,Bal)),
                        mbx_recv(Bal,L),
                        mbx_close(Bal),
                        add_item(Obj,L,Path),
                        basicnew(Object,Obj,sc,Classe,Path),
                        mbx_create(Mbx),
                        mbx_bind(Mbx,Obj),
                        fork(retirer_mess(Mbx)),!.
```

La méthode **new** consiste à créer soit un objet terminal, représentant d'une classe, soit une classe instance d'une méta-classe. Dans le premier cas, si l'objet destinataire est local alors il y a création de l'objet, suivie d'une création d'une boîte aux lettres associée au nom de l'objet créé, ensuite d'une création de thread bloqué derrière la boîte aux lettres créée. Le thread créé va attendre le prochain message à exécuter. Dans le second cas (l'objet destinataire du message est logé sur une machine distante), il y a d'abord vérification de l'existence de l'objet destinataire, puis la procédure est la même que dans le cas précédent. Les attributs d'un objet qui constituent sa partie statique sont rangés dans la data-base du système d'IC-Prolog. Ainsi, le prédicat **basic_new** permet de créer les différents attributs d'un objet terminal ou d'un objet classe. Concrétement, la création des attributs d'un objet revient à insérer ces attributs dans la base de données. Par exemple, la création effective des attributs d'un objet terminal se fait par le prédicat suivant:

```
basicnew(Self,Obj) :- \+ obj(Obj,_,_),
                       assert(obj(Obj,isa,Self)).
```

où **Self** est l'objet destinataire et **Obj** est l'objet à créer. Pour la création d'une classe, le prédicat **basicnew** a 5 paramètres.

```
basicnew(Self,Obj,sc,Classe,Path) :-  \+ obj(Obj,_,_),
                                      assert(obj(Obj,sc,Classe)),
                                      assert(obj(Obj,isa,Self)),
                                      assert(obj(Obj,path,Path)).
```

En ce qui concerne la création des classes, le processus est moins évident par rapport à celui de la création des objets terminaux. Au total, il existe quatre cas liés à la localité ou à la non localité de l'objet destinataire et de la classe héritée. Les objets qui sont des classes sont caractérisés par un attribut *Path* contenant le chemin d'héritage de cette classe. Ainsi, les cas les plus importants concernent la classe héritée car le chemin d'héritage de la classe à créer dépend de celui de la classe héritée. Par conséquent, on doit consulter l'attribut *Path* de la classe héritée afin d'instancier celui de la classe à créer. Le reste du processus est similaire à celui de la création des objets terminaux. Le prédicat *retirer_mess(Mbx)* permet de retirer un message de la boîte aux lettres *Mbx* et de l'exécuter, avant de passer à l'exécution du prochain.

5.4 Gestion de l'héritage

L'héritage est l'une des propriétés les plus importantes des langages à objets. Néanmoins, sa mise en œuvre dans un environnement de programmation distribué n'est pas toujours aisée du point de vue de l'implémentation. La réalisation de l'héritage a nécessité l'ajout d'un nouvel attribut à la classe *object* afin de prendre en compte cet aspect. Cet attribut porte le nom *Path* et est implémenté sous forme d'une liste contenant le chemin d'héritage de chaque classe. À la création d'une nouvelle classe, il y a mise à jour de l'attribut *Path*. Et, si l'objet correspondant à la classe à créer se trouve dans un site distinct de celui où est créée la classe héritée, il y a création d'une boîte aux lettres afin de permettre une communication entre la classe créée et la classe à hériter. Exemple:

Objet::new(Obj,sc,[Classe1,Classe2,...,Classen])

où **Classe1,Classe2,...,Classen** sont des classes.

La communication consiste pour l'objet créé **Obj** à demander à la classe héritée (**Classe**) de lui transmettre le chemin d'héritage. Une fois que la classe créée ait reçu le chemin d'héritage de la classe **Classe**, elle rajoute à ce chemin le nom de sa classe d'héritage qui fait partie désormais de son chemin d'héritage et affecte la liste résultante à son attribut *Path*. La boîte aux lettres qui a été créée pour la communication est fermée par le prédicat **mbx_close(Mbx)** où *Mbx* est l'adresse physique de la boîte aux lettres.
L'héritage multiple est supporté par le langage que nous avons développé. En effet, une classe peut hériter de toutes les propriétés d'un ensemble d'autres classes. Ainsi, pour déterminer les chemins d'héritage d'une classe qui hérite d'un ensemble de classes, il suffit de déterminer le chemin d'héritage engendré par chacune des classes héritées. Finalement, l'attribut *Path* d'une classe qui hérite d'un ensemble de classes est représenté par une liste contenant des éléments sous forme de listes représentant les chemins d'héritage possibles.

Vu que l'opération de création d'un objet (objet terminal ou classe) met en œuvre plusiers communications entre les différents objets qui sont invoqués (objet destinataire, classes héritées,...), il y a nécessairement des retombées négatives sur le plan de l'efficacité du système. Dans le cas par exemple où l'on veut créer une classe qui hérite de deux autres classes créées sur des sites distincts, il y aura au moins 4 communications, correspondant à l'envoi de 4 messages, entre l'objet créé et les classes héritées afin de mettre à jour l'attribut *Path* de la classe créée. Par exemple, l'instruction suivante consiste à envoyer à la classe *Meta_super* le message *new* afin de créer une nouvelle classe appelée *Classe* qui puisse hériter des classes *Classe1* et *Classe2*.

Meta_super::new(Classe,sc,[Classe1,Classe2])

Soient *Path1=[Cl11,...,Cl1n]* et *Path2=[Cl21,...,Cl2m]*. L'attribut *Path* de la nouvelle classe créée aura le contenu suivant:

Path=[[Classe1,Cl11,...,Cl1n],[Classe2,Cl21,...,Cl2m]]

La plate-forme que nous avons développée fournit une couche logique d'abstraction, où une application est considérée comme un ensemble de threads concurrents et répartis sur les nœuds du réseau.

5.5 Implémentation

MAGIQUE est opérationnel sur un réseau de stations Unix. Nous avons utilisé un langage objet à base d'IC-Prolog [Chu93] pour l'implémentation du système. La plate-forme est caractérisée par un modèle de communication basé sur la boîte aux lettres permettant ainsi une communication asynchrone entre les agents qui sont physiquement distribués à travers un ensemble de stations. Ainsi, chaque agent est caractérisé par un thread qui gère la boîte aux lettres.

Le modèle de base est similaire au modèle OBJVlisp [BC86]. Il contient les classes suivantes: *object, class* et *meta_super*. Le choix de ce modèle obéit à des considérations de facilité d'instanciation d'un nombre quelconque de superviseurs représentant les différents niveaux de contrôle. La notion d'envoi de message a été étendue afin qu'elle soit adaptée à un contexte distribué. En effet, si le destinataire du message est un agent local, l'envoi de messages se réduit à un appel procédural. Si le destinataire est un agent localisé sur une autre machine, l'opération d'envoi de message consiste à récupérer l'adresse de la boîte aux lettres à partir du serveur de boîte aux lettres, ensuite à déposer un message dans cette boîte aux lettres. Le dépot d'un message dans une boite aux lettres se fait d'une manière asynchrone et engendre le déclenchement d'un thread, à priori, bloqué derrière cette boite aux lettres. Le déclenchement du thread a pour but d'exécuter le message adressé à l'agent.

La classe *object* (voir figure 5.2) est la classe de tous les objets de la réalité. Par conséquent, elle a deux méthodes qui sont respectivement *ask* et give. La première méthode permet de consulter l'attribut de l'objet, tandis que la méthode *give* a pour rôle d'affecter une valeur à un attribut de l'objet. La classe class est la classe d'instanciation de toutes les classes de la réalité. Elle possède la méthode *new* qui lui permet de créer de nouvelles classes. Il faut noter que la classe *object* est une instance de la classe *class*. En revanche, cette dernière est une sous

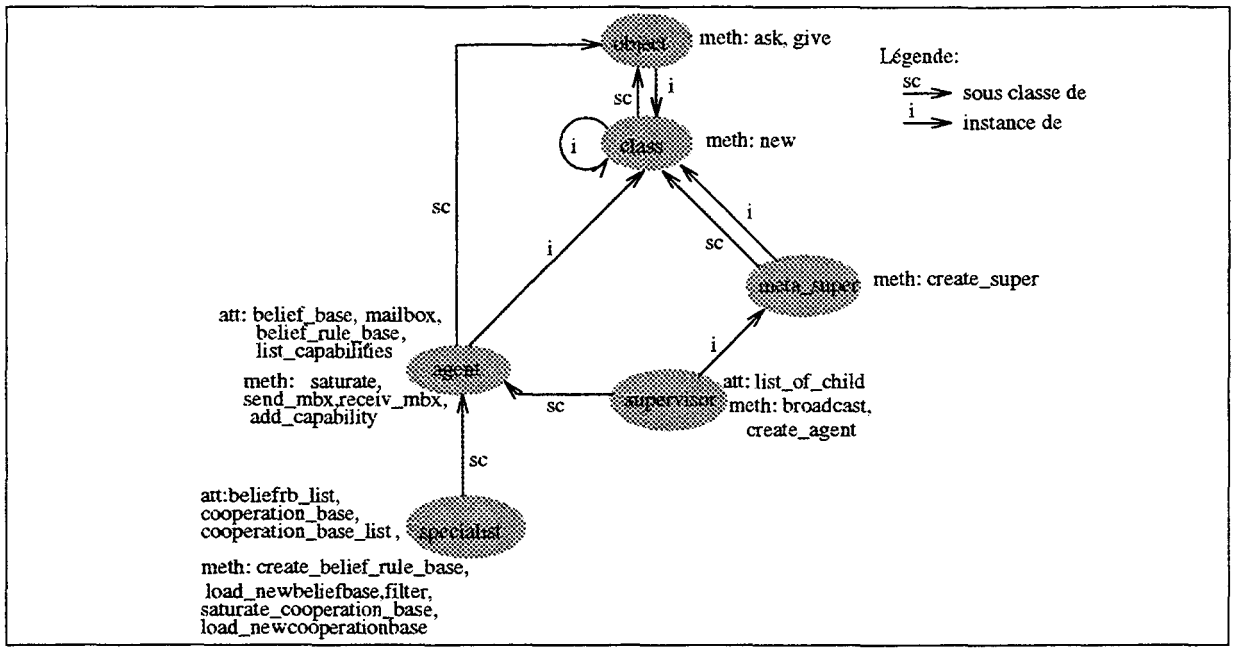

FIG. 5.2 - *Modélisation du système*

classe de la classe *object*. L'implémentation d'un prototype dans un langage objet à base d'IC-Prolog a permis de tester le modèle sur un ensemble d'agents d'inférence coopératifs répartis sur un ensemble de stations [BM97c] [BM95] (voir la section 5.9). Le modèle est aussi caractérisé par ses méthodes de communication récursives qui permettent de diffuser un message aux agents spécialistes (feuilles de la hiérarchie) seulement, ou vers les agents du groupe ou encore destiné à tous les agents de la hiérarchie. Une application est implémentée sous forme d'un ensemble d'agents qui coopèrent mutuellement. Les capacités, les méta-capacités et les bases de règles des croyances des agents sont spécifiées au moment de l'implémentation car elles dépendent du domaine de l'application. Les détails d'implémentation du langage objet ainsi que les méthodes de communication de MAGIQUE sont présentées dans la section 5.3.

5.6 Le noyau de MAGIQUE

Dans cette section, les détails d'implémentation des méthodes les plus importantes, particulièrement celles permettant de créer un agent spécialiste ou un agent superviseur. Les deux classes *specialist* et *supervisor* héritent de la même classe *agent* toutes les propriétées qui leurs sont communes.

La classe meta_super est une méta-classe car elle hérite de la classe *class* la méthode *new* lui permettant d'instancier des classes. Elle est caractérisée par sa méthode *create_super* dont le rôle est d'initialiser l'attribut *list_of_child* de l'agent superviseur.

La classe *specialist* permet d'instancier des agents spécialistes (voir Fig.5.2). Elle est instance de la classe *class* et est sous classe de la classe *Agent*. Elle est caractérisée par les attributs suivants:

belief_base: cet attribut représente la base des croyances de l'agent spécialiste.

mailbox: cet attribut représente la boîte aux lettres de l'agent. Tous les messages destinés à l'agent sont stockés dans la boîte aux lettres.
beliefrb_list: Cet attribut représente la liste de base des règles de croyances.
belief_rule_base: cet attribut représente la base de règles des croyances courante.
En outre, le spécialiste est caractérisé par les méthodes suivantes:
filter: cet attribut permet de sélectionner à chaque cycle de l'agent, la base de règles des croyances la plus adaptée et la charge pour qu'elle soit la base des règles des croyances courante.
cooperation_base: cet attribut est une liste contenant l'ensemble des règles de coopération. À chaque fois qu'une règle de cette base est satisfaite, il y a envoi d'une croyance à un superviseur de groupe. La croyance envoyée est rajoutée à la base des croyances du superviseur.
cooperation_base_list: cet attribut contient la liste des bases de coopération de l'agent.
saturate_coopération_base: est un programme qui consiste à saturer la base de coopération du spécialiste.
list_capabilities: cet attribut renferme les noms des différentes capacités qui caractérisent l'agent. *add_capability:* cette méthode permet de rajouter une capacité à la liste des capacités qui caractérisent l'agent.
saturate: cette méthode a pour rôle de saturer la base de règles des croyances afin de générer de nouvelles croyances.
saturate_cooperationbase: cette méthode est aussi une action interne. Elle permet de saturer la base de coopération de l'agent.
load_newbeliefbase: cette méthode charge une nouvelle base de règles des croyances à la place de l'ancienne base de règles.
load_newcooperationbase: cette méthode est une action interne car le résultat de son exécution n'est pas visible des autres agents. Elle permet de charger la base de coppération correspondante à la nouvelle base de règles des croyances.
create_belief_rule_base: c'est une méthode qui permet à l'utilisateur de créer une nouvelle base de règles de croyances et de la ranger dans la liste *beliefrb_list.*
send_mbx: c'est une primitive dont le rôle est de déposer un message dans la boîte aux lettres d'un autre agent. Le dépot du message se fait d'une manière asynchrone.
receiv_mbx: c'est une primitive qui consiste à retirer un message de la boîte aux lettres de l'agent.

La classe *supervisor* a pour rôle l'instanciation des agents superviseurs. Elle est sous-classe de la classe *agent.* Elle est caractérisée par son attribut *list_of_child* qui est une liste contenant les noms des agents du groupe. Elle renferme deux méthodes appelées respectivement *broadcast* et *create_agent.* La première méthode permet de diffuser un message à un groupe d'agents, ou encore à tous les agents de la hiérarchie, ou encore aux spécialistes seulement. La seconde méthode permet au superviseur de créer un nouvel agent spécialiste.

La méthode create_super

```
'::'(Object,Message) :- obj(Object,_,_),
                        Message='create_super'(Superviseur,null),
                        basicnew(superviseur,Superviseur),
                        mbx_create(Mbx),mbx_bind(Mbx,Superviseur),
                        Superviseur::affect(list_of_child,□),
```

```
                        Superviseur::affect(super,null),
                        fork(retirer_mess(Mbx)),!.

'::'(Object,Message) :- Message='create_super'(Superviseur,null),
                        mbx_getid(Object,_),
                        basicnew(superviseur,Superviseur),
                        mbx_create(Mbx),
                        mbx_bind(Mbx,Superviseur),
                        Superviseur::affect(list_of_child,[]),
                        Superviseur::affect(super,null),
                        fork(retirer_mess(Mbx)),!.

'::'(Object,Message) :- obj(Object,_,_),
                        Message='create_super'(Superviseur,S1),
                        basicnew(superviseur,Superviseur),
                        mbx_create(Mbx),mbx_bind(Mbx,Superviseur),
                        Superviseur::affect(super,S1),
                        Superviseur::affect(list_of_child,[]),
                        S1::add_fils(Superviseur),
                        fork(retirer_mess(Mbx)),!.

'::'(Object,Message) :- Message='create_super'(Superviseur,S1),
                        mbx_getid(Object,_),
                        basicnew(superviseur,Superviseur),
                        mbx_create(Mbx),
                        mbx_bind(Mbx,Superviseur),
                        Superviseur::affect(super,S1),
                        Superviseur::affect(list_of_child,[]),
                        S1::add_fils(Superviseur),
                        fork(retirer_mess(Mbx)),!.
```

De même, l'implémentation du message *create_super*, qui consiste à créer un agent superviseur, est similaire à celle du message *new* décrit précédement. Aprés initialisation des attributs de l'objet, il y a création d'une boîte aux lettres qui sera associée au nom du superviseur créé. Le programme initialise l'attribut *list_of_child* par la liste vide, avant de créer un thread qui se bloquera derrière la boîte aux lettres de l'objet. Dans le cas où la boîte aux lettres est vide, le prédicat *retirer_mess (Mbx)* est bloquant. Dans le cas contraire, tant qu'il existe un message dans la boîte aux lettres, le thread retire le message de celle-ci, l'exécute, ensuite retire le message suivant. Le prédicat *create_super* a deux arguments: le superviseur à créer et l'agent superviseur du groupe auquel appartient l'agent créé. L'objet auquel est destiné le message est nécessairement une classe qui soit un représentant directe ou indirecte de la classe meta_super.

La méthode create_agent

```
'::'(Object,Message) :- obj(Object,_,_),
                        Message='create_agent'(Agent,Classe),
                        mbx_getid(Classe,_),
                        basicnew(Classe,Agent),
                        mbx_create(Mbx),mbx_bind(Mbx,Agent),
                        Object::add_fils(Agent),
                        Agent::affect(super,Object),
                        fork(retirer_mess(Mbx)),!.

'::'(Object,Message) :- Message='create_agent'(Agent,Classe),
                        mbx_getid(Object,_),
                        mbx_getid(Classe,_),
                        basicnew(Classe,Agent),
                        mbx_create(Mbx),mbx_bind(Mbx,Agent),
                        Object::add_fils(Agent),
                        Agent::affect(super,Object),
                        fork(retirer_mess(Mbx)),!.
```

La méthode *create_agent* permet de créer des agents spécialistes. Elle contient deux arguments: l'identificateur de l'objet à créer et la classe d'instanciation. L'objet auquel est destiné le message est nécessairement un agent superviseur.

5.7 Le système de communication

Alors que le langage utilisé pour prototyper MAGIQUE garantit l'envoi de message asynchrone entre agents distribués sur un ensemble de sites, il existe un système de communication basé sur un ensemble de primitives de communication qui caractérisent l'architecture. Le modèle de base de MAGIQUE contient en plus des classes qui caractérisent le langage de base, à savoir *object* et *class*, 3 autres classes caractérisant le modèle MAGIQUE. Il s'agit des classes *specialiste, superviseur* et *meta_super* (voir section 5.5). Dans ce cadre, une couche de communication vient se coupler au modèle de base de MAGIQUE. Elle permet à un superviseur d'avoir la capacité à diffuser un message à tous les agents (spécialistes ou superviseurs) qui font partie de l'arborescence dont il est la racine. Il a aussi la possibilité de diffuser un message seulement aux agents qui sont de niveau immédiatement inférieur. Enfin, il a possibilité de diffuser un message aux agents spécialistes représentés au niveau des feuilles de l'arborescence dont il est la racine. La méthode de diffusion est appelée *broadcast(Type, Message)*.
Elle possède deux arguments: le type qui doit appartenir à l'ensemble (`feuilles, tous, fils`) et le message à diffuser. Par exemple, à chaque fois qu'un superviseur reçoit le message `broadcast(feuilles, Message)`, il le rediffuse récursivement vers les agents de son groupe jusqu'à ce qu'il soit reçu par les spécialistes.
`broadcast(tous, Message)` : tous les agents qui reçoivent le message l'exécutent. Si l'agent recevant le message est un superviseur, le message est exécuté, ensuite rediffusé vers les agents du groupe.
`broadcast(fils, Message)` : seuls les agents membres du groupe vont exécuter le message.

```
:- ['~bensaid/these/sources/noyau.pl'].

broadcast(Self,fils,Message):- Self::ask(list_of_child,L),
                        deposer_list_fils(L,Message).

broadcast(Self,tous,Message):- Self::ask(list_of_child,L),
                        deposer_list_tous(L,Message).

broadcast(Self,feuilles,Message) :- Self::ask(list_of_child,L),
                        deposer_list_feuilles(L,Message).

deposer_list_tous([],_).
deposer_list_tous([X|L],Message):-
                        X::ask(list_of_child,_),!,
                        mbx_getid(X,Ptr),
                        mbx_send(Ptr,X::Message),
                        mbx_send(Ptr,X::broadcast(tous,Message)),
                        deposer_list_tous(L,Message).

deposer_list_tous([X|L],Message) :- mbx_getid(X,Ptr),
                        mbx_send(Ptr,X::Message),
                        deposer_list_tous(L,Message).

deposer_list_feuilles([],_).
deposer_list_feuilles([X|L],Message):-
                        X::ask(list_of_child,_),!,
                        mbx_getid(X,Ptr),
                        mbx_send(Ptr,X::broadcast(feuilles,Message)),
                        deposer_list_feuilles(L,Message).


deposer_list_feuilles([X|L],Message) :-
                        mbx_getid(X,Ptr),
                        mbx_send(Ptr,X::Message),
                        deposer_list_feuilles(L,Message).


deposer_list_fils([],_).
deposer_list_fils([X|L],M) :- mbx_getid(X,Ptr),
                        mbx_send(Ptr,X::Message),
                        deposer_list_fils(L,M).
```

Cette capacité de communication dont dispose les agents superviseurs constitue un outil privilégié pour soumettre des requêtes envoyées sous forme de messages aux autres agents de la

hiérarchie. La capacité de communication *diffuser* est indépendante du nombre de niveaux de contrôle. Ainsi, un superviseur quelconque (quelque soit son niveau hiérarchique) est capable de diffuser un message à tous les agents de sa hiérarchie.

5.8 Utilisation de MAGIQUE

L'utilisation de la plate-forme MAGIQUE consiste à charger le package **MAGIQUE** dans le fichier contenant le code de l'application. Ensuite, tous les services supportés par l'architecture peuvent être invoqués. Cependant, l'utilisateur doit avoir installé IC-Prolog au préalable. La création des agents consiste à faire des appels aux méthodes **create_agent** et **create_super** afin d'instancier la classe **specialist** ou **supervisor**. Ces deux méthodes nécessitent des paramètres tels que le nom de l'agent, sa classe d'instanciation, ainsi que le chemin d'héritage dans le cas oòu l'objet créé est une classe. La création d'un objet ou d'une classe se fait par défaut sur la machine sur laquelle travaille l'utilisateur. La création d'un agent sur une machine donnée du réseau nécessite le lancement d'une session *IC-Prolog-MAGIQUE* sur cette machine, avant de lancer l'opération de création de l'agent. Le fichier contenant l'application doit lancer le noyau de la plate-forme *noyau.pl*, ensuite le noyau de l'architecture *magique.pl*. Les classes *object, class, meta_super, agent, specialist* et *supervisor* sont instanciées au moment du lancement d'une session MAGIQUE sur une machine donnée. La création des agents liés à une application appelée à être supportée par MAGIQUE consiste à instancier les classes *specialist* et *supervisor*. De même que la définition des capacités d'un agent de l'application se fait par le biais de la méthode **add_capability**. Le modèle de coopération entre les spécialistes et les superviseurs doit être aussi spécifié par l'utilisateur qui définit en fonction de son application la base de coopération de l'agent spécialiste, ainsi que les configurations particulières de la base des croyances de l'agent qui permettent de déclencher les actes communicatifs entre les agents spécialistes. La méthode **create_belief_rule_base** permet de définir une nouvelle base de règles des croyances et de la ranger dans la liste **beliefrb_list**. Le filtre de sélection des bases de règles de croyances est un programme qui dépend aussi de l'application. Par conséquent, il doit être spécifié par l'utilisateur. La base de coopération renferme des règles de coopération dont le déclenchement entraine l'envoi d'une croyance au superviseur. Elle est aussi spécifiée par l'utilisateur.

5.9 Application: un système d'inférence réparti

Parmi les applications que nous avons développées sous MAGIQUE, un système d'inférence réparti composé d'un ensemble d'agents dotés de capacités d'inférence. Les systèmes d'inférence répartis sont largement utilisés dans plusieurs domaines comme l'industrie, les systèmes multi-experts appliqués au diagnostique médical, etc...
Ils ont comme but de supporter la coopération entre un ensemble d'agents. Ainsi, dans notre cas, les agents utilisent pleinement les primitives de communication supportées par MAGIQUE [BM97c] [BM95]. Le prototype d'application utilise un système expert propositionnel pour illustrer l'instanciation du modèle. Chaque agent possède sa propre base de règles des croyances `belief_rule_base` et sa propre base des croyances `belief_base` (ces deux bases sont représentées sous forme d'attributs de l'agent `specialist`) et un mécanisme d'inférence invoqué au travers de la méthode `saturate`. Cette dernière consulte les attributs *belief_base* et *belief_rule_base* récupèrent leur contenu dans des listes, puis applique la base des règles en

chînage avant. Le résultat de la saturation est affecté à l'attribut *belief_base*. Le schéma figure. 5.3 montre une hiérarchie d'agents pour lesquels la connaissance est représentée seulement au niveau des spécialistes (feuilles de la hiérarchie). Le modèle supporte un mécanisme d'inférence distribué basé sur la diffusion d'un message `saturate` à tous les agents de la hiérarchie ou seulement aux feuilles de la hiérarchie représentées par les spécialistes.

Création des agents.

```
| ?- supervisor::create_super(sup_global,null).
```

Ensuite, deux superviseurs `sup_1` et `sup_2`, fils de `sup_global`, sont créés.

```
| ?- supervisor::create_super(sup_1,s).
| ?- supervisor::create_super(sup_2,s).
```

`sup_1` gère un groupe composé des spécialistes `spec_1` et `spec_2`.

```
| ?- sup_1::create_agent(spec_1,specialist).
| ?- sup_1::create_agent(spec_2,specialist).
```

`sup_2` gère seulement le spécialiste `spec_3`.

```
| ?- sup_2::create_agent(spec_3,specialist).
```

Initialisation des spécialistes.

```
| ?- spec_1::give(belief_rule_base,[if a then x,if b and c then e]).
| ?- spec_2::give(belief_rule_base,[if b then y,if a and c then f]).
| ?- spec_3::give(belief_rule_base,[if c then z,if a and b then g]).
```

5.9.1 Exemple d'exécution

Dans l'exécution suivante, le superviseur `sup_global` dépose le message `broadcast(feuilles, saturate)` dans la boîte aux lettres des agents `sup_1` et `sup_2`. Ces superviseurs déposent à leur tour le message `saturate` dans la boîte aux lettres des spécialistes `spec_1`, `spec_2` et `spec_3` qui saturent respectivement leur base de règles des croyances et déduisent les faits `x` et `e` pour `spec_1`, `y` et `f` pour `spec_2` et `z`, `g` pour `spec_3`.
Il est possible de vérifier les agents qui sont gérés par `sup_global`.

```
| ?- sup_global::ask(list_of_child,X).
X = [sup_1,sup_2]
```

Le système offre la possibilité de rajouter récursivement (si l'agent recevant le message est un superviseur, il le propage vers les agents de son groupe) les faits `a`,`b`, et `c` à tous les spécialistes.

```
| ?- sup_global::broadcast(feuilles,give(belief_base,[a,b,c])).
```

Saturation des bases de règles de croyances.

```
| ?- sup_global::broadcast(feuilles,saturate).
     the fact x is deduced.
     the fact e is deduced.
     the fact y is deduced.
     the fact f is deduced.
     the fact z is deduced.
     the fact g is deduced.
```

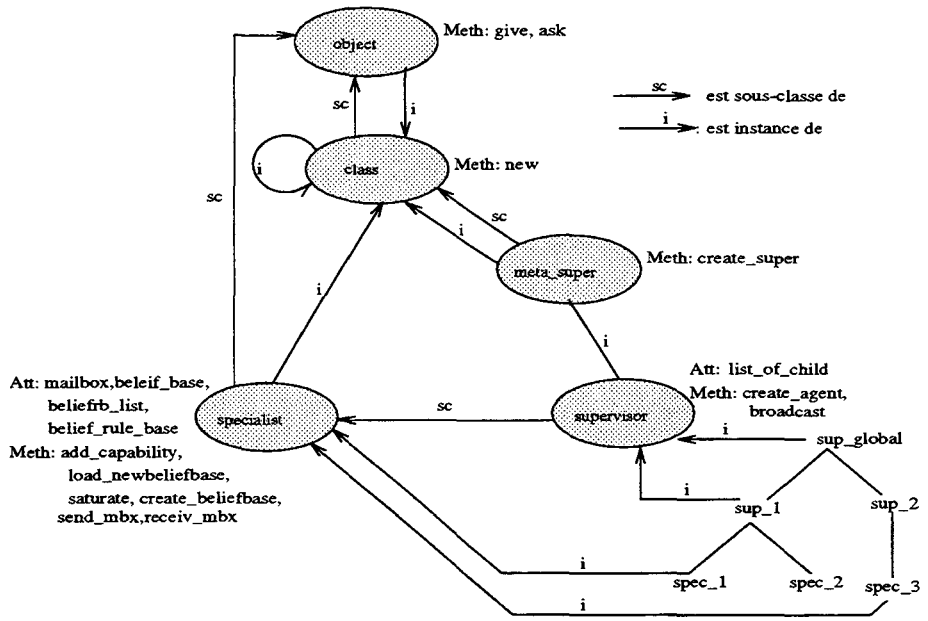

FIG. 5.3 - *Modélisation du système d'inférence distribué*

5.9.2 Un moteur d'inférence généralisé

Le schéma suivant montre une hiérarchie d'agents dans laquelle la connaissance est présente dans tous les agents (feuilles et nœuds de l'arbre) du système, aussi bien les agents spécialistes que les agents superviseurs (voir Fig. 5.4). Le modèle permet de faire de l'inférence répartie en diffusant le message `saturate` à tous les agents du système, quelque soit leur niveau hiérarchique.

Si l'on veut avoir des superviseurs capables de faire de l'inférence, il suffit de créer une classe `sup_inference` qui hérite simultanément des classes `superviseur` et `agent_inference` (voir Fig. 5.5).

Création des agents

```
| ?- meta_super::new(sup_inference,sc,[superviseur,agent_inference]).

| ?- sup_inference::create_super(s,null).

| ?- sup_inference::create_super(s1,s).

| ?- sup_inference::create_super(s2,s).

| ?- s1::create_agent(a1,agent_inference).

| ?- s1::create_agent(a2,agent_inference).
```

Création des bases de règles des agents

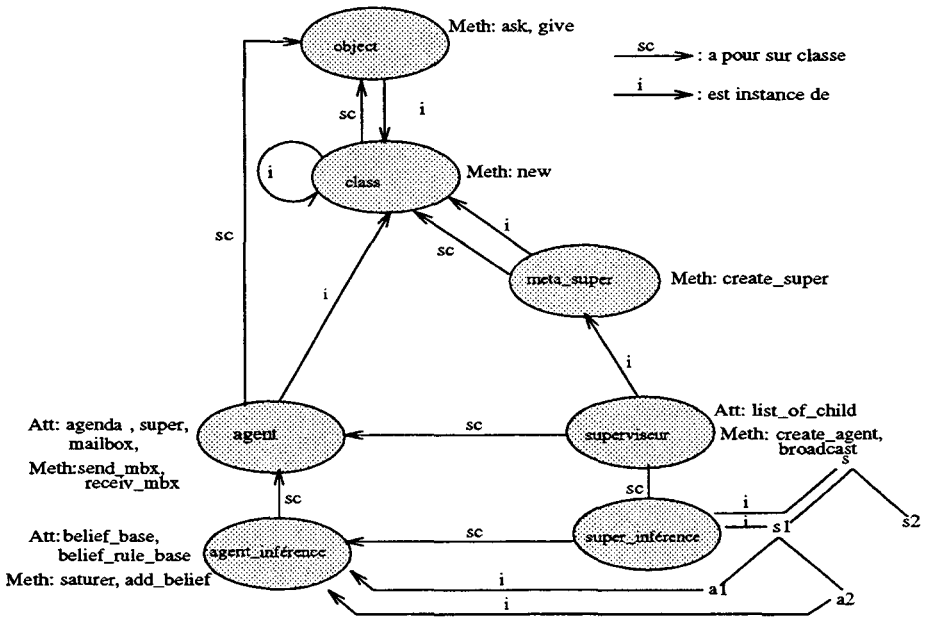

FIG. 5.4 - *Coopération entre superviseurs dotés de stratégie d'inférence*

```
| ?- s::give(belief_rule_base,[si a alors u,si a et b alors f]).

| ?- s1::give(belief_rule_base,[si b alors v,si b et c alors g]).

| ?- s2::give(belief_rule_base,[si c alors w,si a et c alors h]).

| ?- a1::give(belief_rule_base,[si d alors x,si a et d alors i]).


| ?- a2::give(belief_rule_base,[si a et b alors y,si c et d alors j]).
```

Exemple d'exécution

```
| ?- s::broadcast(tous,give(belief_base,[a,b,c,d])).


| ?- s::broadcast(tous,saturer).

     le fait x a ete deduit.
     le fait i a ete deduit.

     le fait y a ete deduit.
     le fait j a ete deduit.


s1::broadcast(tous,saturer)

     le fait v a ete deduit.
     le fait g a ete deduit.
```

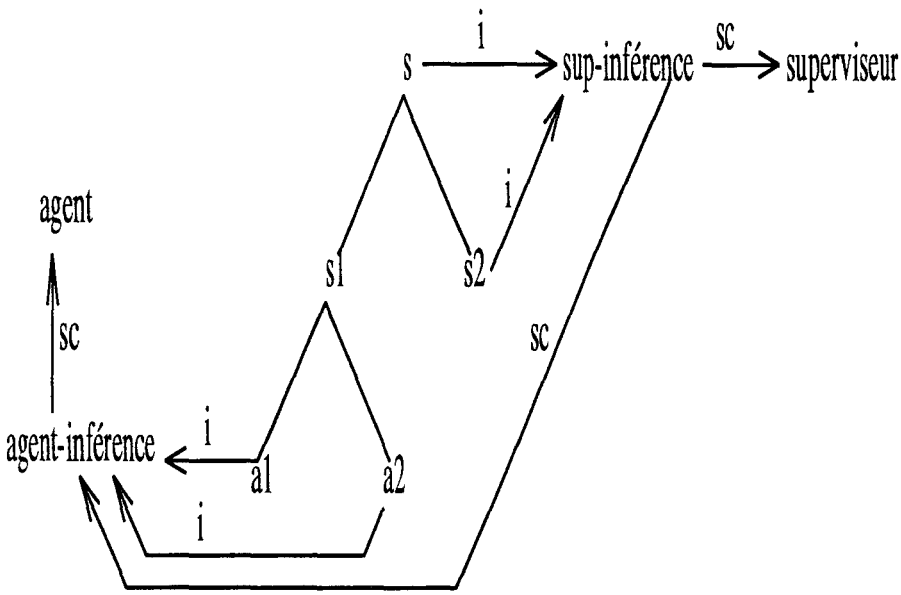

FIG. 5.5 - *Schéma d'instanciation des agents*

```
s2::broadcast(tous,saturer)

     le fait w a ete deduit.
     le fait h a ete deduit.
```

5.10 Raisonnement distribué et logique floue

L'un des intérêts primordiaux de notre architecture consiste à favoriser la communication entre différents types de moteurs d'inférence. Dans l'exemple suivant, les agents spécialistes sont dotés de capacités d'inférence, tandis que les superviseurs disposent d'une vue globale du système. La connaissance représentée au sein d'un agent est souvent implicite et incertaine. La logique floue offre un cadre formel pour le raisonnement sur de telles connaissances. Dans un système multi-agent, une même connaissance peut figurer dans des agents sociaux différents avec un degré de précision ou de certitude différent en fonction de la vision qu'a l'agent du processus de résolution.
Pour prendre en compte cet aspect lié à l'incertitude de l'information manipulée dans le cas de notre modèle (voir Fig. 5.6), chaque agent est doté d'un moteur d'inférence (ayant les deux modes d'inférence: chaînage avant et chaînage arrière) permettant de générer des nouvelles informations sous forme de croyances caractérisées par un coefficient de vraisemblance lié à cette croyance. La technique consistant à affecter un coefficient de vraisemblance à une croyance est celle retenue dans [HBC+91]. Elle est illustrée dans l'exemple suivant. Il faut noter qu'une croyance est représentée par un fait (voir figure 5.6).

```
si cond1(c1=0.8) et (cond2(c2=0.9) ou cond3(c3=0.7)) alors conc (cr=0.8)
```

cr est le coefficient de la règle. **conc** est déduit avec le coefficient **c** calculé par l'application

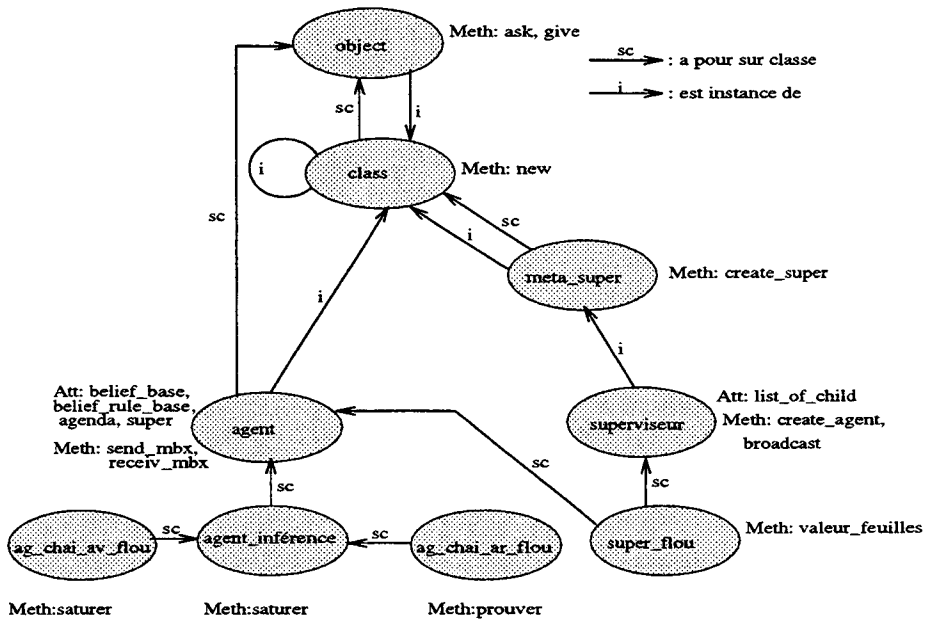

FIG. 5.6 - *Coopération entre plusieurs types de moteurs d'inférence*

de la fonction **max** pour les **ou** (disjonction) et **min** pour les **et** (conjonction).

```
c(conc)=min(0.8, max(0.9,0.7))*0.8= 0.64
```

Chaque agent génère une croyance avec un coefficient de vraisemblance qui lui est propre. Un superviseur peut interroger les agents de son groupe sur une croyance donnée pour récupérer le coefficient affecté par chacun des agents à cette croyance. Le calcul de la valeur de cette croyance au niveau du superviseur est effectué selon une technique donnée.

min(liste des valeurs) pour une approche pessimiste.
max(liste des valeurs) pour une approche optimiste.
moyenne(liste des valeurs) pour une approche équiprobable.

De cette manière, le superviseur se fera une "opinion" sur certaines croyances manipulées par les agents de son groupe.

Les classes **ag_chai_ar_flou** et **ag_chai_av_flou** sont des sous-classes de la classe **agent_inference**. Elles héritent donc des attributs **belief_base** et **belief_rule_base**.
La classe **ag_chai_ar_flou** possède un moteur d'inférence fonctionnant en chaînage arrière flou. Ce moteur est représenté par la méthode **prouver** qui a pour objet de prouver le but et de calculer son coefficient de vraisemblance.
La classe **ag_chai_av_flou** a un moteur d'inférence représenté par la méthode **saturate**. Ce dernier sature sa base de connaissances et génère les croyances déduites aprés avoir calculé leur coefficient de vraisemblance.

La classe **super_flou** hérite simultanément de la classe **superviseur** et de la classe **agent**. Elle possède la méthode **valeur_feuilles** permettant à un objet superviseur d'interroger les agents spécialistes de son groupe sur une croyance donnée et de calculer son coefficient de vraisemblance.

Création des agents

```
| ?- meta_super::new(super_flou,sc,[superviseur,agent]).

| ?- super_flou::give(meth,valeur_feuilles).

| ?- super_flou::create_super(s,null).

| ?- super_flou::create_super(s1,s).

| ?- super_flou::create_super(s2,s).

| ?- s1::create_agent(a1,ag_chai_av_flou).

| ?- s1::create_agent(a2,ag_chai_av_flou).

| ?- s2::create_agent(a3,ag_chai_ar_flou).

| ?- s2::create_agent(a4,ag_chai_ar_flou).
```

Création des bases de règles des agents

```
| ?- a3::give(belief_rule_base,[si b et c alors d cf 0.5 ,
si a et d alors e cf 0.5 , si a alors c cf 0.5]).

| ?- a3::give(belief_base,[connu(a,0.6),connu(b,0.7)]).

| ?- a3::prouver(e,Coeff).

Coeff=0.15

| ?- a1::give(belief_rule_base,[si b et c alors d cf 0.5 ,
si a et d alors e cf 0.5, si a alors c cf 0.5 ]).

| ?- a1::give(belief_base,[connu(a,0.6),connu(b,0.7)]).
```

connu(X,Y) est un prédicat permettant de représenter une information approximative.

Exemple d'exécution

```
| ?- a1::saturate.

le fait e a été déduit coeff 0.15
le fait c a été déduit coeff 0.3
```

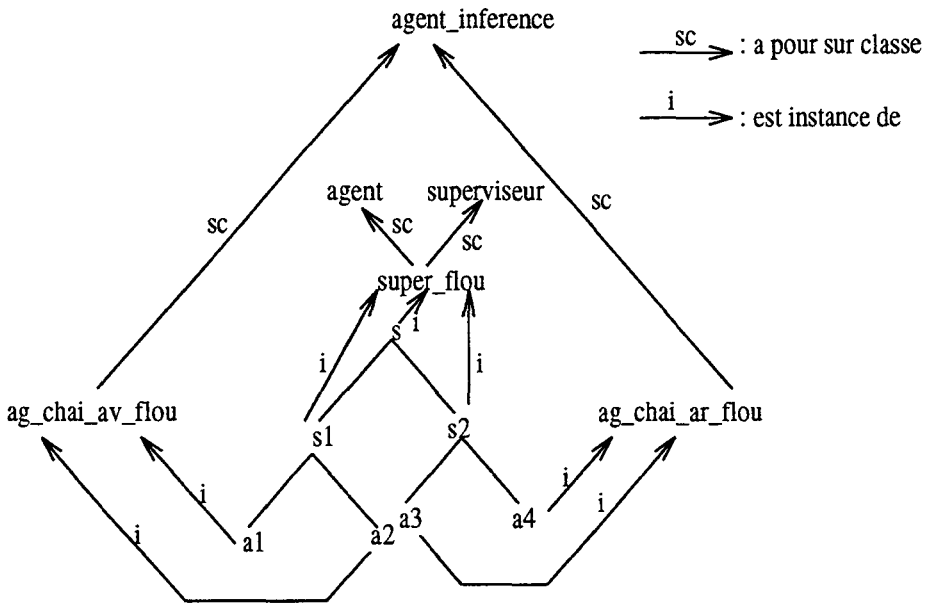

FIG. 5.7 - *Schéma d'héritage du système d'inférence en logique floue*

```
le fait d a été déduit coeff 0.15
```

Le superviseur s peut interroger tous les agents spécialistes (**a1, a2, a3, a4**) sur une croyance donnée et calculer ensuite son coefficient de vraisemblance selon une approche optimiste (prendre le max des valeurs associées à la croyance par les agents spéialistes).

```
| ?- a2::give(belief_base,[connu(b,1)]).

| ?- a2::give(belief_rule_base,[si b alors a cf 1,
si a et b alors e cf 0.5]).

| ?- a4::give(belief_base,[connu(b,0.4),connu(c,0.5)]).

| ?- a4::give(belief_rule_base,[si b ou c alors a cf 0.5]).

| ? a2::saturate.

le fait a a été déduit coeff 1
le fait e a été déduit coeff 0.5

| ?- a4::saturate.

le fait a été déduit coeff 0.25

| ?- s::valeur_feuilles(a,X).
```

```
X = 1 ?;
```

5.11 Limites

Le choix d'IC-Prolog comme langage pour développer notre plate-forme est motivé par un ensemble d'aspects intéressants tels que: le modèle de communication asynchrone basé sur la notion de boîte aux lettres, la bibliothèque de threads permettant de manipuler des tâches concurrentes ainsi que la puissance d'unification et les aspects déclaratifs qu'il offre au programmeur. De plus, le langage a été étendu pour supporter les aspects objets (encapsulation, instanciation, héritage, envoi de messages) sur un réseau de stations Unix. Bien qu'au début, certains tests que nous avons effectués se sont avérés encourageants, nous avons remarqué des blocages du système dès que le nombre d'agents créés dépasse 9 ou 10 agents. Nous avons eu confirmation qu'IC-Prolog contenait des bugs. Nous avons constaté que le système se bloque dès que sa charge dépasse un certain nombre d'agents. Ce problème est l'une des raisons qui nous ont poussé à choisir PM^2 pour implémenter notre application qui consiste à explorer un territoire par un ensemble de robots (voir chapitre 6).

5.12 Conclusions

L'objectif de ce chapitre demeure l'illustration d'une instanciation de notre architecture MAGIQUE. Les différents exemples présentés dans les sections précédentes visent à mettre en évidence les capacités de communication du modèle aux travers de systèmes d'inférence répartis. Les capacités d'inférence du système représentent la partie raisonnement de l'architecture, qui est à base de règles se déclenchant en chaînage avant. Le choix d'un langage objet à base d'IC-Prolog est motivé par un ensemble de propriétés qui le caractérisent notamment les possiblités d'inférence, l'aspect déclaratif ainsi que son système de communication entre threads distribués sur un ensemble de machines. L'aspect multi-threadé du système se justifie par le fait qu'un agent doit avoir la capacité de manipuler plusieurs tâches concurrentes simultanément. Il permet aussi à un agent d'avoir un comportement autonome. Toutes ces motivations ont rendu le prototypage de notre système très aisé. À travers ce chapitre, nous avons montré l'intérêt d'une architecture multi-agent hiérarchique pour supporter la communication de systèmes ayant des moteurs d'inférence différents. Dans ce cadre, les superviseurs gèrent les interactions entre des agents ayant des méthodes d'inférence différentes. En ce qui concerne le moteur d'inférence en logique floue, il peut servir d'outil à un système où la connaissance est connue de façon approximative. Ainsi, le superviseur peut indexer un coefficient de vraisemblance à une connaissance en fonction des coefficients attribués par les autres agents du groupe.
L'une des extensions de ce travail consiste à utiliser MAGIQUE comme outil pour faire communiquer des agents hétérogènes. Ainsi, des systèmes hétérogènes écrits par des personnes différentes dans des langages différents vont communiquer indirectement via des agents superviseurs. Ces derniers contiennent des "dictionnaires" permettant d'effectuer l'association entre une requête soumise par un agent et le service correspondant ainsi que les agents capables de satisfaire cette requête. À ce titre, un protocole doit être défini entre un agent hétérogène et ses superviseurs. Le protocole est une interface, constituée d'un ensemble de services permettant à des agents hétérogènes d'interagir.

L'objectif recherché via la communication d'agents hétérogènes est de fournir un haut degré de réutilisabilité logicielle. Par conséquent, MAGIQUE servira d'outil pour développer des applications à partir d'agents logiciels déjà existants. Il faut souligner que la notion de réutilisabilité diffère fondamentalement de celle utilisée dans le domaine du génie-logiciel. Ce dernier vise à concevoir et développer des composants logiciels en vue d'être utilisés ultérieurement pour le développement de nouveaux logiciels. Néanmoins, la réutilisabilité telle que nous la percevons du point de vue multi-agent concerne la réutilisabilité de systèmes qui n'ont pas été conçus pour être réutilisés. À ce titre, nous pensons établir un protocole de communication entre systèmes ouverts, en utilisant MAGIQUE comme cadre de communication. Ainsi, une batterie de primitives de communication doivent être implémentées au niveau du superviseur afin d'effectuer la traduction d'une requête et de la rediriger vers un agent compétent qui peut la traiter.

Chapitre 6

Exploration d'un territoire

Préambule:

Ce chapitre présente une application du modèle d'architecture conçu à l'exploration d'un territoire inconnu. L'idée sur laquelle repose l'application tire son origine de la métaphore qui consiste à larguer des robots sur une planète non explorée. À ce titre, les robots sont représentés par des agents spécialistes dont la coordination de leurs activités est assurée par des agents superviseurs.

L'application met en évidence un protocole de coopération entre le superviseur et les robots. Le protocole supporte l'aspect régulation de la charge entre les agents spécialistes. La solution du système est représentée par l'ensemble des chemins explorés et stockés dans le tableau noir du superviseur.

Les détails d'implémentation, l'environnement de programmation, ainsi que des exemples d'exécution de l'application sont fournis dans ce chapitre.

La dernière section est consacrée à la description d'une autre application de MAGIQUE dans le domaine de la robotique. Cette application permet de planifier le déplacement d'un agent robot en dehors de son environnement.

6.1 Introduction

Beaucoup de travaux ont vu le jour dans le domaine des systèmes multi-agents, notamment au niveau des aspects liés à la théorie des agents ainsi q'aux architectures et langages. Tous ces aspects ont fait l'objet d'un grand intérêt de la part des experts de la communauté multi-agent. Ces travaux [CL90] [Woo95] [MP93] [Sho93] visent à fournir un ensemble d'outils théoriques destinés à développer des environnements de conception, de spécification et de programmation de systèmes multi-agents et d'agents intelligents. Tous ces outils garantissent les fondements théoriques pour des systèmes performants supportant des applications complexes. Ainsi, plusieurs domaines d'application sont concernés par le paradigme multi-agent. Parmi les applications typiques, nous pouvons citer la collecte d'informations sur Internet, la téléconférence, la robotique intelligente [SP95] [Bal97], les agents softwares hétérogènes et la modélisation des processus industriels ou encore commerciaux. Dans ce contexte, nous avons utilisé l'architecture MAGIQUE [BM97b] pour simuler un groupe de robots explorant les différents chemins d'un territoire inconnu. L'idée résulte de la métaphore qui consiste à larguer à partir d'un engin aérien un ensemble de robots sur une planète non explorée. Les robots entament l'exploration à partir des différents points du territoire. Chacun des robots a pour tâche de poursuivre une direction jusqu'à ce que le robot rencontre un obstacle. Dans ce cas, le robot a trouvé un chemin, par conséquent, il informe ses accointances que ce chemin est exploré. Si le robot a d'autres alternatives à explorer, il réitère le processus précédent. Dans le cas contraire, le robot suscite une coopération avec ses accointances afin de trouver un autre point à partir duquel il va commencer l'exploration. Avant de commencer l'exploration d'un chemin, le robot doit s'assurer qu'il existe au moins un chemin non exploré issu à partir du point représentant la position courante du robot. Ainsi, le robot doit connaître tous les chemins qui sont déjà explorés par ses accointances. Il a une vision limitée de son environnement. Il peut juste percevoir si les points de son voisinage sont des obstacles ou non. Dans le cas où il existe plusieurs points du voisinage qui ne sont pas des obstacles, le robot choisit un point à explorer et enregistre les autres points de "choix" (ce sont des points qui constituent une alternative). Une fois le chemin complètement exploré, le robot revient à l'un de ses points de choix et réitère le même processus.
Un agent superviseur a une vision globale du territoire à explorer. A chaque fois qu'un robot a fini l'exploration de sa zone, le superviseur lui indique une autre zone non explorée. La coopération entre robots intervient quand un robot a fini l'exploration de sa zone et qu'il n'y a plus de nouvelles zones à explorer. Dans ce cas, le robot ayant fini le travail communique avec le robot le plus chargé. Un robot est considéré comme étant le plus chargé s'il a un grand nombre de points de choix à explorer. La coopération est possible s'il existe un chemin exploré entre les zones explorées par le robot le plus chargé et le robot le moins chargé. MAGIQUE a été utilisé comme cadre pour supporter cette application car tous les robots du système partagent le même espace d'exploration et les solutions obtenues sont stockées dans un tableau noir visible pour tous les robots [BM98]. Le tableau noir est géré par le superviseur du groupe.

6.2 La robotique distribuée

La robotique distribuée se fixe comme objectif la réalisation non pas d'un seul robot, mais d'un ensemble de robots qui coopèrent en vue de l'accomplissement d'une tâche donnée. À

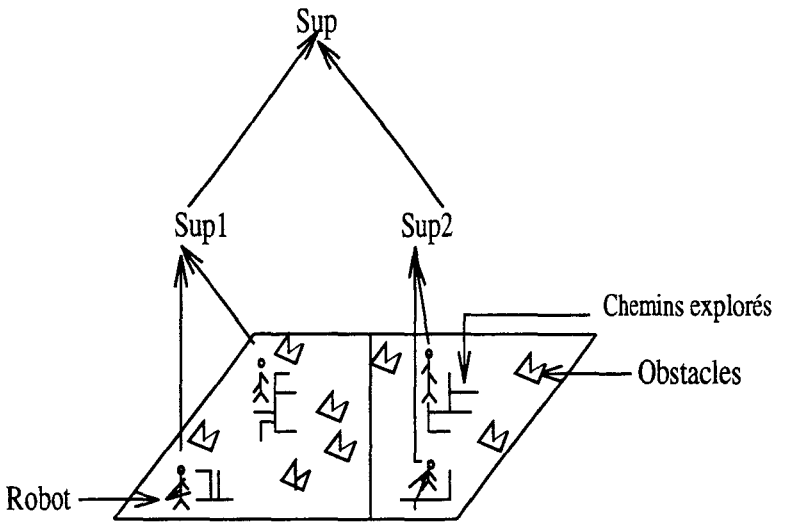

FIG. 6.1 - *Environnement de l'application*

la différence des autres domaines d'application des systèmes multi-agents, la robotique distribuée met en œuvre des agents concrets se déplaçant dans un environnement réel. Il existe deux types de robotique bien distinctes: la *robotique cellulaire* et la *robotique mobile.*
La robotique cellulaire postule qu'un robot doit être conçu comme un système multi-agent dont chacun des composants du robot est considéré comme un agent. Ainsi, un mouvement du robot résulte de la coordination d'un ensemble d'agents. Des travaux dans le domaine de la robotique cellulaire sont effectués dans les équipes de J. Perram au Danemark [OPP99] et de D. Duhaut en France [RD95] et ont donné naissance à des modèles de déplacement de bras manipulateur.
La robotique mobile suppose l'existence d'au moins deux robots. Chacun des robots est considéré comme un agent, et l'ensemble des robots forment le système multi-agent. Les robots doivent coordonner leurs déplacements et coopérer pour accomplir des tâches complexes comme le nettoyage d'un plancher, la surveillance des bâtiments, l'exploration d'espaces lointains ou dangereux. De nombreuses contributions ont vu le jour, notamment par M. Mataric [Mat94], L. Steels [Ste94] et J.-L. Deneubourg [BHD94].
Il faut noter aussi que la coordination de mobiles, qu'il s'agisse de voitures, d'avions ou de bateaux, fait partie du domaine d'application de la robotique distribuée. Par exemple, deux voitures qui roulent sur la même route, ou deux avions qui évoluent dans des zones voisines doivent coordonner leurs actions de telle sorte à ce qu'ils ne rentrent pas en collision. MAGIQUE est une architecture adaptée pour supporter des applications ayant des spécificités décrites précédemment. Dans notre cas, l'exploration d'un labyrinthe (voir la section 6.3) met en œuvre des robots représentés sous forme d'agents autonomes. La coordination et la coopération entre robots se fait via un protocole entre les robots et le superviseur.

6.3 Description de l'application

Cette application est une instanciation du modèle MAGIQUE pour simuler un groupe d'agents robots explorant un territoire. Les robots sont largués sur un territoire inconnu dont on veut identifier les chemins existants. La mission des robots consiste à explorer les diffé-

rents chemins de ce territoire en tenant compte des multiples obstacles (voir Fig.6.1). Dans notre cas, le territoire est représenté par un labyrinthe. Les robots sont représentés par des agents spécialistes ayant un ensemble de propriétés (voir Fig.6.2). Un spécialiste est capable de percevoir son environnement afin de savoir s'il existe des points obstacles ou non au sein de son voisinage. Si les points du voisinage de l'agent ne sont pas des obstacles, le spécialiste choisit aléatoirement d'aller dans une direction et range les autres points de choix dans sa base des croyances. Chaque position du labyrinthe, qui n'est pas un obstacle, a deux états: *exploré* et *non exploré*. Durant sa progression dans le processus d'exploration, le spécialiste ne prend pas en compte un point qui est déjà exploré ou qui est un obstacle. Comme les spécialistes accédent au labyrinthe et modifient l'état des positions explorées, ce labyrinthe est considéré comme une seconde structure tableau noir du système. Tous les points de choix (alternatifs) constituent le modèle de l'environnement du spécialiste. La base des règles de croyances consiste à effectuer un bactracking au niveau du processus d'exploration et à retirer de la base des croyances les points qui ont fait l'objet d'une exploration.
La base de coopération permet au superviseur de connaître la charge d'un spécialiste, particulièrement quand la charge de ce dernier croit considérablement. Le processus d'exploration consiste à exécuter une capacité appelée *explore(P)* permettant à un robot de commencer l'exploration à partir du point *P*. Quand un robot est chargé (il contient un nombre important de points de choix), il envoie une requête à son superviseur afin que ce dernier puisse opérer une régulation de charge entre les robots. Le superviseur suscite un processus de coopération entre le robot chargé et un autre robot qui peut être disponible ou le moins chargé parmi les autres. La coopération entre les deux robots est possible s'il y a un point de choix, dans la base des croyances du robot chargé, à partir duquel il existe un chemin exploré entre les deux robots candidats à la coopération. La recherche d'un chemin entre deux points se fait par le biais d'une capacité, caractérisant un robot, appelée *find_path*. Ainsi, le superviseur coordonne les interactions des robots. Sa base des croyances représente un modèle du groupe de robots. L'information relative à la charge des robots, ainsi que leur état (robot disponible ou actif) est stockée au niveau du superviseur. En outre, le superviseur gère un tableau noir contenant la solution courante. Le tableau noir est une structure contenant tous les chemins explorés. À chaque fois qu'un robot a fini l'exploration de sa zone, il envoie une requête à son superviseur pour l'informer. Ce dernier modifie sa base des croyances. Subséquemment, il cherche s'il existe une autre zone non explorée, auquel cas il active le robot disponible afin de commencer l'exploration; sinon, il cherche le robot le plus chargé et déclenche un processus de coopération entre le robot le plus chargé et le robot le moins chargé.
En plus du tableau noir contenant la solution courante et de celui représentant le labyrinthe, le superviseur renferme deux structures de données contenant la charge des robots et leur état (agents disponibles ou actifs). Il a aussi les capacités qui lui permettent de sérialiser les accès concurrents au tableau noir. Les robots utilisent des actions communicatives, *requête* et *informer* similaires à celles proposées dans [Sho93].

6.4 Protocole de coopération

Afin d'illustrer le fonctionnement du système, nous considérons un exemple d'interaction entre deux robots. Soient **A** et **B** deux robots ayant pour tâche d'explorer le territoire. Il existe trois cas d'interaction entre les deux robots.
(i) Un robot est disponible et il existe une zone non explorée. Le robot commence l'exploration

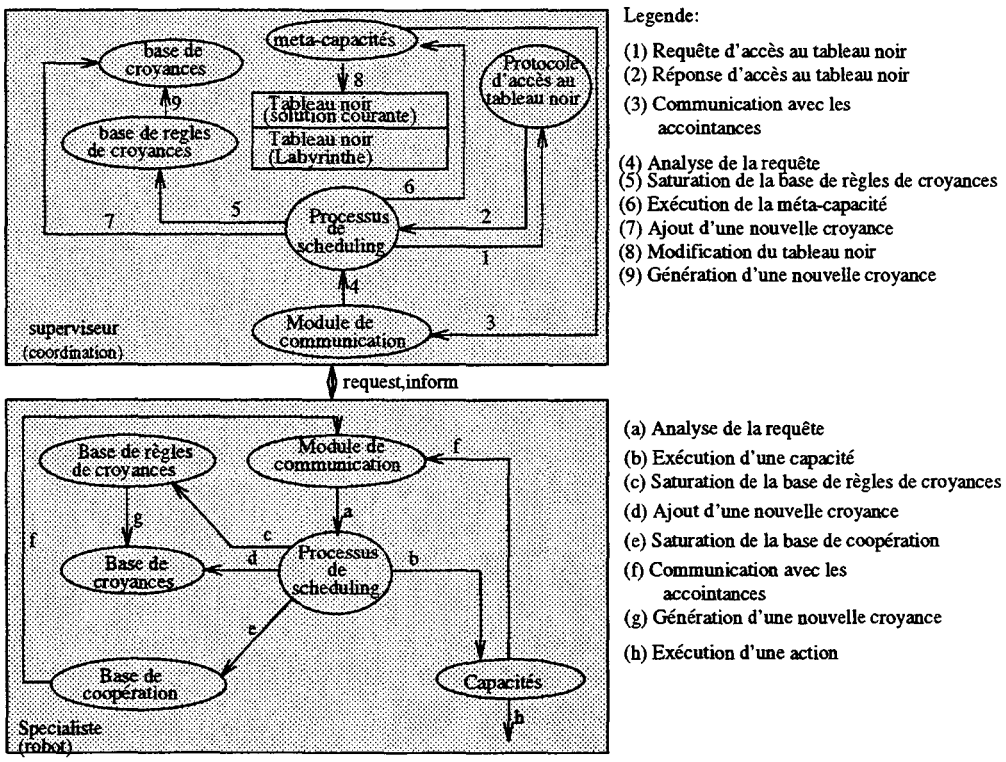

FIG. 6.2 - *Structure interne des agents*

de la nouvelle zone.

A: `inform(Sup, available_agent(A));`
Sup: `request(A, explore(P));`

P est un point à partir duquel le robot A va commencer le processus d'exploration. Le point P est identifié par le superviseur **Sup**.

(ii) Un spécialiste est disponible et il n'y a pas une nouvelle zone à explorer. Dans ce cas, le robot disponible coopère avec le robot le plus chargé.

A: `inform(Sup, available_agent(A));`
Sup: `inform(A, acquaintance(B));`
A: `request(B, cooperate(A));`
B: `inform(A, L);`

(iii) Le robot le plus chargé coopère avec le robot le moins chargé. Ainsi, le robot le plus chargé soumet une liste de points alternatifs au robot le moins chargé.

A: `request(Sup, loaded_robot(A));`
Sup: `inform(B, acquaintance(A));`
B: `request(A, cooperate(B));`
A: `inform(B, L));`

L est une liste de points alternatifs.

Un robot est considéré comme chargé si la taille de la pile des points alternatifs dépasse un nombre donné. Tous les robots ayant leur charge supérieure à ce nombre informent périodiquement leur superviseur sur leur charge respective. Ainsi, le superviseur a une vision globale de l'état de la charge des robots. Il détermine instantanément le robot le plus chargé et le robot le moins chargé. Pour une surface de territoire très grande, le territoire est divisé en plusieurs zones explorées par plusieurs groupes de robots. Par conséquent, le système utilise plusieurs tableaux noirs gérés par des superviseurs de groupe qui ont une vision globale de la zone qui leur est affectée. A chaque fois qu'une zone est entièrement explorée, le superviseur de groupe correspondant répercute la solution au superviseur global. Ainsi, ce dernier construit la solution globale à partir des solutions partielles obtenues par les superviseurs de groupes. Le nombre de robots qui participent à l'exploration est paramètrable.
Les robots sont dotés de trois stratégies d'exploration: exploration en profondeur, exploration en largeur et exploration sans backtracking. L'exploration en profondeur consiste, pour un robot arrivant à un carrefour, de choisir en premier lieu, la direction verticale. En revanche, l'exploration horizontale postule qu'un agent arrivant à un carrefour doit aller d'abord, dans la direction horizontale. Les deux stratégies précédentes permettent d'orienter l'exploration vers une direction donnée. L'exploration sans backtracking postule qu'un robot se trouvant à un carrefour doit choisir par défaut une direction et ignorer l'autre direction. Comme conséquence de ce dernier type d'exploration, des zones non explorées apparaissent dans le territoire.
Ces stratégies d'exploration peuvent être utilisées successivement quand un robot quittera dynamiquement un groupe pour intégrer un autre. Ainsi, il a la capacité d'adapter sa stratégie d'exploration à celle du groupe qu'il a rejoint.

6.4.1 Structure d'un spécialiste

Un robot a la structure suivante: une base des croyances, une base de règles des croyances, une base de coopération et un ensemble de capacités.
La base des croyances d'un robot (voir Fig.6.2) contient les attributs de l'ensemble **belief_base**

```
belief_base= {
          alternative_point_list,
          load_of_robot,
          state_of_robot,
          robot_current_position,
          };
```

Le premier attribut **alternative_point_list** contient un ensemble de points de "choix" représentant le modèle de l'environnement de l'agent. Le second attribut est une variable représentant la charge du robot (i.e., le nombre de points de "choix" qui restent à explorer). L'attribut **state_of_robot** représente l'état du robot. Ce dernier passe à l'état *inactif* dès que le robot ait fini l'exploration. Il passe à l'état *actif* dès qu'il est activé par son superviseur. Le dernier attribut est une structure de données représentant la position courante du robot.

La base de règles des croyances contient les règles décrites dans belief_rule_base. Il faut signaler que la base des règles de croyances n'est pas visible de l'extérieur.

```
belief_rule_base= {
          add_alt_point,
          remove_alt_point,
          change_load_of_robot,
          modify_current_position,
             };
```

Cette base de règles représente la partie *raisonnement* du robot. Elle contient 4 règles dont la première, **add_alt_point** permet de rajouter un point de "choix" à la base des croyances du robot. La seconde règle, **remove_alt_point**, a pour rôle de supprimer un point de "choix" de la base des croyances. La règle **change_load_of_robot** permet de modifier la charge du robot, tandis que la règle **modify_current_position** permet de modifier la position courante du robot.

La base de coopération contient deux règles correspondant aux événements robot disponible et robot chargé. Ces deux événements sont à l'origine des interactions aussi bien entre un robot et son superviseur, qu'entre deux robots distincts. La base des règles de coopération représente le modèle de coopération verticale, i.e., entre le spécialiste et le superviseur.

```
cooperation_base= {
          robot_available,
          robot_more_loaded,
                  };
```

La règle **robot_available** permet de gérer l'événement **robot disponible.** Dans le cas où il reste encore des zones non explorées, l'agent superviseur affectera une nouvelle zone non explorée à l'agent disponible. Dans le cas où il ne reste plus de zones non explorées, l'agent superviseur identifie le robot le plus chargé et l'informe de l'identité du robot disponible. Ce dernier envoie une requête de coopération à l'agent chargé. Finalement, l'agent chargé envoie à l'agent disponible une liste de points de "choix" à explorer.
La règle **robot_more_loaded** gère l'événement **robot chargé.** Quand la taille de la pile contenant des points de "choix" dépasse une certaine limite, le robot informe son superviseur. Dans le cas où il existe un robot disponible, le superviseur l'informe et suscite, en conséquence, une interaction entre l'agent disponible et l'agent chargé. Dans le cas contraire, le superviseur identifie l'agent le moins chargé et l'informe de l'identité du robot le plus chargé. Conclusion, le robot le moins chargé envoie une requête de coopération au robot le plus chargé.
Les capacités de l'agent sont définies dans l'ensemble **capabilities.**

```
capabilities= {
          explore, move_straight,
          move_left, move_right,
          acquaintance, cooperate,
          add_path_to_blackboard,
          find_path,
           };
```

La capacité **explore** permet de commencer l'exploration à partir d'un point donné. Les capacités **move_left** et **move_right** permettent au robot de tourner à gauche ou à droite en tenant compte de la position initiale et de la position courante. Les capacités **acquaintance** et **cooperate** gèrent les interactions d'un robot avec ses accointances. La capacité **add_path_to_blackboard** rajoute un nouveau chemin dans le tableau noir. L'invocation de da capacité **find_path** a pour effet de trouver, éventuellement, un chemin déjà exploré entre deux points donnés du labyrinthe.

6.4.2 Structure d'un superviseur

La structure du superviseur est similaire à celle du robot:

```
belief_base= {
          blackboard_path,
          blackboard_labyrinth,
          available_table,
          loaded_table,
             };
```

La base des croyances contient les deux structures de données, **available_table** et **loaded_table** représentant respectivement les noms des robots qui sont disponibles, et les noms des robots chargés ainsi que leur charge respective. Elle contient également, les structures de données **blackboard_path** et **blackboard_labyrinth** représentant respectivement le tableau noir renfermant les chemins explorés (i.e., la solution courante) et le tableau noir représentant le territoire à explorer.

```
belief_rule_base= {
          find_less_load_robot,
          find_more_load_robot,
          chang_load_robot,
          change_state_robot,
             };
```

La base des règles de croyances contient des règles et représente la partie raisonnement du superviseur. La première règle, **find_less_load_robot** permet de trouver le robot le moins chargé du groupe. En revanche, la seconde règle, **find_more_load_robot** permet de trouver le robot le plus chargé. La règle **chang_load_robot** a pour rôle de modifier la charge d'un robot. La dernière règle, **change_state_robot** permet de modifier l'état du robot.

Les méta-capacités du superviseur sont définies dans l'ensemble meta_capabilities.

```
meta_capabilities= {
          initialize_environment,
          blackboard_access,
          blackboard_leave,
          blackboard_modify,
          available_robot,
```

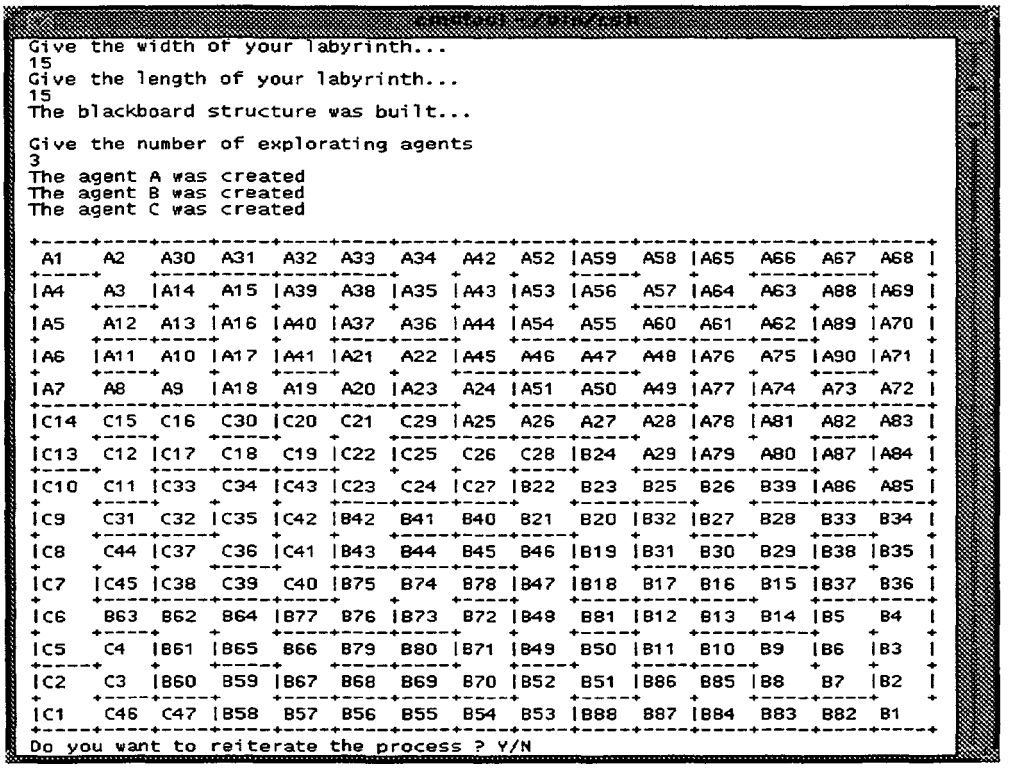

FIG. 6.3 - *Coopération de trois robots*

```
loaded_robot,
create_robot,
activate_robot,
remove_robot,
   };
```

Les méta-capacités caractérisent les superviseurs. L'exécution d'une méta-capacité affecte l'environnement partagé du groupe (exemple: initialiser l'environnement), ou encore permet de manipuler des objets partagés entre un groupe d'agents.
La méta-capacité **initialize_environment** permet d'initialiser l'environnement en créant les différents objets nécessaires (par exemple: créer le labyrinthe, créer le tableau noir, créer les robots, etc...). En plus des primitives *blackboard_access* et *blackboard_leave* dont le rôle est de sérialiser les accès concurrents au tableau noir, la méta-capacité *blackboard_modify* permet de modifier le contenu du tableau noir. Les méta-capacités *available_robot* et *loaded_robot* sont les plus importantes car elles sont à l'origine de la gestion de tout le protocole de coopération entre les agents, particulièrement l'aspect régulation de charge entre les robots. Enfin, *activate_robot* et *remove_robot* sont à l'origine de l'activation d'un robot ou de son retrait à partir de l'environnement physique.

6.5 Environnement de programmation

6.5.1 L'environnement PM^2

PM^2 (**P**arallel **M**ultithreaded **M**achine)[1] [NM95, Nam97] est un environnement de programmation bâti au dessus de PVM [GDD94]. Il permet de développer des applications à base de threads préemptifs dans un contexte distribué. Il est le résultat d'un couplage d'une bibliothèque de gestion de threads appelée MARCEL [NM95, Nam97], et de la bibliothèque de communication de PVM.
Ainsi, PM^2 hérite la notion de tâche à partir de PVM. Contrairement à PVM, les tâches de PM^2 sont "threadées" et offre ainsi une abstraction totale de l'architecture sous-jacente. La communication entre les différents threads de l'application s'appuie sur la notion de **LRPC**[2]. Cette dernière est la primitive principale de PM^2. Elle est implémentée en trois versions différentes: *synchrone, asynchrone et asynchrone avec attente différée.* La première version (LRPC synchrone) induit un blocage du thread initiateur jusqu'à ce que le résultat attendu soit retourné. La version asynchrone lance des traitements (locaux ou distants) indépendants. Elle représente en quelque sorte une version «dégradée» des LRPC qui n'attendent pas de résultats de la part du traitement effectué. Enfin, un LRPC nécessitant une attente explicite d'un résultat est appelé un "LRPC avec attente différé". Cette dernière primitive est importante car elle permet de lancer des traitements avant d'attendre leurs résultats correspondant. Il faut signaler aussi que PM^2 fournit plusieurs fonctionnalités intéressantes telles que la création à distance et la migration de threads, la gestion des priorités et le changement dynamique de la taille de la pile.

6.5.2 Implémentation

L'application est implémentée sur un réseau de stations Unix, sous l'environnement de programmation PM^2 [NM95]. Le choix de PM^2 par rapport au langage objet à base d'IC-Prolog, que nous avons développé, est motivé par les besoins en performances que pourrait induire un environnement comme PM^2. Il faut signaler que notre langage a pour but de permettre un prototypage facile et rapide du système en tirant profit des avantages du langage IC-Prolog. Néanmoins, il ne garantit point une bonne efficacité vu que le code généré est interprété.
PM^2 offre la possibilité de développer des agents distribués sur des machines différentes, capables de communiquer directement par l'envoi de messages. En plus de la communication asynchrone entre les agents, il existe un espace partagé entre les agents. Cet espace est induit par l'aspect régulation de charge consistant à permettre aux robots de manipuler des structures de données partagées, notamment la pile représentant la charge d'un robot. À cet égard, PM^2 est un environnement de programmation adapté car il fournit un modèle transparent de communication entre threads répartis sur des machines différentes. Dans notre cas, un robot correspond à un processus esclave, tandis que le superviseur correspond à un processus maître. Le modèle d'implantation est un ensemble de threads communiquant entre eux via un réseau de stations. Chaque agent est implémenté sous forme d'un objet concurrent renfermant un ensemble de threads. Le nombre de robots est variable. Le superviseur a la capacité d'ajouter ou de retirer dynamiquement des robots à son groupe. La capacité d'un robot est

1. PM^2 est développé au Laboratoire d'Informatique Fondamentale de Lille par l'équipe GOAL
2. Lightweight Remote Procedure Call

programmée sous forme d'un service LRPC ou sous forme d'une méthode. Un service LRPC est un morceau de code pouvant être exécuté en parallèle par plusieurs threads "logés" sur un même site ou sur des sites différents. La structure tableau noir est représentée par un fichier partagé par l'ensemble des agents du système. Un mécanisme d'exclusion mutuelle, basé sur les sémaphores, est implémenté pour prendre en compte le problème de cohérence dû aux multiples accès au tableau noir. Ce mécanisme est constitué de deux services LRPC appelés respectivement: **blackboard_access** et **blackboard_leave**. En plus des services précédents, le robot a une capacité nommée **explore** dont le rôle est de permettre à un robot de commencer l'exploration d'une zone donnée du territoire. Au niveau du tableau noir, un chemin est représenté par une structure contenant un ensemble de points constituant ce chemin. Concernant le robot, le protocole de coopération est supporté par deux services appelés respectivement: **acquaintance** et **cooperate**. Le service **acquaintance** est invoqué à partir du superviseur. Il a pour but de déclencher la coopération entre un robot inactif et le plus chargé des autres. Le service **cooperate** est exécuté au niveau du robot le plus chargé. Il est invoqué à partir d'un robot inactif. Son but est de fournir une liste de points alternatifs au robot inactif. Pour le superviseur, le protocole de coopération est supporté par deux services: **available_robot** et **loaded_robot**. Le service **available_robot** est invoqué par un robot ayant fini l'exploration de sa zone, tandis que le service **loaded_robot** est invoqué par un robot chargé. En plus des services précédents, le superviseur est caractérisé par deux capacités: **activate_robot** et **initialize_environment**, programmées comme des méthodes. La capacité **initialize_environment** permet au superviseur de générer un territoire, tandis que la capacité **activate_robot** est utilisée pour activer un robot inactif. Le superviseur a deux structures de données: **available_table** et **loaded_robot**. La première contient la liste des robots inactifs (ayant fini le travail), tandis que la seconde maintient la charge courante des robots.

6.6 Exemple de code

6.6.1 Création d'un spécialiste

Le morceau de code suivant permet de créer un agent spécialiste sur une machine donnée.

```
void supervisor::specialist_create(int i)
   { int cc;

    cc= pm2_start_modules("/home/smac/bensaid/pvm3/bin/SUN4SOL2/
    robot", NULL, PvmTaskDefault, "",1 , &tid[i]);
               if (cc != 1)
              {
                cout<<"The task: "<< cc<<"is launched\n";
                cout<<"The task identifier is: "<< tid[i]<<"\n";
                exit(1);
              }
   }
```

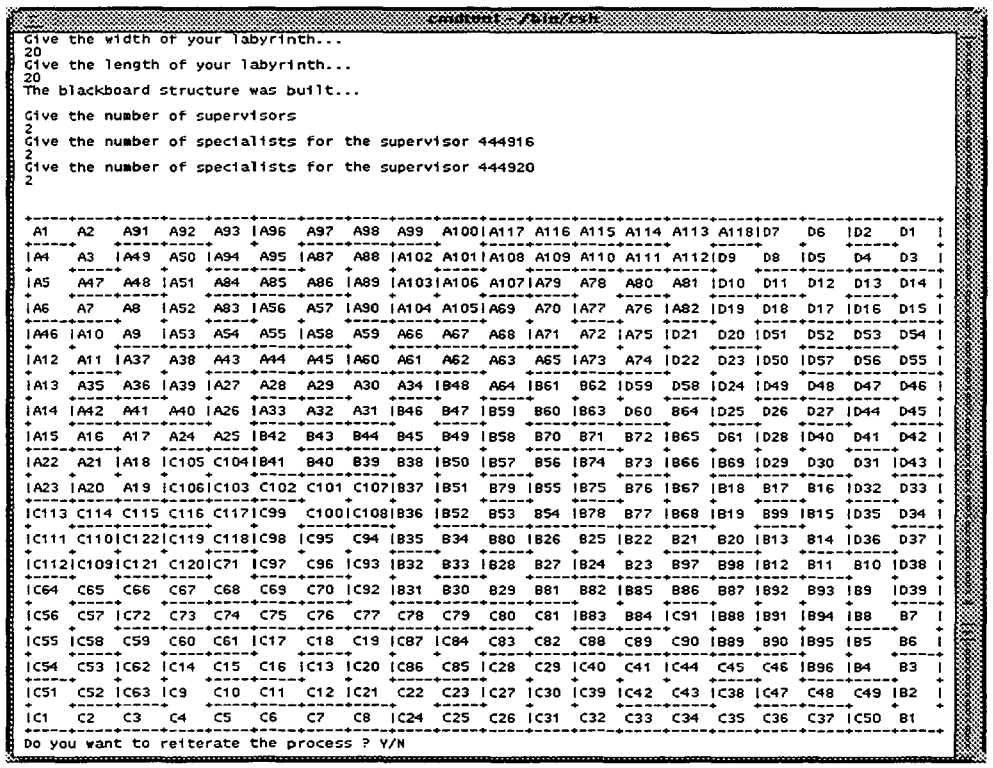

FIG. 6.4 - *Coopération de deux groupes de robots*

6.6.2 Activation d'un agent

Le morceau de code suivant permet de lancer le processus d'exploration d'un agent robot sur une machine donnée en invoquant le service **EXPLORE.**

```
void supervisor::specialist_activate(int agent_identifier, point P)
   {
    LRPC_REQ(EXPLORE) req_exp;
    req_exp.abscissa=P.abscissa;
    req_exp.ordinate=P.ordinate;
    ASYNC_LRPC(agent_identifier,EXPLORE,STD_PRIO,DEFAULT_STACK,
    &req_exp);
   }
```

Règle de croyance d'un robot

```
// tester si on peut aller dans la direction est
if((Point.ordonne<colonne)&&
// tester si le point n'est pas déjà exploré
```

```
(my_blackboard[Point.abscisse][Point.ordonne].state == '0') &&

(my_blackboard[Point_courant.abscisse][Point_courant.ordonne].east =='1') &&
// tester si le point perçu n'est pas un obstacle
non_deja_empile(&pile,Point))
// tester si le point n'a pas été déjà perçu

      {
       pthread_mutex_lock(&mutex);
       empiler(&pile,Point);
       // mettre le point perçu (alternative) dans la base des faits (pile)
       pthread_mutex_unlock(&mutex);
       // passer à un autre point à percevoir
       Point.abscisse = Point_courant.abscisse;
       Point.ordonne = Point_courant.ordonne -1;
      }
```

6.6.3 La capacité COOPERATE

Cette capacité est implémentée sous forme d'un service sous PM^2. Il est exécuté au niveau d'un agent robot et est invoqué par un autre agent accointance.

```
LRPC_SERVICE(COOPERATE)
// Ce service permet de décharger un robot en divisant sa charge par 2
 int i;
 point P;

    tfprintf(stderr,"le thread COOPERATE est invoqué \n");
    if (pilenonvide(&pile)){
     res.size=(pile.sommet)/2;
     //diviser la charge du robot par 2
     i=0;
     while ((i<res.size) && (pilenonvide(&pile))){
         // accès en exclusion mutuelle
         pthread_mutex_lock(&mutex);
         // préparer la structure de données
         P=depiler(&pile);
         pthread_mutex_unlock(&mutex);
         res.tab_abscisse[i] = P.abscisse;
         res.tab_ordonne[i] = P.ordonne;
         i++;
         }
         if(i==0) res.size=-1;
         else{
             state_charge =1;
             res.size = i;
             }
```

```
      }
   else res.size = -1;

END_SERVICE(COOPERATE)
```

6.6.4 La capacité LOADED_ROBOT

Cette capacité est implémentée sous forme de service et est exécutée au niveau du superviseur. Elle est invoquée par un agent robot.

```
LRPC_SERVICE(LOADED_ROBOT)

 int i,j,agent_disp,agent_charg;

 // Déclaration des services invoqués dans le présent service.
 LRPC_REQ(ACCOINTANCE) req_acc;
 LRPC_RES(ACCOINTANCE) res_acc;
 LRPC_REQ(EXPLORE) req_exp;

 // Exécution du service en exclusion mutuelle.

 pthread_mutex_lock(&mutex);

 i=0;
 while ((i<NbAgent) && (tab_charge[i].tidag != req.tidag)) i++;
    if (tab_charge[i].tidag == req.tidag){
         tab_charge[i].charge++;
         tfprintf(stderr, "Agent charge trouve...\n");
         }
         // Rechercher l'agent le plus chargé et l'agent disponible

         agent_disp=agent_disponible();
         agent_charg=agent_le_plus_charge();
         if ((agent_disp != -1) && (agent_charg != -1)){
        req_acc.tidag = tab_charge[agent_charg].tidag;

        // Invoquation du service accointance pour effectuer une régulation de
        // la charge entre le robot disponible et le robot chargé.

        LRPC(tab_available[agent_disp].tidag,ACCOINTANCE,STD_PRIO,DEFAULT_STACK,
  &req_acc,&res_acc);

        // Préparation de la requête du service.

        req_exp.abscisse = res_acc.abscisse;
        req_exp.indice = agent_disp+1;
        req_exp.ordonne = res_acc.ordonne;
```

```
        if ((req_exp.abscisse != -1)&&(req_exp.ordonne != -1)) {
        tab_available[agent_disp].etat=1;
        tab_charge[agent_charg].charge=0;

// Activation du  robot disponible et invoquation de la capacité EXPLORE
ASYNC_LRPC(tab_available[agent_disp].tidag,EXPLORE,STD_PRIO,DEFAULT_STACK,&req_exp);
        }
        }
 // Sortie de la section critique.

 pthread_mutex_unlock(&mutex);
 END_SERVICE(LOADED_ROBOT)
```

6.6.5 Exemple d'exécution:

La figure 6.3 montre un labyrinthe de forme carrée et de dimension 15. L'exploration est effectuée par 3 robots nommés respectivement A, B et C. Chaque robot "signe" la case qu'il a explorée en inscrivant la lettre correspondant à son nom ainsi que le numéro de l'étape d'exploration. À la onzième case d'exploration, le robot **A** rencontre un obstacle. En conséquence, il backtracke afin de commencer l'exploration d'un nouveau chemin à la case **A12**. Les robots **A** et **B** rentrent en conflit quand ils arrivent respectivement aux cases **A29** pour **A** et **B24** pour *B*. Dans ce cas, les deux robots rebroussent chemin et commencent l'exploration de nouveaux chemins aux cases **A30** pour **A** et **B25** pour **B**. La figure 6.4 montre un labyrinthe de dimension 20, exploré par deux groupes de robots dont les superviseurs sont **S1** et **S2**. Le labyrinthe n'est pas divisé en zones. Il est partagé entre tous les robots du système. Le superviseur **S1** supervise les robots **A** et **B**, tandis que **S2** gère le groupe constitué des robots **C** et **D**. Les robots du premier groupe ont une stratégie d'exploration verticale, tandis que les robots du second groupe ont une stratégie d'exploration horizontale. Pour la première stratégie, le robot **A**, une fois arrivé à la case **A2**, choisit d'aller dans la direction verticale. En revanche, pour la seconde stratégie, le robot **C**, une fois arrivé à la case **C66**, préfère aller dans la direction horizontale. À la case **C50**, le robot **C** se rendant compte que la prochaine case (**B1**) a déjà été explorée, backtracke pour commencer l'exploration d'un nouveau chemin à la case **C51**. Les problèmes de conflits entre robots sont résolus de la même manière que dans l'exemple de la figure 6.3.

La figure 6.5 montre un labyrinthe de dimension 20 dont l'exploration est effectuée par 4 agents robots nommés respectivement: **A, B, C** et **D**. La stratégie d'exploration est, par défaut, une stratégie verticale. On remarque que l'aspect régulation de charge est illustré par la coopération entre les robots **C**, **D** et **A**. À l'étape 22, le robot **C** a fini l'exploration de sa zone. Par conséquent, le robot **C** se déplace jusqu'à la zone explorée par le robot **D** et commence l'exploration dans cette zone à l'étape 23. Après avoir exploré 5 cases, le robot **C** rencontre un obstacle. Il se déplace, comme précédemment, dans la zone explorée par le robot **A** afin de participer à l'exploration de cette zone. Par conséquent, il commence l'exploration dans cette zone à l'étape 31.

La figure 6.6 montre un labyrinthe de dimension 20 avec trois superviseurs de groupe dont

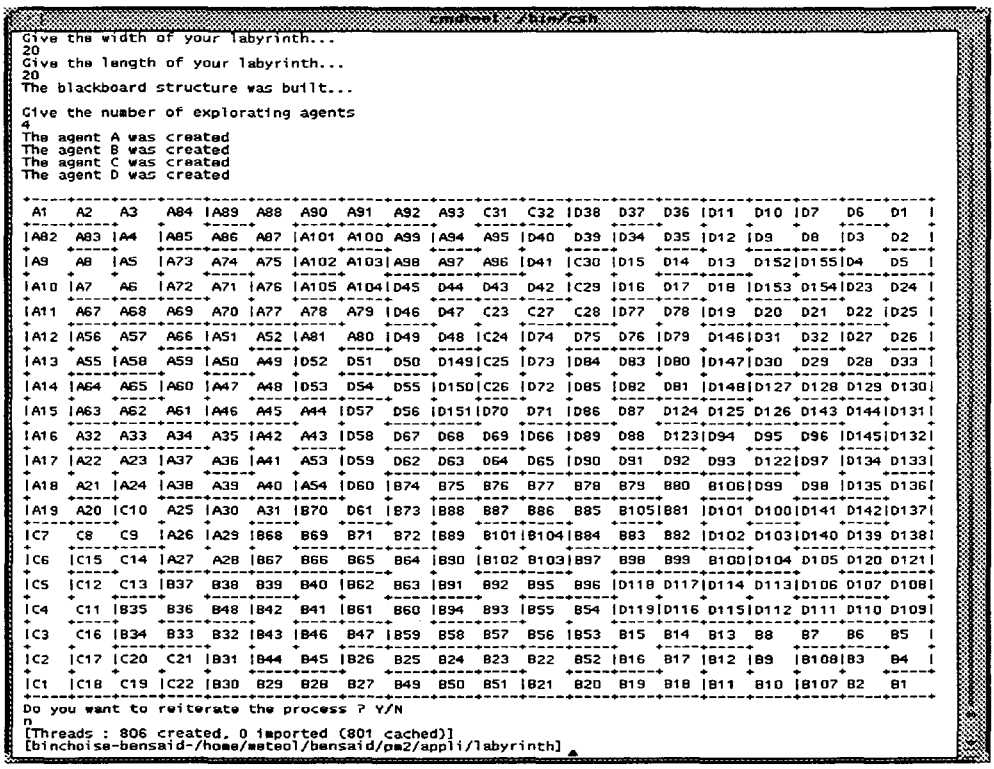

FIG. 6.5 - *Coopération avec régulation de charge entre robots*

chacun est caractérisé par une stratégie d'exploration. Chaque groupe contient un seul agent robot ayant une stratégie différente de celle des autres agents robots. Le robot du premier groupe est nommé A et a une stratégie d'exploration verticale, celui du second groupe est nommé B et a une stratégie d'exploration horizontale; par contre, le troisième a une stratégie verticale sans backtracking. Vu que le robot **C** a une stratégie d'exploration sans backtracking, il ignore complètement les points de choix. Par conséquent, il y a apparition de zones dans le labyrinthe qui n'ont pas été explorées. La régulation de charge entre les robots illustrent bien l'interaction entre les agents robots et met en avant le concept de coopération horizontale qui caractérise le modèle MAGIQUE.

6.7 Avantages du système

Dans cette section, nous examinons les avantages du système:

- MAGIQUE est adapté pour supporter l'application car les solutions explorées par les robots sont stockées dans le tableau noir, alors que les robots du système sont dotés de capacités de communication et de raisonnement élaborées. La structure tableau noir contient des objets représentés par des chemins qui sont déjà explorés. Un robot doit tenir compte de la configuration du tableau noir (i.e.: la solution courante), avant d'entamer l'exploration d'un nouveau chemin. En d'autres termes, le comportement du robot est opportuniste car il tient compte de la solution courante.

- Chaque robot est considéré comme un agent autonome car il possède toutes les capacités d'action (déplacement dans l'environnement), de perception (le robot a la capacité de détecter si les points du voisinage sont des obstacles ou non), de communication (les robots interagissent mutuellement afin de réguler la charge entre eux) et de prise de décisions (un robot informe son superviseur sur sa charge respective, envoie une requête à ses accointances afin d'obtenir des points de coopération, etc...).

- Le contrôle est efficace car il est hiérarchique et est basé sur plusieurs agents superviseurs. La connaissance de contrôle, à l'image de la charge de chacun des robots, ainsi que leur état (actif ou non actif) est représentée explicitement au niveau des agents superviseurs. Dans le cas où plusieurs groupes de robots ayant des stratégies d'exploration différentes coopèrent, un agent superviseur est affecté à chaque groupe pour coordonner les interactions des robots de ce groupe. Les robots ainsi que les superviseurs travaillent en parallèle, ce qui améliore considérablement les performances du système.

6.8 Efficacité

Bien que l'utilisation de l'environnement de programmation PM^2 au lieu du langage objet à base d'IC-Prolog a pour but de garantir une certaine efficacité au système, il n'est pas toujours vrai que l'efficacité soit au rendez-vous. En effet, dans un réseau ou la bande passante est limitée, ou encore dans le cas ou il y a un "trafic" important sur le réseau se traduisant par une saturation de la bande passante du réseau, les délais de communication entre les agents robots sont très importants. Par conséquent, l'objectif du processus de régulation de charge conçu initialement pour améliorer les performances du système se voit hypothéqué car entre le moment où un agent chargé communique à un autre agent moins chargé une partie de son travail, et l'instant où ce dernier reçoit la requête, l'agent chargé aurait déjà fini son travail. Par conséquent, les résultats sont intéressants dans le cas où le coût de la communication est faible. En revanche, les performances se détériorent dès que le coût de la communication devient important.
Ainsi, pour pallier à ce problème, il convient de prendre en compte la charge du réseau comme paramètre dans le processus qui permet à un agent de décider s'il faut effectuer une régulation de charge ou non. À ce titre, il convient de coupler à notre système un régulateur de charge capable de tenir compte de ce coût de la communication. L'objectif est de savoir d'une manière instantanée à quel moment il est intéressant d'effectuer une régulation de la charge de telle sorte à ce que cette régulation soit "rentable" pour le système en terme de performances. Nous avons remarqué durant les différents tests que nous avons effectués, que pour des dimensions faibles du labyrinthe, il est plus intéressant en terme de performances d'utiliser un seul robot que plusieurs. Par exemple pour un labyrinthe de dimensions 20, la coopération de plusieurs robots donnent des performances très modestes voir négatives. En revanche, pour des dimensions importantes (supérieures à 50), la coopération de plusieurs robots offrent un gain sensible en terme d'efficacité.

6.9 Simulation d'un environnement multi-robot

Nous décrivons dans cette section une autre application de l'architecture MAGIQUE. Dans les environnements industriels, les robots sont utilisés comme processus physiques de

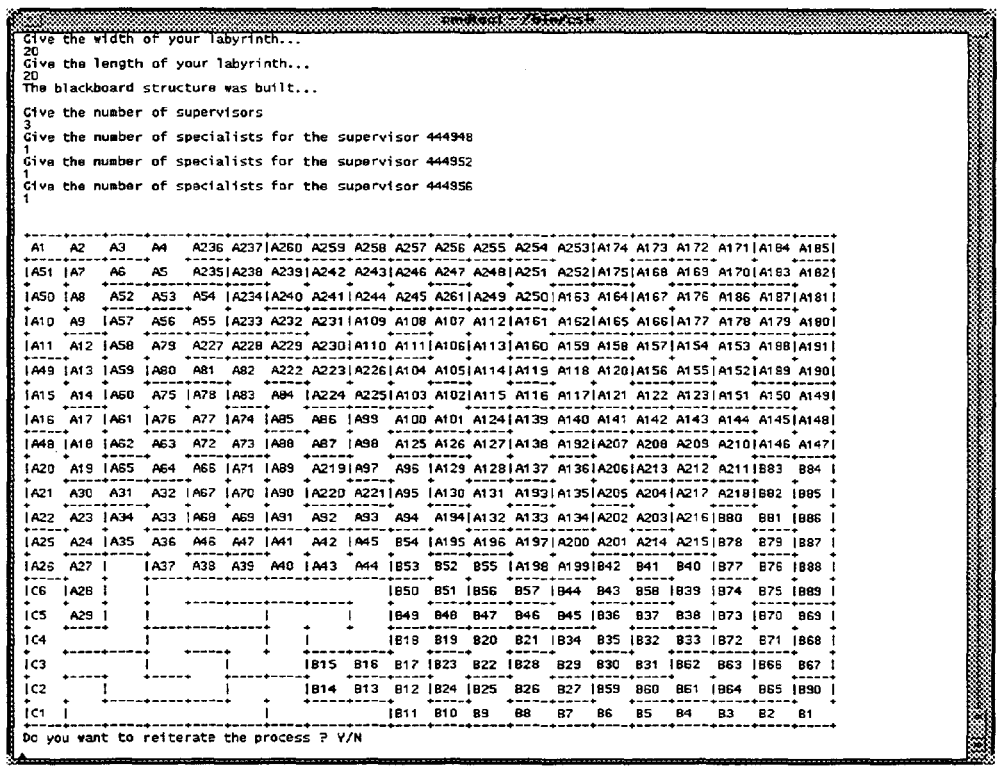

FIG. 6.6 - *Coopération entre superviseurs ayant des stratégies différentes*

production. Dans un environnement dynamique, un robot doit être capable de s'adapter à une nouvelle situation et se comporte ainsi comme un agent adaptatif. Dans notre étude, nous supposons qu'une application contient plusieurs groupes d'agents robots appelés spécialistes. Ces derniers travaillent simultanément pour satisfaire un ensemble de buts. Un spécialiste a une vision partielle du système; il peut simplement percevoir son propre environnement et connaît ses accointances avec lesquels il coopère. Il est incapable de planifier une tâche en dehors de son environnement. Le problème auquel nous nous intéressons est le déplacement du robot en dehors de son environnement de travail. Par exemple, on peut imaginer une panne au niveau d'un robot appartenant à un groupe d'agents donné. Dans ce cas, le superviseur global désigne un superviseur de groupe qui doit déléguer un robot pour remplacer celui qui est affecté par cette panne. Le superviseur global peut aussi consolider un groupe d'agents en demandant à un superviseur d'un autre groupe de déléguer un robot pour cette mission. Durant le déplacement, le robot doit éviter les multiples obstacles qu'il peut rencontrer. Comme ce robot a une vision seulement de l'environnement dans lequel il travaille, le superviseur doit lui fournir un plan détaillé afin de lui décrire le plan à exécuter pour atteindre la position désirée. Ce plan contient un ensemble de capacités que le robot délégué doit exécuter. Dans le cas où un superviseur de groupe n'est pas en mesure d'établir ce plan, il doit coopérer avec d'autres superviseurs de groupe sous la responsabilité du superviseur global.

Un robot peut être vu comme une entité générique dotée d'un ensemble de capacités qui lui permettent d'actionner son bras manipulateur. Il peut être fabriqué par une firme et utilisé par une autre dans un environnement spécifique. MAGIQUE est adapté pour assurer la réutilisabilité d'agents spécialistes développés pour contrôler les mouvements d'un bras manipulateur d'un robot, par le biais de superviseurs qui ont pour tâche d'adapter ces agents à

un environnement particulier.
Nous ne nous intéressons pas à la manière dont le robot est construit, ni à sa structure physique. Des problèmes de synchronisation peuvent aussi avoir lieu; néanmoins, nous faisons abstraction de ce type de problèmes.

6.9.1 Structure d'un robot

Un agent robot peut être dans une situation de déplacement dans l'environnement, ou dans un état opérationnel. Dans le dernier cas, le robot exécute un plan d'actions entraînant un mouvement des doigts de son bras manipulateur. Il a la capacité *hand_move* pour déplacer chacun de ses doigts dans les trois directions possibles.

```
hand_move(name_finger,direction,distance)
```

Le paramètre `name_finger` indique le nom du doigt à actionner.
Le paramètre `direction` indique la direction (horizontale, verticale ou en altitude) du mouvement du doigt du robot.
Le paramètre `distance` indique la longueur du déplacement du doigt du robot.
En plus de la capacité précédente, le robot possède la capacité d'effectuer une rotation à ses doigts.

```
hand_rotate(name_finger,Φ,Θ)
```

Les arguments Φ et Θ représentent les angles polaires de la rotation.
Pour un spécialiste, la résolution d'un problème consiste à planifier une séquence de capacités (décrites précédemment) en fonction de son environnement, puis l'exécution de ces capacités. Comme souligné dans la section précédente, un robot a la capacité de se déplacer dans l'environnement. Il peut se déplacer d'une position A pour aller vers une position B. Ainsi, il peut être considéré comme un *agent physique mobile*. Les capacités pour le déplacement dans l'environnement sont décrites ci-dessous.

```
move_straight(distance);
turn_left(angle);
turn_right(angle);
```

Au delà des capacités décrites, un robot a la capacité `stop_activities`, lui permettant d'arrêter ses activités et de se préparer à faire un déplacement dans l'environnement. L'exécution de cette capacité provoque le blocage de toutes les tâches actives liées à son bras manipulateur, ainsi que toutes ses activités sensorielles.
Le superviseur fournit au robot un plan composé d'un ensemble de capacités à exécuter. Sa base des croyances maintient l'état de l'environnement du groupe de spécialistes. Ces croyances représentent les coordonnées des différents obstacles de cet environnement. Vu que l'environnement est potentiellement dynamique, à chaque fois qu'un obstacle apparaît (respectivement disparaît), le superviseur réagit à cet événement en modifiant sa base des croyances. Par conséquent, le superviseur doit éventuellement réviser le plan courant du spécialiste en déplacement afin de prendre en compte le changement survenu. Il a une fonction appelée `plan_path` qui prend comme arguments les coordonnées de la position courante, celles du point d'arrivée,

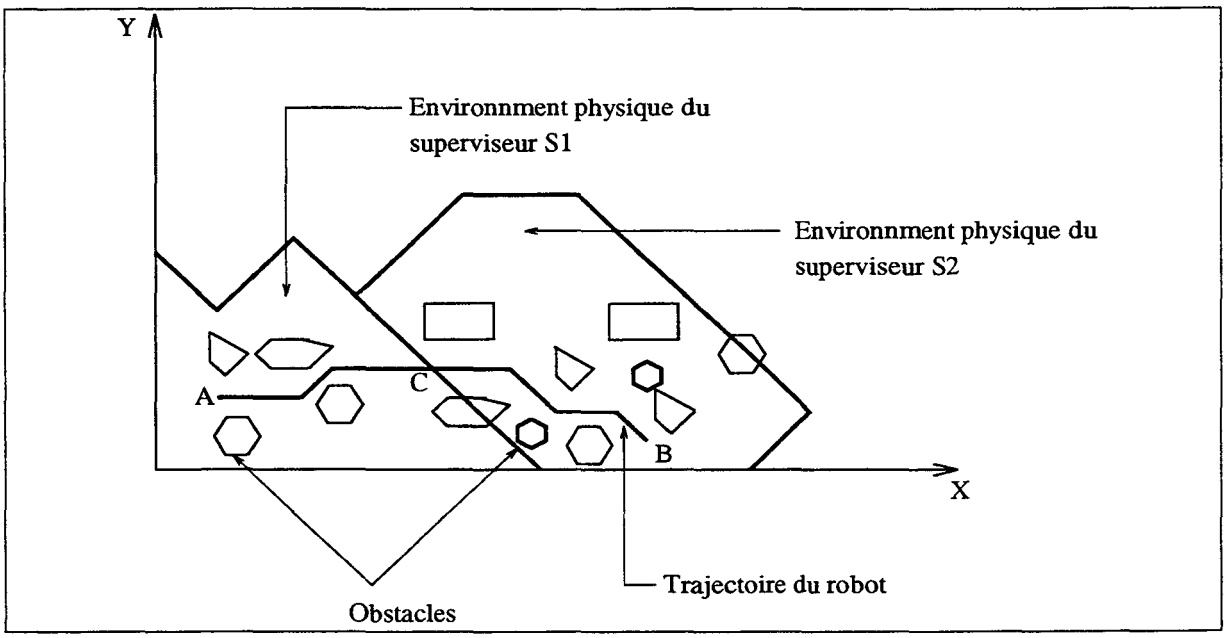

FIG. 6.7 - *Coopération entre superviseurs de groupe*

la base des croyances et retourne une liste d'actions à exécuter pour permettre à un robot d'atteindre sa position finale. Par exemple, le superviseur fournit le plan suivant.

```
turn_left(20°);
move_straight(10 m);
turn_right(90°);
move_straight(20 m);
```

6.9.2 Coopération entre superviseurs

Un superviseur de groupe a une vision limitée de l'environnement physique. Tous les superviseurs de groupes dont l'environnement physique allait être traversé par le robot doivent coopérer afin d'établir un plan global de déplacement. Chaque superviseur de groupe planifie une séquence d'actions pour conduire le robot dans la partie de la trajectoire traversant son environnement physique. Le superviseur global détermine les superviseurs qui doivent coopérer et spécifie la trajectoire que le robot doit suivre. Il fournit les coordonnées du point C (voir Fig.6.7) comme croyances aux superviseurs de groupes. En connaissant la position initiale et le point final que le robot voudrait atteindre (les points A et B dans notre cas), le superviseur global dispose d'une fonction `sup_cooperate` qui prend les coordonnées de ces points, sa base des croyances et retourne une liste (`List_of_sup`) de superviseurs de groupe qui peuvent coopérer pour satisfaire ce but. La base des croyances du superviseur global représente les coordonnées des points appartenant à l'intersection de deux environnements physiques adjacents. Ces points d'intersection sont appelés *points de coopération* entre superviseurs de groupe.
Soit `S` le superviseur global et `S1,S2` les superviseurs de groupe.
Pour planifier un robot afin de lui permettre de se déplacer d'une position A vers une position B, le superviseur `S` doit exécuter la séquence suivante.

$sup\_cooperate(bel - base, A, B, List\_of\_sup)$;
$\forall(S_i, S_j) \in List\_of\_sup \times List\_of\_sup$
$inform(S_i, M(X,Y))$;
$inform(S_j, M(X,Y))$;
$broadcast(List\_of\_sup, plan\_path)$;

M est le point d'intersection entre les environnements physiques de S_i et S_j, `X,Y` sont ses coordonnées. Le résultat de l'exécution du message `plan_path` par un superviseur de groupe va engendrer la planification du robot sur une trajectoire gérée par ce superviseur.

Pour le superviseur S1, le plan est constitué des actions suivantes :

```
move_straight(30 m);
turn_left(45°);
move_straight(30 m);
```

Pour le superviseur S2, la séquence d'actions planifiée est :

```
move_straight(25 m);
turn_right(45°);
move_straight(20 m);
turn_left(45°);
move_straight(20 m);
turn_right(45°);
move_straight(15 m);
```

Une fois que le robot ait atteint la position finale, il doit être intégré dans un nouveau groupe de spécialistes. Le superviseur S2 doit initialiser le robot en lui spécifiant la base de règles des croyances à charger afin de lui permettre d'être opérationnel dans le nouvel environnement.

6.10 Conclusions et perspectives

À travers ce chapitre, nous avons proposé une instanciation de notre modèle MAGIQUE pour simuler un groupe de robots représentés sous forme d'agents spécialistes, dans l'exploration d'un territoire inconnu. Le système développé illustre les interactions directes ou indirectes via le superviseur d'un groupe d'agents spécialistes. L'application est un exemple démontrant que le modèle du tableau noir et le modèle d'agents autonomes peuvent être combinés pour former un modèle supportant une large variété d'applications. La structure tableau noir contient les chemins explorés et représente la solution courante du système. L'implémentation du système a révélé que pour des problèmes caractérisés par une solution globale, MAGIQUE constitue l'alternative pour supporter ce type de systèmes. L'information globale relative à la charge des robots, ainsi que leur état, sont représentés explicitement au niveau du superviseur. Nous avons remarqué que dans le cas où le labyrinthe a des petites dimensions et que plusieurs robots participent au processus d'exploration, les performances du système

se détériorent à cause d'une densité de communication importante entre les agents.
Subséquemment, il existe un intérêt certain à établir une relation entre la surface du labyrinthe et le nombre de robots actifs afin d'améliorer l'efficacité du système. Un aspect qui mérite d'être exploré, consiste à déterminer dynamiquement le nombre de superviseurs qui seront chargés de coordonner les interactions des robots en tenant compte des dimensions du labyrinthe. Le but est de distribuer dynamiquement le processus de contrôle au travers d'un ensemble de superviseurs de groupes afin d'accroître les performances du système. Après l'implémentation de l'application sur un réseau de stations Unix, nous pensons à une implémentation sur d'autres types d'architectures telle que les processeurs DEC/ALPHA, puis nous comparons les résultats afin de déterminer l'architecture qui soit la mieux adaptée.
Un autre problème qui peut être résolu dans le cadre de l'architecture MAGIQUE consiste à utiliser des robots déménageurs pour effectuer des déménagements. La mission des robots consiste à porter des meubles et de transporter des objets fragiles, ainsi que des véhicules autonomes. Ainsi, le type de comportement requis est très similaire à celui des robots explorateurs. En outre, des contraintes s'y ajoutent, notamment la nature des objets (certains meubles peuvent être lourds pour qu'un seul robot puisse les porter tout seul, les couloirs sont étroits et il est parfois impossible pour que deux robots se croisent). Ce problème implique des modèles de coordination, de coopération et de négociation appropriés. En effet, un robot incapable de porter un meuble tout seul doit coopérer avec un autre robot afin d'atteindre son but; un robot doit coordonner ses actions afin de ne pas rentrer en conflit (passage au travers d'une porte de sortie ou seulement un seul robot doit traverser en même temps) avec les autres; enfin, les robots peuvent être appelés à négocier afin de trouver une solution de compromis de telle sorte à ce qu'aucun agent ne serait pénalisé par rapport aux autres. Dans ce cadre, MAGIQUE peut aussi supporter des applications de ce genre dans le sens ou il offre un cadre de coopération et de coordination adapté.
Ainsi, les robots seront représentés sous forme d'agents spécialistes ayant une autonomie quant à l'exécution de leurs actions. Des protocoles de coopération adaptés, des algorithmes de négociation et des stratégies de coordination doivent être spécifiés au niveau des superviseurs afin de prendre en compte tous les problèmes évoqués précédemment. Alors que les robots ont une vision locale, l'environnement global est représenté au niveau des superviseurs qui pourraient interagir avec les robots lorsque ces derniers quittent par exemple leur environnement pour intégrer un autre. Les premiers travaux peuvent se faire en simulation car le problème est en lui même potentiellement distribué et est caractérisée par la forte multiplicité des paramètres. Une fois que le fonctionnement du système aurait donné des résultats satisfaisants, il y aurait lieu de passer à la réalisation effective du système.

Conclusions et perspectives

Contexte

L'objectif de notre thèse est de proposer un modèle d'architecture multi-agent pour supporter des applications complexes et variées. L'architecture proposée est intermédiaire à deux modèles jadis antinomiques, en l'occurence les architectures tableau noir et les systèmes d'agents autonomes. L'architecture tableau noir est un modèle de résolution de problèmes distribués, où les modules du système coopèrent via un partage d'informations. Elle est caractérisée par la structure tableau noir qui sert de cadre pour une représentation explicite de la solution courante du système. D'un autre côté, les systèmes d'agents autonomes offrent un cadre conceptuel et méthodologique pour concevoir des systèmes complexes. Ils postulent qu'un système doit être conçu comme un ensemble d'agents autonomes coopérant mutuellement afin d'atteindre un but commun: la solution au problème. Ainsi, des systèmes complexes basés sur des agents autonomes et qui nécessitent un cadre partagé de coopération, exigent une approche nouvelle combinant aussi bien les fonctionnalités des architectures tableau noir que celles des agents autonomes.

Propositions

Partant de ce constat, notre contribution vise à définir une nouvelle architecture multi-agent hiérarchique dont le but est de tirer profit des avantages des architectures tableau noir et de ceux des systèmes d'agents autonomes. L'architecture MAGIQUE est caractérisée par deux types d'agents: les spécialistes et les superviseurs. Les premiers ont toutes les propriétés des agents autonomes (perception de l'environnement, raisonnement sur les connaissances perçues et enfin action sur l'environnement). Ils détiennent l'expertise du domaine et sont spécialisés dans la résolution d'une tâche spécifique. Le modèle de coopération entre les spécialistes repose sur deux types de communication: communication directe et communication indirecte. La communication directe entre spécialistes s'appuie sur l'échange d'actes communicatifs abstraits, tandis que la communication indirecte vise à fournir un cadre de coopération partagé entre un groupe de spécialistes via leurs superviseurs. Ces derniers offrent un cadre de coopération partagé entre un ensemble de spécialistes. En plus, ils coordonnent les interactions de leur groupe.
Alors que la notion d'autonomie est une propriété fondamentale pour un agent, les agents de MAGIQUE sont caractérisés par une propriété supplémentaire qui est la semi-autonomie. Cette dernière est complémentaire à la notion d'autonomie. La semi-autonomie est fondée sur l'idée qu'un agent a besoin d'être "guidé" afin d'évoluer dans un environnement inconnu. Contrairement aux spécialistes, les superviseurs ont une vision globale de l'environnement de

leur groupe d'agents. Conséquemment, ils ont la capacité d'orienter le processus de résolution dans une direction donnée en sélectionnant un ensemble de capacités au niveau des spécialistes.
L'implémentation de notre modèle a nécessité un environnement de programmation adapté afin de supporter des aspects inhérents aux architectures multi-agent tels que: la concurrence des tâches d'un agent, la communication asynchrone entre les agents, la transparence au niveau de la location des agents et enfin les aspects objets (instanciation, envoie de messages, héritage) qui garantissent une programmation naturelle d'un système multi-agent. Nous avons proposé une plateforme qui en plus de tous les aspects évoqués précédemment, permet de prototyper facilement et rapidement des systèmes multi-agents. IC-Prolog a été choisi pour implémenter la plateforme à cause de sa puissance d'unification, de ses capacités d'inférence ainsi que de ses aspects multi-threadés. Le modèle a été testé sur un ensemble de systèmes d'inférence répartis. Malheureusement, des blocages inhérents au langage IC-Prolog ont été constatés dès que la charge du système devienne importante.
MAGIQUE a été appliquée pour simuler un territoire inconnu par un ensemble de robots. Le système est caractérisé par son comportement global au travers de la solution courante représentée par l'état courant du tableau noir. En plus, les agents robots ont des capacités de perception, de raisonnemnt et d'action sur leur environnement. Le système supporte un mécanisme de régulation de la charge entre les robots. Ce mécanisme met en œuvre un ensemble d'interactions d'un côté entre les robots (interaction horizontale), de l'autre côté, entre les robots engagés dans le processus de régulation et le superviseur (interaction verticale).
Des stratégies d'exploration différentes ont été implémentées afin d'illustrer l'utilisation de notre architecture en vue de supporter des systèmes complexes. À cet égard, trois groupes de robots ayant des stratégies d'exploration différentes participent simultanément à l'exploration du labyrinthe. Chaque groupe de superviseur contient un superviseur qui coordonne les interactions des robots du groupe.
Il faut noter qu'en terme de performances, le gain est très modeste voir négatif lorsque la taille du labyrinthe est petite. Par exemple pour un labyrinthe de taille 20, un seul robot explorateur est plus efficace qu'un groupe de robots travaillant simultanément. Par contre, pour des labyrinthes de taille importante (50, 80 ou plus), la participation de plusieurs robots au processus d'exploration est fortement recommandée car le gain en efficacité est notable.
Enfin, il est envisagé d'utiliser MAGIQUE pour supporter une application de navigation dans une surface virtuelle. Le système contient 3 agents spécialistes (un agent interface, un agent planificateur et un agent navigateur) et un superviseur. Les spécialistes ont des tâches spécifiques telles que l'identification des profils d'un client en fonction de ses préférences, l'élaboration d'un plan de navigation adapté au profil du client, ou encore la navigation du client dans la surface (exécution du plan). Les spécialistes sont des agents intelligents qui interagissent directement ou indirectement via un tableau noir. L'agent superviseur coordonne les interactions en tenant compte de l'état du tableau noir.
La nouveauté dans cette application est que contrairement aux applications client/serveur classiques, les traitements sont distribués entre la machine du client (les agents interface et navigateur s'exécutent sur la machine du client) et la machine du serveur. Par conséquent, il y a un gain sensible sur le plan efficacité et fiabilité.

Perspectives

L'avènement du réseau Internet avec toutes les retombées suscitées en matière de stockage de masses d'informations importantes au travers des sites Web nécessitent un accès et une utilisation adaptés aux besoins et au profil de l'usager. Alors que les applicatifs classiques consistent à interroger d'une manière statique des banques de données éloignées, il convient de fournir des outils et des techniques qui permettent de doter ces masses d'informations de mécanismes capables de supporter des services de très haut niveau. Un agent doit avoir des outils de coopération et de négociation avec d'autres agents distants géoraphiquement et conçus et développés par d'autres personnes. L'objectif visé est de garantir une meilleure exploitation du logiciel en privilégiant la réutilisabilité et l'interopérabilité d'un côté et une meilleure coopération entre des entités logicielles ouvertes de l'autre côté, afin de fournir des services de qualité.
La tendance se dirige vers une coopération tout azimut d'entités logicielles hétérogènes et réparties géographiquement. L'idée est de mettre à la disposition des développeurs d'applicatifs des outils et des techniques leur permettant de "produire" un logiciel de qualité en favorisant l'accès à des sources d'informations distantes et en fournissant des protocoles de coopération et de négociation de haut niveau. Dans ce cadre, l'approche multi-agent est appelée à jouer un rôle important dans la conception et le développement de protocoles de coopération et de négociation entre systèmes ouverts. Particulièrement, les systèmes multi-agents hiérarchiques sont des "candidats" sérieux pour cibler ce type d'applications, car les niveaux hiérarchiques constituent un cadre pour spécifier les protocoles de coopération et de négociation entre des agents hétérogènes.

Un autre piste de recherche concerne l'utilisation de MAGIQUE pour supporter des applications en robotique. Par exemple, des robots déménageurs peuvent être représentés par des agents autonomes dont le protocole de négociation et de coopération dépend de l'état de l'environnement réel dans lequel ils évoluent. Vu que les robots n'ont pas une vision globale de l'environnement réel, la coordination des actions des robots est assurée par des agents superviseurs. En conséquence, les protocoles de négociation, de coordination et de coopération se trouvent spécifiés au niveau des agents superviseurs. Ainsi, un robot peut s'intégrer dans un environnement réel auquel il n'etait pas préparé à priori. Le superviseur se charge de le "guider" dans le nouvel environnement en le configurant de telle sorte à ce que son comportement soit cohérent et que l'objectif visé soit atteint.
Une piste supplémentaire concerne l'utilisation de l'approche multi-agent pour modéliser des applications de travail coopératif. Dans ce type d'applications, le comportement de l'utilisateur est représenté par un agent cognitif. Par exemple, dans une application de téléenseignement (un groupe d'élèves assiste à un cours donné par un professeur se trouvant dans une salle lointaine), on peut utiliser MAGIQUE pour représenter l'application. Dans ce cas, l'enseignant est représenté par un superviseur, tandis que les élèves sont représentés par des agents spécialistes. Cependant, il se pose souvent un problème de gestion des ressources critiques. Par exemple, l'écran de l'enseignant ne peut pas visualiser l'ensemble des élèves simultanément, d'où la sélection d'un sous ensemble des élèves à afficher. Cette dernière opération correspond à une intention dans les architectures BDI car elle permet de sélectionner un sous ensemble d'options parmi un grand nombre. L'intention résulte du fait que l'agent est limité au niveau des ressources et qu'un agent est incapable de tout faire faute de ressources.

Travaux futurs

Préambule:

Cette partie de notre travail décrit un application multi-agent destinée à supporter une composante d'un système de commerce électronique. Le but est de fournir un service permettant à un client, accédant via un ordinateur, de naviguer dans un magasin virtuel d'une manière efficace et souple. Le système fournit une vue et un parcours du magasin adaptés au profil du visiteur.

Nous avons utilisé l'architecture MAGIQUE pour supporter les aspects multi-agents nécessaires pour le développement d'applications complexes et adaptatives. Les systèmes de commerce électroniques existants sont rigides et statiques pour permettre une navigation adaptative et efficace d'un magasin virtuel.

Le catalogue d'un magasin peut contenir des milliers d'articles, alors qu'un client ne s'intéresse en général qu'à un nombre restreint de ces articles. Aussi, le parcours d'un catalogue n'est pas linéaire et n'est pas forcément le même pour deux personnes différentes.

L'idée est de proposer à un client donné, accédant à distance à un magasin virtuel, un plan de navigation adapté. En effet, le plan de visite dépend du profil du client. Avec cette approche, nous passons du marketing de masse au marketing personnalisé.

L'utilisation des agents intelligents en tant que nouvelle technologie permet de représenter aisément les propriétés d'adaptation, de planification et de coopération qui caractérisent le système.

Introduction

L'essor pris par Internet au niveau des possibilités de communication entre ordinateurs distants géographiquement a suscité un grand intérêt qui s'est traduit par la nécessité de mettre au point des services de très haut niveau permettant aux utilisateurs d'accéder à une

variété de ressources distribuées physiquement. Ces services de haut niveau sont des applications distribuées qui permettent par exemple à un utilisateur, d'effectuer un achat à distance, de confirmer son départ à une agence de voyage, ou encore de consulter son compte bancaire. L'avènement du World Wide Web autorise des accès à de grandes masses d'informations par le biais de son navigateur, moyennant une grande richesse ergonomique de l'interface Homme-Machine [BLC90]. Le navigateur communique avec le serveur WWW par l'intermédiaire du protocole HTTP[3].

Cependant, tous ces outils d'accès à des ressources distribuées sur Internet ne sont pas suffisants pour concevoir et développer des applications distribuées, complexes et adaptatives. L'approche multi-agent caractérisée par un haut degré de modularité et d'abstraction est bien adaptée pour résoudre des problèmes physiquement distribués. Elle postule qu'un système doit être conçu comme un ensemble d'agents autonomes qui coopèrent entre eux afin de résoudre un problème donné [SZ96]. De plus, elle offre un cadre conceptuel et méthodologique pour supporter des applications distribuées. En outre, les applications client/serveur sur un réseau peuvent être supportées par une architecture multi-agent hiérarchique comme MAGIQUE. Dans ce cas, chaque client se comportera comme un agent spécialiste autonome ayant toutes les propriétés lui permettant de percevoir son environnement, de raisonner sur le comportement de ses accointances et de coopérer avec eux directement ou indirectement via l'agent superviseur. Le serveur est représenté par un agent superviseur car il coordonne des activités concurrentes appartenant à plusieurs agents. Ainsi, des usagers distants auront accès à des services de très haut niveau en utilisant des supports applicatifs adaptés.

Dans ce cadre, nous proposons une application de l'architecture MAGIQUE à la navigation dans un magasin virtuel. Le but est de fournir un nouveau service permettant à n'importe quel usager (peut importe le lieu géoraphique où il se trouve!) d'accéder à une surface (exemple: Auchan, Carrefour,...) virtuelle contenant des articles, en fonction de son profil et de ses préférences. L'application constitue un bon support de publicité dans la mesure où l'entreprise dispose d'un outil puissant lui permettant de faire connaître ses produits aux clients même les plus lointains. En effet, l'intérêt des entreprises pour des systèmes de ce genre réside au niveau de leur volonté d'accroître leur productivité en augmentant le volume de vente de leurs articles.

Le système a pour tâche de saisir les préférences de l'utilisateur, de les analyser, puis d'établir un plan de navigation de l'utilisateur dans le magasin. Pour ce faire, un ensemble d'agents spécialisés doivent interagir et coopérer mutuellement afin d'assister l'usager dans son processus de navigation.

Une description détaillée de l'application sur le plan fonctionnelle est donnée. Ensuite, un ensemble d'agents spécialisés sont présentés. Tout d'abord, un agent interface dont le rôle est de recueillir les préférences de l'utilisateur. Un agent planificateur analyse les préférences de l'utilisateur et construit un plan de navigation. Enfin, un agent navigateur qui permet à l'usager de naviguer d'une manière conviviale dans le magasin en utilisant des techniques liées aux interfaces Homme/Machine. L'ensemble des préférences générées par l'agent interface sont consignées dans un tableau noir géré par un agent superviseur. En plus, ce dernier coordonne les activités des autres agents spécialisés. Toutefois, le système fournit un plan de navigation adaptatif car le système est en mesure de réviser dynamiquement le plan de navigation courant s'il détecte certaines particularités au niveau du comportement du client. Les sections suivantes définissent les cycles de fonctionnement de chacun des agents du sys-

3. Hyper Text Transfer Protocol

tème, le modèle de raisonnement des agents sur leurs connaissances, le processus de construction du plan de navigation, ainsi que le modèle de coopération entre les agents.

Description de l'application

L'objectif de cette application est de fournir aux entreprises un outil d'aide au commerce éléctronique. Le système développé aura à offrir à un usager virtuel (distant géographiquement) une vue adaptée à son profil et conforme aux préférences qu'il aura fournies au système. Le fonctionnement du système consiste en premier lieu à interroger le client et à saisir ses préférences. Cette phase est interactive et vise à extraire de l'usager le maximum d'informations susceptibles de contribuer à déterminer la catégorie du client, (i.e., client sportif donc il aime les articles de sport, le client aime bien la musique donc il faut s'intéresser aux types de musiques qui lui tiennent à cœur, etc...). Aussi, le système doit fournir des informations sous forme de suggestions afin d'aider l'usager à bien spécifier son profil au système. Par exemple, si le client se trompe dans la saisie d'un article, le système réagira automatiquement en proposant au client l'ensemble des articles à choisir en consultant le dictionnaire d'articles.
Une fois que le client aura saisi toutes ses préférences d'une manière interactive, le système effectue une analyse de ses préférences afin de déterminer la catégorie de client dont il s'agit. Le processus de *catégorisation* du client passe par une activité de raisonnement du système sur la base des préférences qui sont fournies par ce client. En fonction de la catégorie du client et de l'ensemble des autres informations *invariantes* (par exemple, tenir compte de certains événements qui sont invariants pour le système comme le mois courant, la saison actuelle, l'approche des fêtes de Noël, le nouvel an, les fêtes nationales, etc...), le système établit une première version du plan de navigation du magasin. Ce plan comporte un ensemble d'articles à montrer au client. Le plan contient trois sous-plans dont chacun de ces derniers est un ensemble d'articles à afficher. Le premier sous-plan renferme des articles à montrer impérativement car ils répondent parfaitement au profil du client, le second sous-plan renferme un ensemble d'articles qui répondent moyennement au profil du client; le dernier sous-plan contient des articles d'intérêt général (articles invendus ayant fait l'objet d'une promotion, nouveaux articles pas suffisamment connus et ayant fait l'objet d'une publicité, etc...).
La dernière phase consiste à exécuter le plan de navigation en affichant les articles selon des techniques qui garantissent l'ergonomie du logiciel. Le système réagit au comportement du client au moment de l'exécution du plan d'affichage et agit dynamiquement en analysant ses choix et en révisant le plan de navigation si un comportement particulier est décelé chez le client. Cette capacité du système à tenir compte du comportement du client et à réagir en temps réel lui confère un comportement adaptatif. En outre, l'agent interface et l'agent navigateur tiennent compte des comportements antérieurs du client. Conséquemment, ils génèrent de nouvelles préférences en fonction des comportements et "habitudes" antérieurs du client.
L'application renferme 4 agents dont 3 agents spécialistes et un superviseur (voir Fig.6.8). Les spécialistes sont un agent interface, un agent planificateur et un agent navigateur. Le superviseur gère les interactions entre les spécialistes.

Les agents de l'application sont cognitifs car ils ont la capacité de percevoir leur environnement, de raisonner sur leurs croyances et d'établir des plans d'action. De plus, les préférences de l'usager sont manipulées par tous les agents d'où la nécessité de les stocker dans un tableau noir. Tous ces aspects font que MAGIQUE est une architecture adaptée pour supporter

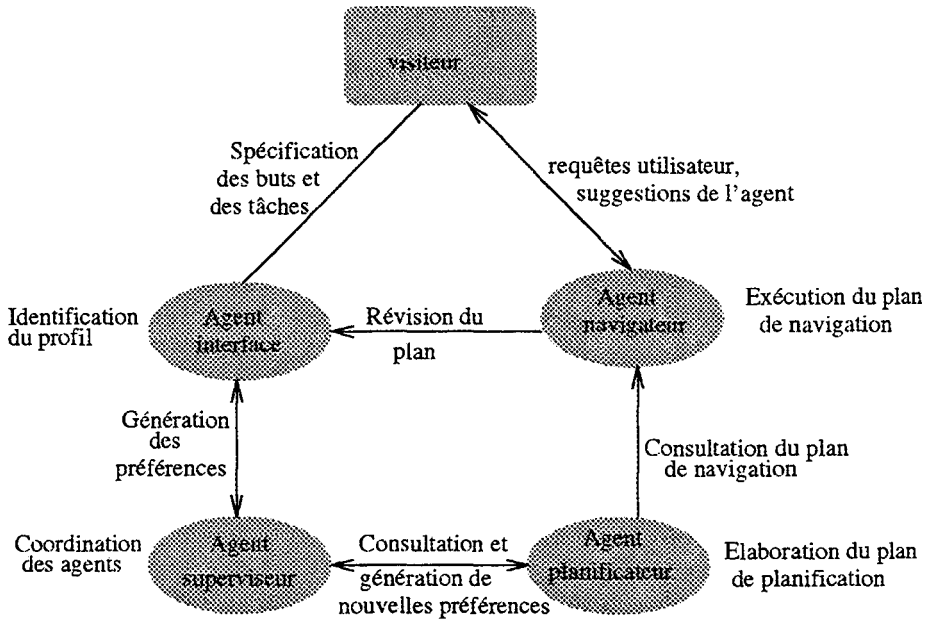

FIG. 6.8 - *Architecture du système*

l'application décrite précédement.

L'agent interface

Cet agent joue le rôle d'interface entre l'usager et le système. Son but consiste à déterminer la catégorie du client et de fournir un ensemble d'informations concernant le client au superviseur. La structure de l'agent interface est la même que celle définie dans la section 4.2 (cf. chapitre 4). L'environnement de l'agent est constitué de l'usager avec lequel il interagit et des autres agents accointances avec lesquels il coopère. Il possède une base de croyances renfermant l'ensemble des préférences du client. Les préférences sont de deux types: des préférences *statiques* qui sont extraites directement de l'usager, et les préférences *dynamiques* qui sont déduites à partir de l'application de la base des règles de croyances sur les préférences statiques. Les préférences dynamiques sont sujets à révision durant le fonctionnement du système. Les capacités de l'agent interface consistent à filtrer les préférences et à déterminer la catégorie du client. Un ensemble de capacités appelées *méthodes de filtrage* sont appliquées successivement afin d'identifier le profil du client. Chaque méthode contient un profil particulier défini sous forme d'un ensemble de propriétés qui le caractérisent. Sur la base des résultats obtenus par l'exécution des méthodes précédentes, un module d'évaluation évalue les résultats obtenus et détermine en conséquence la catégorie du client. Le processus d'évaluation dépend du degré d'unification des résultats d'exécution des méthodes avec les préférences du client. En plus, un ensemble d'actions communicatives permettent à l'agent interface d'interagir avec son environnement et ses accointances. L'ensemble des croyances générées sont rangées dans un tableau noir. En outre, l'agent maintient l'ensemble des profils associés à un client donné et tient compte de l'ensemble des préférences qu'il a exprimées

dans le passé. Le cycle d'exécution de l'agent interface est le suivant:

- Interroger le client (action sur l'environnement). Cette action consiste à extraire les préférences du client.
- Extraire les préférences du client (perception de l'environnement). Cette action saisit les préférences du client et les stocke dans la base des croyances de l'agent.
- Analyser les préférences du client et déterminer sa catégorie.
    1. appliquer les méthodes de filtrage. Ces méthodes sont appliquées séquentiellement et permettent de connaître le degré d'unification des préférences avec le profil représenté par la méthode de filtrage appliquée. Chaque méthode prend en entrée l'ensemble des préférences du client et effectue une unification avec l'ensemble des propriétés qui caractérisent un profil particulier. La méthode rend un entier représentant le degré d'unification des préférences du client avec le profil représenté par la méthode.
    2. Appliquer une fonction d'évaluation des résultats obtenus. Cette fonction permet de sélectionner un ou plusieurs profils à associer au client.
- Ranger les préférences dans le tableau noir du système afin qu'elles soient visibles pour tous les agents.
- Informer l'agent superviseur. Cette opération est une action communicative qui consiste à informer le superviseur afin de coordonner les activités des autres agents. L'action de coordination du susperviseur vise principalement à réagir au changements du tableau noir et à activer les agents spécialistes concernés par ces changements.

Un profil est représenté sous forme d'une paire contenant l'identificateur du profil et une liste de propriétés qui caractérisent ce profil. Les capacités de filtrage vérifient les profils qui sont satisfaits.
L'ensemble des préférences fournies par le client à l'agent interface sont stockées localement au niveau de l'agent afin qu'elles soient consultées dans les sessions ultérieures du système. De ce fait, l'agent tient compte du comportement antérieur du client dans le processus d'extraction des préférences et d'identification du profil.

L'agent planificateur

Cet agent a pour but d'établir un plan de navigation du magasin. Pour cela, il s'appuie sur le type de profil (ou catégorie) du client et l'ensemble des préférences du tableau noir. À l'instar de l'agent interface, l'agent planificateur dispose d'un ensemble de capacités dont l'exécution permet de définir l'ensemble des articles qui intéressent le client. La sélection de ces articles détermine le plan courant de l'agent. En fonction des préférences du client, une fonction de calcul des priorités prend en entrée les préférences du client, l'ensemble des articles à afficher et renvoie une liste ordonnée d'articles. Le plan de l'agent contient trois niveaux. Le premier niveau contient une liste d'articles qu'il va falloir impérativement afficher car ils répondent parfaitement au profil du client. Le second niveau contient une liste d'articles qui répondent moyennement au profil du client, par contre le troisième niveau renferme une liste

d'articles d'intérêt général qui sont proposés par les experts commerciaux du magasin afin de faire une promotion ou encore des articles nouveaux que l'on souhaite faire connaître au grand public.
Pour élaborer son plan, l'agent s'appuie aussi sur un certain nombre d' informations que l'on appelle des invariants comme par exemple, la saison courante, l'approche des fêtes réligieuses, les fêtes nationales, etc...
Ces invariants sont combinés avec les préférences du client afin d'avoir une masse d'informations conséquente permettant à l'agent de proposer un plan de navigation.
Le cycle de l'agent consiste à consulter dans le tableau noir l'ensemble des préférences générées par l'agent interface, appliquer sa base des règles de croyances sur les préférences et les invariants afin de générer de nouvelles croyances et finalement appliquer séquentiellement les capacités de filtrage qui permettent de générer les articles susceptibles d'intéresser le client.
Les méthodes de filtrage associent des coefficients de vraisemblance aux articles proposés afin d'établir un ordre d'affichage. Par exemple, pour sélectionner un article pour qu'il soit affiché, il faut qu'un certain nombre de croyances soient vérifiées. Le coeficient de vraisemblance associé à l'article est proportionnel au nombre de croyances qui sont satisfaites. Le cycle d'exécution de l'agent est le suivant:

- Consulter les préférences du client dans le tableau noir: Cette opération est une action de perception de l'environnement de l'agent car le superviseur peut être considéré comme un agent autonome qui peut fournir des informations aux autres agents.
- Appliquer la base des règles de croyances en tenant compte des préférences du client et des invariants. Les invariants du système sont mis à jour périodiquement en réagissant à des événements qui proviennent des autres entités comme le système d'exploitation et autres agents.
- Générer tous les articles qui satisfont les profils du client. Le plan de navigation du client est un sous ensemble de ces articles.
- Appliquer séquentiellement les capacités de filtrage des articles. Ces capacités prennent en entrée un ensemble de préférences et génèrent un ensemble d'articles pondérés par des coefficients. Les articles proposés sont donc caractérisés par le degré de *satisfaction* qui va jouer un rôle sur le rang de l'article dans le plan.
- Appliquer la fonction d'ordonnancement: cette fonction prend en entrée l'ensemble des articles sélectionnés et leur coefficient de vraisemblance et retourne une liste ordonnée d'articles.
- Communiquer à l'agent superviseur la liste des articles à afficher. Cette opération est une action communicative entre le spécialiste et le superviseur.

Le plan proposé par l'agent de planification constitue une vue abstraite du magasin, adaptée au profil du client.

L'agent navigateur

Cet agent a pour but de faire naviguer le client dans le magasin virtuel. L'affichage des articles se fait d'une manière séquentielle en respectant l'ordre des articles dans le plan.

L'agent consulte le tableau noir afin de récupérer le plan de navigation, avant de commencer l'affichage des articles. Le processus d'affichage d'un article se traduit par une série d'actions interactives de l'agent avec le client. Sur demande du client, l'agent aura à fournir une série d'informations supplémentaires concernant l'article. Les requêtes du client sur l'interface de l'agent pourrait se traduire par des interactions de l'agent navigateur avec d'autres agents afin de fournir les informations requises. Par exemple, pour un survêtement de sport, le client peut demander quelle sont les matières qui sont à l'origine de la composition de ce produit. Dans ce cas, l'agent navigateur transmettra une requête à un agent d'information (base de donnée des articles) afin de satisfaire la requête du client.
Aussi, en fonction des suggestions de l'agent navigateur, les choix du client peuvent engendrer une révision dynamique du plan de navigation. Par exemple, si le client s'intéresse à un aspect particulier, ou à une propriété particulière des articles, l'agent navigateur intégrera cette propriété comme préférence du client et informera l'agent planificateur afin de réviser le plan de navigation et de tenir compte de la nouvelle préférence. Le cycle d'exécution de l'agent est le suivant:

- Consulter le plan au sein du tableau noir. Cette opération est une action communicative car elle met en œuvre une interaction de deux agents.

- Exhiber un plan d'affichage de l'article. Cette opération consiste à proposer une maquette d'affichage de l'article avec toutes les propriétés inhérentes à l'article: la couleur choisie, le mode d'affichage, l'environnement de l'article, etc...
  La maquette contient aussi un ensemble de suggestions a fournir si le client souhaite un complément d'informations concernant l'article.

- Réagir aux choix du client. Cette opération pourrait se traduire par un processus de révision du plan de navigation si de nouvelles préférences peuvent être extraites.

- Passer à l'article suivant et réitérer le processus.

L'adjonction d'un ensemble de suggestions que l'agent navigateur met à la disposition du client participe du souci de rendre flexible l'interaction entre le système et l'utilisateur. L'approche multi-agent est ainsi utilisée pour accroître l'ergonomie du logiciel en facilitant l'interaction entre l'homme et la machine.

Pourquoi utiliser MAGIQUE?

L'établissement du plan dépend des connaissances générées respectivement par l'agent interface (préférences statiques et dynamiques), par l'agent planificateur (changement des invariants) et par l'agent navigateur (détection d'une situation au niveau du comportement de l'usager, qui nécessite une révision dynamique du plan). Toutes ces connaissances sont centralisées au niveau du tableau noir et constituent le contexte courant pour la construction du plan (voir Fig. 6.9). En plus, le tableau noir contient la version courante du plan. Le superviseur se charge d'activer les agents et de veiller à la bonne cohérence du plan durant sa révision. Les changements effectués dans le tableau noir sont scrutés par l'agent superviseur.

L'agent superviseur

L'objectif du superviseur est de gérer les interactions entre les agents spécialistes du système. Il est caractérisé par le tableau noir qui renferme l'ensemble des préférences du client ainsi que la version courante du plan de navigation du magasin. Le superviseur dispose d'un ensemble de capacités qui sont déclenchées si certaines de ses croyances sont vérifiées. Ces capacités consistent à réagir aux changements effectués dans le tableau noir et à activer en conséquence les agents spécialistes. Par exemple, dès que l'agent interface aurait identifié le profil du client, le superviseur active l'agent planificateur. Le cycle d'exécution du superviseur est le suivant:

- Détecter des configurations particulières du tableau noir.
- Activer les spécialistes intéressés par ces configurations.

Le tableau noir contient 5 niveaux d'abstraction. Le premier niveau contient les préférences du client générées par les agents interface et planificateur, le second niveau contient des articles correspondant aux profils du client, le troisième niveau fournit une première partie du plan contenant une liste d'articles qui répondent parfaitement au profil du client, suivi de la seconde partie contenant des articles répondant moyennement au profil du client, et enfin d'un dernier niveau renfermant les articles d'intérêt général proposés par le magasin à sa clientèle.
Typiquement, le superviseur est un agent qui perçoit de l'information en provenance des agents de son groupe. Cette connaissance est rangée dans son tableau noir qui représente l'état courant du processus de résolution. Le superviseur réagit aux changements suscités par les spécialistes en envoyant des requêtes d'activation aux agents concernés par ces changements.

Coopération entre les agents

La coopération entre les agents du système implique un processus d'interaction entre ses agents, du début du processus de résolution jusqu'à sa fin. Après avoir atteint son but, l'agent interface informe l'agent planificateur du (ou des) profil(s) du client. Dans le cas de l'architecture MAGIQUE, cette interaction est perçue comme étant de la coopération horizontale entre spécialistes. Une autre interaction horizontale entre l'agent navigateur et l'agent planificateur dans le cas ou une révision du plan courant est requise. En ce qui concerne la coopération verticale, elle se résume aux accès au tableau noir du superviseur afin de mettre à jour les préférences ainsi que les informations invariantes du système. Le superviseur joue son rôle classique de mécanisme de contrôle du tableau noir.
Les interactions entre les agents sont engendrées aussi par le processus de révision du plan courant de navigation. En effet, les préférences d'un même client sont stockées et sont consultées à chaque fois qu'il lance une session de l'application. Durant la navigation (exécution du plan par l'agent navigateur), si l'agent détecte un comportement particulier chez le client, il envoie une requête à l'agent interface. Ce dernier applique sa base de règles des croyances et génére de nouvelles croyances ayant trait aux préférences du client. Par conséquent, l'agent planificateur est informé à son tour ainsi que le superviseur. L'agent planificateur tient compte de la partie du plan qui a été exécutée, et élabore ainsi son nouveau plan en continuité de la partie exécutée du plan précédent.
Finalement, la révision d'un plan de navigation engendre un cycle entier de tous les agents et par ricochet un cycle de tout le système. Ce processus de révision confère au système un

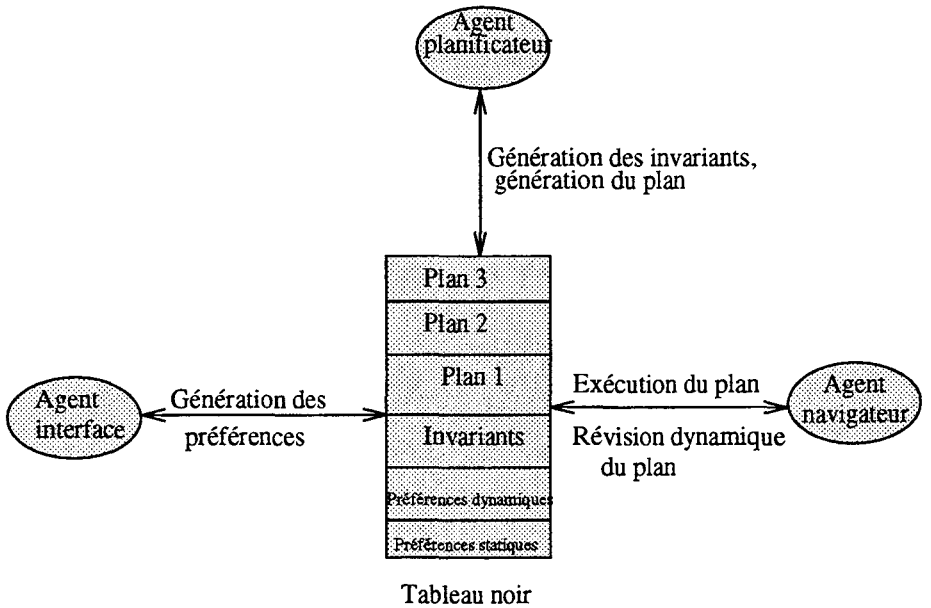

FIG. 6.9 - *Coopération via le tableau noir*

comportement adaptatif et permet d'appréhender des situations dynamiques de l'environnement.
De plus, l'utilisation d'une structure tableau noir pour représenter le contexte courant de la solution, ainsi que les capacités de haut niveau des agents en terme de raisonnement (identification des profils), de planification (élaboration du plan par l'agent planificateur) et d'interaction avec l'utilisateur, nécessitent l'utilisation des agents intelligents et justifient en conséquence l'utilisation de l'architecture MAGIQUE.

Réalisation

Les applications client/serveur classiques supposent que les traitements du client sont effectués au niveau du site serveur, d'où le caractère centralisé du système supportant l'application. Ce dernier aspect induit un problème de fiabilité du système vu que le serveur peut être considéré comme une ressource critique et partagée. Afin de pallier à ce dernier inconvénient, il convient de distribuer les traitements entre le serveur et les clients qui deviennent ainsi des entités "intelligentes". L'application que nous proposons est constituée de 4 agents dont un agent superviseur et 3 agents spécialistes (un agent interface, un agent planificateur et un agent navigateur). L'agent interface et l'agent navigateur interagissent directement avec le client via une interface. Par conséquent, selon le modèle d'application client/serveur, ils se comportent comme des clients dont les traitements sont effectués au niveau de la machine du client. Ces deux agents utilisent un mécanisme basé sur les *cookies* qui leur permet de sauvegarder et de consulter des informations sur la machine du client, ce qui autorise l'agent interface par exemple à raisonner sur le comportement antérieur du client. La notion de cookies est un mécanisme général que les connexions du côté serveur (tels que des scripts CGI[4])

4. Common Gateway Interface

peuvent utiliser pour sauvegarder et consulter des informations du côté client de la connexion. Par exemple, quand un serveur retourne un objet HTTP à un agent client, il peut lui envoyer aussi une information à sauvegarder.
À l'inverse des agents interface et navigateur, l'agent planificateur et l'agent superviseur s'exécutent sur la machine du serveur. À chaque fois qu'un client se connecte au système via un quelconque navigateur (par exemple, netscape), le serveur déclenche une applet qui correspond à l'agent interface. Ce dernier sera exécuté sur la machine du client. L'ensemble des préférences générées par l'agent interface sont transférées à l'agent superviseur afin qu'elles soient exploitées par l'agent planificateur pour l'élaboration du plan de navigation.
La notion de cookies est un outil qui permet de développer des applications multi-agents physiquement distribuées dont les différents agents du système s'exécutent sur des machines distinctes. L'objectif est de décharger le site du serveur et d'accroître en conséquence l'efficacité du système.

Avantages

Les avantages de l'application sont:

- Efficacité: les agents interface et navigateur s'exécutent sur la machine du client, ce qui décharge considérablement la machine du serveur supportant initialement tous les traitements, aussi bien ceux du client que ceux liés à la gestion des multiples sessions parallèles. Par conséquent, le système s'exécute en parallèle sur plusieurs machines distantes géographiquement, d'où une amélioration certaine des performances du système.

- Fiabilité: les applications client/serveur souffrent d'un grand handicap: la tolérance aux pannes. La raison est que tous les traitements, aussi bie ceux des clients que ceux du serveur sont supportés par la machine serveur. Dans notre approche, la distribution des traitements apporte une amélioration importante au niveau de la fiabilité du système car si une panne survient au niveau d'un agent interface interagissant avec un client donné, elle n'affecte pas les autres sessions du système.

- Adaptativité: le système permet de s'adapter aux besoins du client en tenant compte de ses comportements antérieurs, et en générant un plan de navigation adapté à son profil. Ainsi, le système gagne en flexibilité car ces agents ont un comportement opportuniste (les actions exécutées dépendent de l'état mental de l'agent) et sont capables aussi d'assister un utilisateur en lui proposant une panoplie de choix multiples (l'agent navigateur a pour rôle d'assister l'utilisateur dans ces interactions avec le système).

Conclusion

Cette application montre que des applications client/serveur physiquement distribuées peuvent être supportées par l'architecture MAGIQUE. Contrairement aux applications client/serveur classiques ou le client est réduit à une simple interface, l'application présentée supporte des clients ayant des fonctionnalités complexes, représentés par des agents cognitifs de notre architecture. Ainsi, nous pouvons conclure que MAGIQUE est une architecture adaptée pour supporter des apllications client/serveur complexes et adaptatives sur Internet. Les avantages discutés dans la section précédente renseignent sur les performances escomptées.

L'un de nos objectifs est d'étendre l'architecture de base de MAGIQUE afin de prendre en compte les aspects Internet en intégrant le mécanisme de cookies qui permet aux agents du site serveur et à ceux du site client de s'échanger facilement des informations. Le but est de faciliter l'instanciation d'autres applications comme celles qui consistent à fournir à un client distant géographiquement d'avoir un plan de visite d'une région ou d'une ville donnée ou encore d'un musée. Ainsi, le système fournira à l'usager un service de très haut niveau lui permettant de visiter un musée situé par exemple aux USA ou au Japan. Un adepte du tourisme aura une bonne idée des vestiges et sites historiques de la ville de Paris, etc...

Annexe A

Contrôleurs de tableau noir

Préambule:

Cette annexe présente l'ensemble des modèles de contrôle dans les architectures tableau noir. Dans chaque modèle de contrôle, les connaissances de contrôle, le problème de leur représentation ainsi que les modules du système sont examinés.

Le premier modèle de contrôle se rapporte au contrôle procédural dont l'archétype est le système HEARSAY-II. Cette architecture de contrôle incarne le contrôle centralisé dans les systèmes tableau noir.

Les autres types de contrôle ont trait aux systèmes BB1, CRYSALIS et ATOME. Les spécificités, les avantages ainsi que les faiblesses de chaque type de contrôle sont présentés et discutés.

A.1 Contrôle procédural

Une architecture tableau noir à base d'un contrôle procédural repose sur l'idée que le processus de contrôle se réduit à un programme unique et complexe manipulant les connaissances de contrôle tant sur le contenu que sur la façon de les utiliser. L'entité de contrôle contient les éléments suivants: *l'ordonnanceur*, le *moniteur* et une structure de donnée appelée *agenda*.
L'agenda contient la liste des modules déclenchables. Ce sont les KSARs.
L'ordonnanceur sélectionne le prochain module à exécuter en fonction de l'état global du tableau en s'appuyant sur des heuristiques de contrôle. Ainsi, il utilise des procédures d'évaluation des heuristiques sur les KSARs de l'agenda en consultant le tableau noir. L'évaluation des heuristiques permet de déterminer le KSAR le plus prioritaire qui sera choisi pour être déclenché.
Le moniteur a pour rôle de détecter les modules déclenchables après chaque événement et

met à jour l'agenda.
Il faut souligner que les stratégies de contrôle sont déterminées de façon implicite par l'activation des procédures évaluant les heuristiques. En revanche, le comportement du système est opportuniste car il dépend de l'état courant du tableau noir.

A.2 Le système HEARSAY-II

Le système HEARSAY-II est conçu à l'université du Carnegie Mellon (CMU), suite à un appel d'offre du ministère de la défense américain. Il est destiné à la compréhension de la parole. La tâche principale du système consiste à reconnaître des énoncés oraux d'un vocabulaire en un temps proche du temps réel.
Bien que n'ayant pas satisfait l'objectif assigné en terme de performances, HEARSAY-II a mis en avant de nouveaux concepts de programmation et de représentation des connaissances. Ces nouveaux concepts ont donné naissance à un nouveau paradigme de système qui est: l'architecture de tableau noir comme modèle de résolution de problèmes. Par la suite, l'architecture a été appliquée à plusieurs domaines d'application dont les performances se sont avérées intéressantes.

A.2.1 Connaissances de contrôle

Les objets et les solutions: Dans la réalité, les connaissances sont caractérisées par l'imprécision et des erreurs sont inhérentes aux données. Afin d'adapter ces connaissances pour qu'elles soient utilisées par le système tableau noir, il faut successivement représenter les données initiales sur plusieurs niveaux d'abstraction jusqu'à ce que le niveau atteint corresponde à celui de la solution désirée. Le passage d'un niveau à un autre est géré par une source de connaissances experte: le module. Néanmoins, si le module est un expert, sa contribution en terme de résolution se rapporte seulement à une partie du problème, la solution partielle. Le traitement de données partielles engendre plusieurs interprétations: ces données sont ambiguës et sont susceptibles d'être comprises différemment sous le point de vue du niveau de sortie du module. Des ambiguïtés artificielles dues à la distribution des connaissances linguistiques en modules ont été signalées dans [Rad83]. Elles restent un problème central des techniques classiques de compréhension de la langue.
La séparation des connaissances en niveaux hiérarchiques différents permet de traiter des données erronées, ambiguës et partielles. En revanche, des solutions partielles supplémentaires s'ajoutent aux ambiguïtés déjà inhérentes au problème.

Les modules: Dans le système HEARSAY-II, les sources de connaissances sont décrites par le formalisme condition/action. La partie condition indique si l'état courant du tableau permet de créer des hypothèses stipulées par la partie action. Dans le cas où la partie condition est satisfaite, la source est exécutable. Cette dernière comprend, aussi, deux champs supplémentaires: le *stimulus frame* qui comprend les hypothèses du tableau ayant satisfait la partie condition et le *response frame* qui contient une vue des travaux potentiels du module.
HEARSAY-II est caractérisé par son contrôle procédural. Il utilise un ordonnanceur, un moniteur et un agenda. Le moniteur scrute les modules dont les parties condition sont satisfaites par les événements opérés dans le tableau, et les stocke dans l'agenda. Il utilise une table définie a priori, dont le rôle est d'associer pour chaque événement possible du tableau les modules concernés. Il met à jour, aussi, les changements effectifs du tableau dans une structure

de contrôle.
Ainsi, l'agenda comprend deux types de possibilité: une partie condition d'un module gérée par le moniteur et une partie action gérée par l'ordonnanceur. Ce dernier choisit la possibilité la plus prioritaire et la déclenche. Si la possibilité se trouvant dans l'agenda est une partie condition et qu'elle est vérifiée, les champs *stimulus frame* et *response frame* sont mis à jour et la partie action est placée dans l'agenda. Par contre, si la possibilité est une action, son exécution engendre une modification du tableau noir se traduisant par la création d'hypothèses. Finalement, les KSARs sont, soient des parties conditions dont le déclenchement consiste à les tester, soit des parties action qu'il va falloir exécuter. Le choix d'un KSAR à un instant donné repose sur un certain nombre d'heuristiques, énumérées comme suit:

- Compétition : cette heuristique choisit un module parmi plusieurs autres travaillant sur des hypothèses compétitives. Le choix porte sur le module dont le champs *response frame* comporte les hypothèses, de plus haut niveau d'abstraction, de la plus grande durée et aussi les plus fiables.

- Crédibilité : cette heuristique retient parmi les *stimulus frame* d'un même module déclenchable sur plusieurs configurations d'objets, celui contenant les hypothèses les plus crédibles. Les modules peuvent souligner la confiance qu'ils attachent aux hypothèses qu'ils émettent.

- Importance : cette heuristique consiste à retenir les modules de l'agenda ayant le champs *response frame* le plus important. Il faut noter que l'importance d'une action est une fonction croissante du niveau d'abstraction.

- Efficacité : cette heuristique retient le module dont le déclenchement est le plus efficace.

- Satisfaction du but : cette heuristique choisit le module dont le champs *response frame* satisfait le mieux les objectifs visés ainsi que les buts poursuivis à un moment donné du processus de résolution.

Les heuristiques précédentes sont représentées par des valeurs numériques et sont utilisées par des procédures de l'ordonnanceur.
Dans HEARSAY-II, l'entité de contrôle n'a aucune stratégie d'utilisation des connaissances de contrôle. L'ordonnanceur suit le résultat des votes à l'aide des heuristiques.

A.2.2 Représentation des connaissances de contrôle

En fonction des différentes implémentations de HEARSAY-II, le nombre de niveaux hiérarchiques d'abstraction varie. On peut citer la hiérarchie suivante:

Signal paramétrisé, segment, syllabe, mot, séquence de mots, phrase et accès à une base de données.

Toutes les données initiales sont dispatchées à chaque niveau sous un certain point de vue. Les objets du tableau sont des solutions partielles ayant une structure attribut/valeur. Cette structure est valable pour tous les objets et à tous les niveaux. Il faut noter que certains attributs sont obligatoires et identiques pour tous les objets, comme le nom du niveau de l'objet

et les données décrites par l'objet. Dans HEARSAY-II, un niveau définit un vocabulaire qui lui est propre. Les solutions partielles sont définies comme étant une interprétation possible de données à un certain niveau d'abstraction. Ainsi, toutes ces relations sont représentées explicitement sur les objets sous la forme de paires attribut/valeur. Une information importante sur un objet est accessible directement par une simple lecture des attributs de cet objet.
La séparation des connaissances en modules experts engendrent la création d'hypothèses concurrentes, i.e., des objets décrivant les mêmes données sur le même niveau. Les relations entre modules se réduisent alors à la coopération, i.e., des objets ne décrivent aucune donnée commune, ou à la compétition, i.e., des objets décrivent des données partagées.

Vu qu'il n'existe pas de stratégies d'utilisation, le comportement des modules est transcrit de façon implicite au niveau des heuristiques de contrôle. La boucle de base consiste donc à effectuer séquentiellement les opérations suivantes:

- Le moniteur détecte les modules concernés par les changements opérés dans le tableau noir. Ensuite, il insère les parties condition des modules sélectionnés, comme nouveau KSARs dans l'agenda.

- L'ordonnanceur applique les heuristiques pour déterminer le meilleur KSAR.

- Le KSAR choisi est déclenché. Si le KSAR correspond à une partie condition alors cette condition est testée et les champs *stimulus frame* et *response frame* du module sont mis à jour et stockés dans l'agenda avec la partie action du module. S'il correspond à une partie action alors les changements qu'elle représente sont effectués sur le tableau.

A.2.3 Conclusion

Le contrôle est caractérisé essentiellement par son efficacité et son opportunisme. En revanche, des problèmes se posent comme le chevauchement de données d'objets de même niveaux. Cela provient du fait que des relations entre objets ne sont pas formalisées. Des relations comme la redondance, la contradiction et l'incompatibilité ne sont pas définies clairement et dépendent en grande partie des connaissances du domaine. Pour y remédier, il est nécessaire d'harmoniser l'inférence avec la compétition des objets. D'un autre coté, les hypothèses sont mal distinguées des déductions auxquelles elles appartiennent. Les modules déterminent difficilement leur éventuelle contribution, ainsi que le travail qui reste à faire et qui nécessite l'examen de tous le réseau et non seulement les objets sur lesquels ils sont déclenchables.
La représentation des connaissances est presque inexistante. Les connaissances sont donc difficiles à exprimer car il faut les coder en dur. Comme conséquence de la faiblesse de la représentation, la mise en œuvre des modification des connaissances devient délicate et difficile à mettre en œuvre. Pour pallier à ces inconvénients, il est conseillé de faire une déclaration explicite et modulaire des connaissances de contrôle.
Finalement, le contrôle procédural est adapté pour des applications où les connaissances de contrôle sont bien définies et organisées, et d'autre part où le fonctionnement du système est simple afin que les choix demeurent implicites.

A.3 Le système CRYSALIS

CRYSALIS [Ter88][Nii86b] a été conçu entre 1976 et 1983 pour reconnaître la structure en trois dimensions des protéines, à partir de leur séquence d'acides aminés et de la diffraction obtenue par rayons X sur la protéine cristallisée. Ce projet a été le fruit d'une collaboration entre des experts en cristallographie de l'université de San Diego et des chercheurs en IA de Stanford travaillant sur le projet "Heuristic Programming Project".
Le système CRYSALIS illustre parfaitement l'utilisation des connaissances de contrôle explicites. Il repose sur une déclaration explicite des connaissances de contrôle en autorisant une stratégie d'utilisation figée. Ce modèle est caractérisé par le fait que toutes les connaissances du domaine ou de contrôle sont organisées en modules appartenant à une même hiérarchie dont les modules du domaine constituent les feuilles. L'activation d'un module appartenant à un niveau donné induit le déclenchement de certains modules du niveau inférieur.

A.3.1 Connaissances de contrôle

CRYSALIS se distingue des autres systèmes tableau noir par le fait qu'il utilise plusieurs panneaux dans le tableau (voir Fig.A.1). Ce dernier comprend deux panneaux: le panneau densité destiné à contenir les données et le panneau hypothèse contenant les résultats des traitements effectués par les modules.

Panneau densité : il est structuré en niveaux hiérarchiques représentant les multiples représentations des données. La diversité des formalismes de représentation des données a motivé l'introduction d'une seconde hiérarchie dans le tableau. Les niveaux sont: *points, pics, squelette* et *segment.*

Panneau hypothèse : il est structuré aussi en niveaux hiérarchiques renfermant des objets. Ces derniers sont générés à partir d'hypothèses présentes sur le panneau et de données du panneau précédent. Les niveaux sont appelés respectivement: *atomes, superatomes* et *stéréotypes.*
Durant le processus de résolution, CRYSALIS crée des objets sur le panneau des hypothèses sans modifier celui de la densité. Ce dernier est construit a priori de façon totalement ascendante par des algorithmes manipulant des objets d'un niveau pour produire les objets au niveau immédiatement supérieur. Le système effectue ses inférences internes des données vers les hypothèses, au sein même du panneau des hypothèses. Le résultat est fourni par les objets appartenant au niveau *atomes.* Ce dernier fournit des atomes dans une structure tridimensionnelle. Les opérations sur le panneau des hypothèses consistent à affiner progressivement des hypothèses depuis les stéréotypes jusqu'aux atomes. Alors que la solution dans le cas du système HEARSAY-II est donnée par les objets les plus abstraits, dans CRYSALIS, la solution est donnée par les objets les plus spécialisés du panneau hypothèse.

Modules: les connaissances de contrôle sont organisées en modules ayant une structure hiérarchique. Cette structure possède trois niveaux. Le module de la racine comporte des stratégies d'utilisation. Le niveau intermédiaire contient des modules appelés *tâches.* Ces dernières possèdent une partie détection indiquant les événements intéressants, et une partie contenu indiquant la coopération possible des modules sur les événements qui ont été détectés.

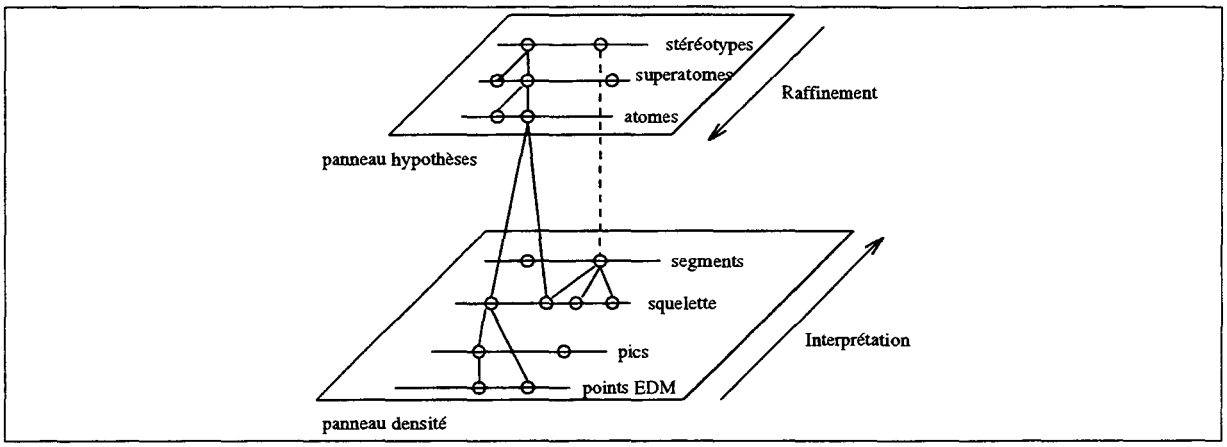

FIG. A.1 - *Les panneaux de CRYSALIS*

Les éventuelles contributions des modules du domaine sont représentées par une liste d'événements contenant l'ensemble des changements, dans le tableau, qui n'ont pas été encore exploités. La tâche a pour rôle d'activer séquentiellement des modules du domaine en fonction des événements se trouvant dans la liste d'événements et du foyer d'attention courant transmis par la stratégie.
Comme conséquence de taille d'une telle organisation du contrôle, les modules du domaine ne possèdent pas le formalisme condition/action. La détection des contributions que les modules peuvent apporter est effectuée par les tâches. Ce nouveau aspect introduit dans CRYSALIS ne respecte pas le modèle fondamental des tableaux noirs, ce qui a poussé Penny Nii à considérer CRYSALIS comme un *blackboard-like system* [Nii86b].
Les tâches représentent le schéma de coopération des modules du domaine en proposant une séquence de modules à activer.
Le module de contrôle se trouvant au niveau de la racine de la structure hiérarchique de contrôle est appelé *stratégie*. Son rôle consiste d'une part à localiser un foyer d'attention contenant les objets intéressants, et d'autre part fournit une stratégie de planification en activant une séquence de tâches. La focalisation d'attention est basée sur une structure de contrôle appelée: *la liste des formes*. Son rôle consiste à visualiser la chaîne des acides aminés et le résultat de l'analyse de leur répartition spatiale dans la molécule. La stratégie consulte la liste des formes pour estimer l'état de la résolution et détermine, en conséquence le plan à suivre. La stratégie choisit un foyer d'attention et le transmet aux modules du niveau inférieur.

A.3.2 Connaissances de contrôle

Les données consignées dans le tableau noir ont une structure attribut/valeur. Les solutions sont des hypothèses appartenant au niveau le plus spécifique du panneau hypothèse. Les hypothèses concurrentes sont des objets appartenant au même niveau d'abstraction et décrivant une même région de données.
Ainsi, la structure hiérarchique permet de représenter explicitement et d'une manière modulaire les connaissances de contrôle. Le cycle de base de CRYSALIS est le suivant:

- Mise à jour des événements lors des changements survenus sur les panneaux du tableau noir, ainsi que la liste des formes.

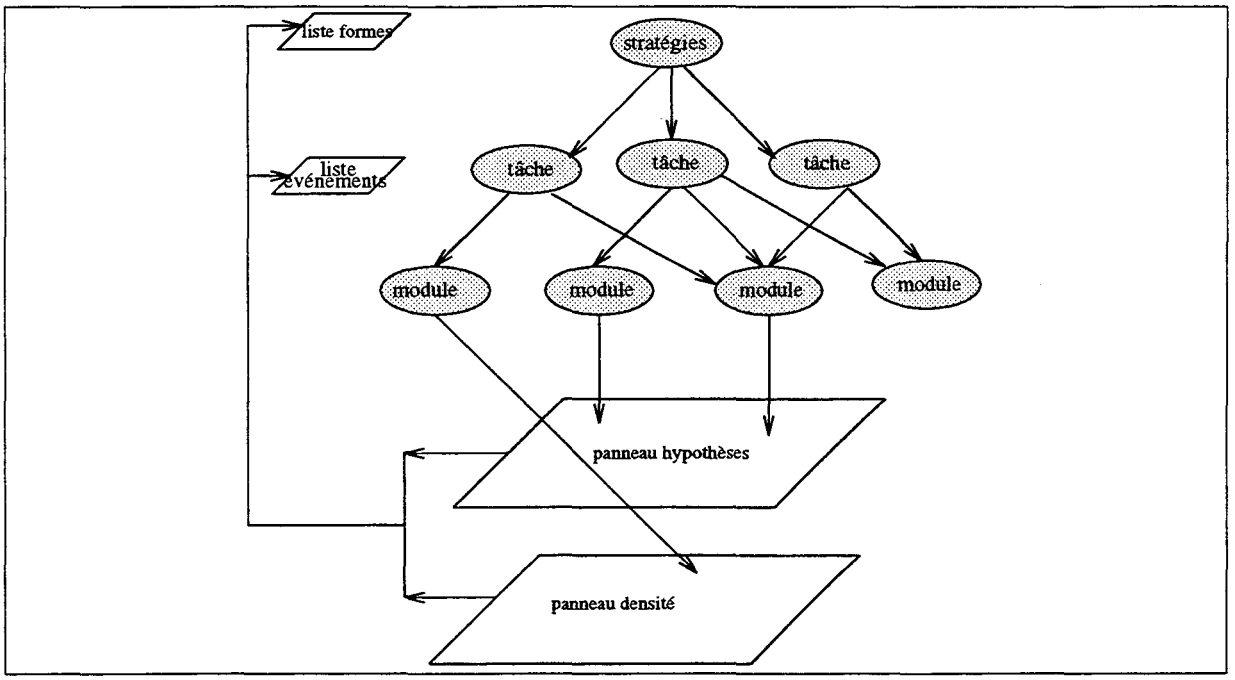

FIG. A.2 - *Architecture de CRYSALIS*

- Consultation par la stratégie de la liste des formes afin de déterminer le foyer de focalisation d'attention.
- La stratégie appelle une séquence de tâches et leur transmet le foyer comme contexte.
- Les tâches activées consultent leur liste d'événements et déclenchent les modules en fonction du contenu de la liste d'événements.
- Les modules appelés par les tâches sont déclenchés.

A.3.3 Conclusion

CRYSALIS illustre un type de contrôle où la représentation des connaissances de contrôle est explicite et modulaire (voir Fig.A.2). Pour des applications où les flux de contrôle sont bien maîtrisés, les connaissances stratégiques sont reconnues facilement. En d'autres termes, la détection des sous-problèmes et des actions à effectuer pour les résoudre est bien connue. En revanche, il se trouve des applications où l'expression des connaissances est moins évidente car elles ne sont pas disponibles. Il faut, aussi, souligner une utilisation efficace des connaissances à cause de deux caractéristiques du système: la focalisation d'attention sur une région donnée du tableau et l'absence d'heuristiques complexes à calculer sur les KSARs. En revanche, le système résultant n'est pas opportuniste car l'utilisation des connaissances est explicite mais figée et obéit donc davantage à des stratégies préétablies qu'à des considérations sur l'état courant du tableau. Comme conséquence, il n'existe pas réellement de KSARs. La détection et la détermination des contributions des modules sont effectuées au niveau des tâches. Afin que le comportement recouvre la propriété d'opportunisme, les stratégies d'utilisation doivent être dynamiques et souples, tout en conservant la propriété d'explicitation des connaissances de contrôle.

A.4 Le système BB-1

Dans BB-1 [HR85], un second tableau est dévolu exclusivement au contrôle. L'idée repose sur le fait que les tableaux noirs permettent de suivre la résolution de problèmes de façon explicite, en faisant interagir des modules hétérogènes. Les modules du domaine construisent sur le tableau noir du domaine des solutions au problème posé. En revanche, les modules de contrôle construisent sur le tableau noir de contrôle un plan de contrôle pour mener à bien le processus de résolution. Les activités de tous les modules du domaine et du contrôle sont enregistrées dans un même agenda. L'ordonnanceur choisit l'activité la plus prioritaire et la déclenche.

A.4.1 Les objets et les solutions

Le tableau noir du domaine est analogue à celui des autres systèmes tableau noir. Il est structuré en niveaux d'abstraction contenant des objets de la forme attribut/valeur. De la même manière, le tableau noir de contrôle est organisé en niveaux hiérarchiques d'abstraction comprenant des objets appelés *décision de contrôle*, de type attribut/valeur. Les niveaux sont de deux types: ceux qui contiennent des décisions sur ce qui doit être fait, et ceux renfermant des décisions sur ce qui est faisable. Les niveaux concernant les décisions sur ce qui doit être fait sont les suivants:

Le problème : c'est le niveau où le problème est représenté. Les attributs des objets de ce niveau initialisent le processus de résolution en déclenchant au moins un module du domaine ou un module de contrôle.

Les stratégies : elles représentent des plans généraux du comportement que doit suivre le processus de résolution. Il peut y avoir plusieurs stratégies concurrentes qui peuvent opérer simultanément.

Les focalisations : ce sont des buts locaux inhérents au processus de résolution. Généralement, une séquence de buts sert à coder une décision stratégique. Les buts sont temporaires et peuvent être en compétition s'ils traduisent des décisions contradictoires. Les décisions sont utilisées pour calculer des coefficients à associer aux KSARs. Ces coefficients sont interprétés par l'ordonnanceur pour choisir le prochain KSAR à déclencher.

Les heuristiques : elles établissent des critères de classification des KSARs en fonction de leur attributs et de leurs valeurs. Les heuristiques sont valables durant tout le processus de résolution.

Agenda : c'est une structure de données contenant tous les KSARs créés. Il comprend deux listes:

- Une liste de KSARs sélectionnés, i.e., les modules dont la partie **déclencheur** est satisfaite par les changements ayant eu lieu dans le tableau.
- Une liste de KSARs activables, i.e., les modules ayant leur partie **pré-condition** satisfaite. La partie pré-condition dépend de l'état courant du tableau.

Le KSAR choisi possède la plus forte priorité et fait partie de la liste des KSARs activables.

A.4.2 Modules

À l'instar des modules du domaine, les modules de contrôle ont une structure condition/action. La partie condition contient une partie déclencheur, dont le rôle est de reconnaître si le module est concerné par les changements survenus dans le tableau. L'autre partie est une pré-condition qui reconnait si l'état du tableau permet d'activer le module. Dans BB-1, la coopération entre modules est dynamique car elle repose sur l'écriture des décisions dans le tableau noir. Les stratégies d'utilisation sont aussi des décisions proposées sur le tableau de contrôle.

A.4.3 Connaissances de contrôle

La représentation des connaissances dans BB-1 est uniforme. La structure d'un module est la suivante:

- Condition: il comprend deux champs : un déclencheur pour reconnaître les événements concernant le module. L'autre champs est la pré-condition dont le rôle consiste à vérifier que l'état du tableau est adéquat pour déclencher la partie action.
- Action: c'est une procédure contenant les changements à apporter aux tableaux.
- Variables conditions: ce champs décrit les variables utilisées dans la partie condition.
- Variables planification : ce champs décrit les variables liées à la création du KSAR et utilisées pour la planification du KSAR.

Les modules du domaine et ceux de contrôle créent les mêmes KSARs. La structure des KSARs créés est la suivante:

- Module: c'est le nom du module concerné par l'événement.
- Événement: nom de l'événement ayant déclenché le module.
- Décision: c'est l'objet du tableau concerné par l'événement.
- Valeur condition : c'est le contenu des variables spécifiées dans le module et qui seront utilisées dans la partie action.
- valeur déclencheur: c'est le contenu des variables de planification du module et qui seraient utilisées dans la planification du KSAR.

BB-1 utilise le formalisme habituel des objets. Ainsi, dans les deux tableaux, un objet représente des données à un certain niveau d'abstraction. Un objet a une structure de la forme attribut/valeur.
Les modules de contrôle sont spécifiques au contrôle du domaine ou indépendants du domaine. Par conséquent, le nombre de sources de connaissances de contrôle dépend de l'application.
Le cycle de base de contrôle est constitué des modules suivants: *mettre à jour l'agenda, choisir le KSAR et l'interpréter.*

A.4.4 Conclusion

BB-1 est un système totalement déclaratif. Il possède des connaissances explicites dont l'utilisation découle des décisions qu'il crée sur le tableau de contrôle. Les avantages de l'architecture de contrôle de BB-1 sont les suivants:

- Les connaissances de contrôle sont utilisées explicitement.
- Contrairement à l'ordonnanceur de HEARSAY-II, celui de BB-1 est très simple.
- Les modules tant du domaine que ceux du contrôle sont indépendants les uns des autres. Un module peut être modifié, supprimé ou ajouté sans "perturber" le système.
- L'ordonnanceur agit tant sur les KSARs du domaine que sur ceux du contrôle. Il ne dépend pas d'un plan a priori concernant ses choix mais reflète la stratégie de contrôle en cours dans le tableau de contôle.

Bien que BB-1 représente un aboutissement d'une conception déclarative du contrôle, il demeure des problèmes liés à la nature des connaissances et à leur utilisation. Le formalisme déclaratif des connaissances n'est pas une condition suffisante pour avoir la convergence et la cohérence entre les objets construits. Ainsi, il n'y a pas de critères de pertinence permettant de définir la nature de la connaissance car les objets ont une signification implicite.

A.5 Le système ATOME

ATOME [LMH87b][LMH88b][LMH88a][LMM+89] est un environnement de développement de systèmes tableau noir. La nouveauté introduite dans ce modèle est qu'il permet d'expliciter déclarativement les connaissances de contrôle ainsi que leur utilisation. En outre, il permet d'utiliser plusieurs tableaux noirs accessibles par des modules appelés *spécialistes* (voir Fig.A.3). En matière de contrôle, ATOME réalise un compromis entre CRYSALIS et BB-1. Il s'appuie sur une structure de contrôle hiérarchique, constituée d'une méta-méta-source de connaissance appelée *stratégie* qui active des méta-sources de connaissances, appelées *tâches*. Ces dernières regroupent des spécialistes qui coopèrent pour résoudre un sous-problème.

A.5.1 Objets et solutions

À l'instar des autres systèmes déjà décrits, le tableau noir du système ATOME est structuré en niveaux hiérarchiques d'abstraction. Pour chaque niveau est associé une classe d'objets caractérisés par leur attribut. Les hypothèses du niveau sont des instances de la classe. Les relations entre hypothèses sont exprimées explicitement par des attributs spéciaux appelés *liens*. Un lien d'une hypothèse comprend les hypothèses en relation avec cette dernière et comprend aussi un lien inverse. Les objets importants sont consignés dans une structure appelée *résumé de tableau*. L'importance des objets est décidée exclusivement par les modules qui les créent.

A.5.2 Les modules

ATOME comprend trois types de modules: les spécialistes, les tâches et la stratégie.

Les spécialistes: ce sont des sources de connaissances du domaine. Ils possèdent une structure de type condition/action. Le spécialiste est caractérisé par ses variables locales qui spécifient le contexte dans lequel la partie action doit être exécutée. Quand une tâche appelle un spécialiste, celui-ci devient activable. Le spécialiste est exécutable quand l'unification de sa pré-condition avec les focalisations d'attention qui lui sont transmises est satisfaite. La partie action d'un spécialiste peut être soit un programme, soit un système expert ou encore une base de règles écrite dans le formalisme d'ATOME.

Les tâches: une tâche est une méta-source de connaissances ayant pour but de résoudre un sous-problème particulier du problème posé. Elle contient de la connaissance pour décrire le modèle de coopération d'un groupe de spécialistes.
La partie détection des tâches est basée sur les *événements*. Ces derniers sont des structures de données créées lors des créations ou des modifications des hypothèses du tableau. Ils sont consignés dans une liste contenant tous les changements du tableau qui n'ont pas fait encore l'objet d'une utilisation pour la résolution du problème. Lors de sa création, un événement est dupliqué dans toutes les listes d'événements de toutes les tâches concernées.
La partie *contenu* des tâches est une base de règles dont les prémisses sont une conjonction entre les événements de la liste et des variables locales ou globales. Les conclusions contiennent la liste des spécialistes à activer de façon séquentielles ou opportuniste.

La stratégie: c'est une méta-méta-source de connaissances dont le rôle est d'analyser la qualité de la solution courante afin de déterminer la région du tableau sur laquelle il faut focaliser la résolution, ainsi que les sous-problèmes à résoudre et la manière de les résoudre. À l'instar de la tâche, la stratégie est une base de règles dont les prémisses testent les hypothèses jugées importantes pour la résolution. La partie action de la règle contient une ou plusieurs tâches à exécuter séquentiellement. Les hypothèses importantes que la stratégie consulte régulièrement sont consignées dans le résumé du tableau.

A.5.3 Connaissances de contrôle

Au niveau d'un spécialiste, une base de règles est interprétée suivant les procédures suivantes:

- Dirigée par les règles: l'interpréteur déclenche la règle la plus importante. En mode simple, le système s'arrête dès qu'une règle est déclenchée où si aucune ne l'est. En mode multiple, le système s'arrête après l'exécution de toutes les règles déclenchables en un seul passage. En revanche, en mode cyclique, le système boucle jusqu'à ce qu'aucune règle ne soit plus déclenchable.

- Dirigée par les événements: l'interpréteur choisit l'événement le plus important de la liste et recherche les règles déclenchables sur cet événement. Les modes simple et multiple sont similaires à ceux des spécialistes dirigés par les règles. En revanche, le mode cyclique-simple choisit l'événement le plus important, déclenche la première règle et recommence avec l'événement le plus important jusqu'à ce qu'il n'y aurait plus. Pour le mode cyclique-multiple, l'interpréteur choisit l'événement le plus important pour exécuter toutes les règles déclenchables, puis recommence avec le nouvel événement le plus important jusqu'à ce qu'il n'y aurait plus de règles déclenchables.

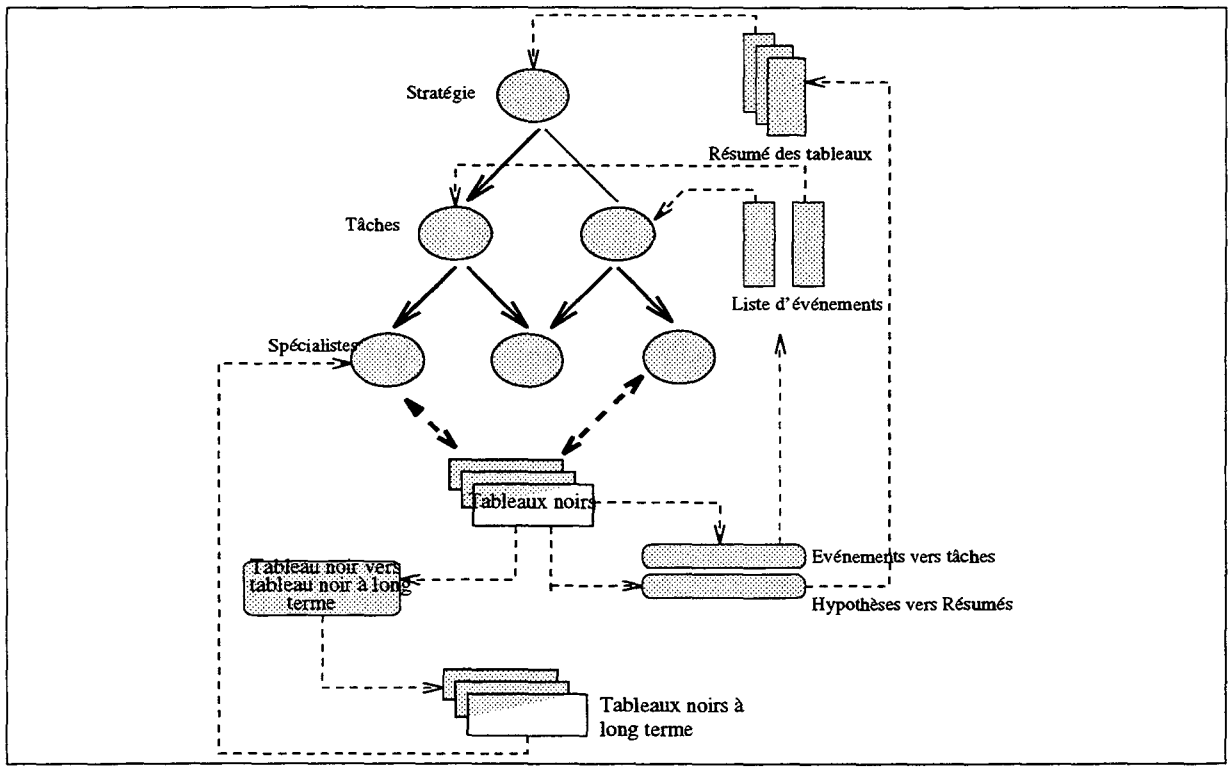

FIG. A.3 - *Architecture du système ATOME*

La base de règles d'une tâche est interprétée de trois façons différentes: dirigée par les règles, dirigée par les événements où de façon opportuniste.

- Dirigée par les règles: les spécialistes figurant dans les parties conclusion des règles sont déclenchés séquentiellement dans l'ordre où ils sont déclarés si leur pré-condition est vérifiée. Les événements ayant provoqué le déclenchement des règles font partie du foyer d'attention des spécialistes activés.

- Dirigée par les événements: contrairement au cas précédent, l'interpréteur choisit l'événement le plus important et recherche une règle à déclencher avec cet événement. Le mode de fonctionnement de la tâche peut être simple, multiple, cyclique simple ou cyclique multiple. Ainsi, les spécialistes activés sont déclenchés séquentiellement si leurs pré-conditions sont vérifiées. Comme dans le cas précédent, l'événement ayant provoqué le déclenchement est passé comme foyer d'attention aux spécialistes activés.

- Opportuniste: dans ce cas, l'interpréteur identifie pour chaque événement figurant dans la liste locale l'ensemble des règles déclenchables sur cet événement. Les spécialistes des règles détectées sont placées sous forme d'Instances de SPécialistes (ISP) dans un agenda appelé *agenda des ISPs sélectionnées.* Une ISP contient le spécialiste et l'événement qui l'a activé. L'interpréteur identifie parmi les ISPs, celles qui ont leur pré-condition vérifiée et les déplace dans l'agenda des ISPs exécutables. L'interpréteur détermine l'ISP la plus prioritaire en calculant les heuristiques pour toutes les ISPs exécutables.

Le cycle de base: l'exécution d'une règle de la base de règle de la stratégie déclenche un cycle de base du système. Pour un cycle donné, la boucle que suit le système est la suivante:

- L'interpréteur choisit la règle la plus importante ayant sa partie condition satisfaite et la déclenche. Les tâches activées disposent de foyers d'attention qui sont des résumés de tableaux noirs.
- L'activation de la tâche courante provoque l'activation et l'exécution des spécialistes. La tâche s'appuie sur les hypothèses fournies par la stratégie et les événements présents dans sa liste locale.
- L'activation des spécialistes engendrent des créations, des suppressions ou encore des modifications d'hypothèses dans le tableau.

Le système s'arrête lorsqu'il n'y aurait aucune règle de la stratégie à déclencher.

A.5.4 Conclusion

ATOME a concilié l'efficacité du contrôle hiérarchique avec la souplesse d'une gestion opportuniste basée sur des agendas et des heuristiques explicites. Il est considéré comme un environnement de développement de systèmes à base de tableau noir. Au delà de l'architecture de contrôle qu'il propose, il fournit un ensemble d'outils de gestion de la cohérence, de raisonnement temporel et approximatif,...
En effet, le fonctionnement d'ATOME est similaire à celui de CRISALIS si les connaissances de contrôle sont bien connues et les flux maîtrisés. Pour la mise au point des connaissances, le système fonctionne selon un mode opportuniste jusqu'à ce que des procédures figées de coopération entre modules apparaissent. Dans ce dernier cas, les procédures peuvent être codées directement dans les tâches. L'autre caractéristique d'ATOME est la déclarativité des connaissances de contrôle via des modules hiérarchiques. Cette dernière caractéristique permet de modifier facilement ces connaissances. En plus, les heuristiques sont explicites, par conséquent, elles ne suscitent pas de calculs compliqués. En résumé, si les connaissances sont connues et maîtrisées, il est possible de les représenter afin d'en faire une utilisation efficace. Dans le cas contraire, le comportement opportuniste permet de les utiliser de telle sorte à mettre en relief les coopérations fructueuses entre les modules.
Les modèles de contrôle décrits dans cette annexe ont été instanciés avec succès dans le milieu industriel. Il reste que l'absence d'une représentation explicite de l'environnement constitue un sérieux handicap pour ce type d'architectures. L'approche multi-agent repose particulièrement sur cet aspect qui consiste à caractériser un agent par un ensemble de propriétés lui permettant de percevoir son environnement, de raisonner sur ces croyances et d'établir un plan d'actions.

Annexe B

Les SMAs: interaction, coopération et communication

Préambule:

Cette annexe aborde les problèmes inhérents aux systèmes multi-agents. L'interaction est un aspect important dans un système multi-agent. Elle est matérialisée au travers des protocoles de communication et de coopération appropriés. Un ensemble de critères de classification, suivi d'une typologie des situations d'interaction sont fournis afin de permettre d'identifier les multiples cas d'interaction.

Cette annexe expose aussi les différentes formes de coopération et examine les méthodes mises en œuvre pour susciter un processus de coopération. Les avantages de la coopération, en terme de performances et d'utilisation rationnelle des ressources, sont aussi passés en revue.

Enfin, le dernier aspect abordé concerne l'aspect communication dans un univers multi-agent. Un ensemble de fondements basés sur la linguistique et la philosophie du langage, à l'image de actes de langage, sont examinés.

B.1 Interaction dans un contexte multi-agent

L'interaction est un aspect crucial dans la problématique des systèmes multi-agents. Elle consiste à mettre en relation dynamique deux ou plusieurs agents par l'intermédiaire d'une série d'actions réciproques. L'exécution de ces actions suscitent une influence sur le comportement futur des agents, ce qui donne naissance à l'interaction entre agents. L'interaction suppose un contact, entre agents, qui peut être direct ou indirect par le biais d'un autre agent ou de l'environnement. Il existe de multiples situations d'interaction entre agents comme, par exemple, l'aide d'un robot par un autre, l'échange de données entre serveurs informatiques,

l'association des compétences de plusieurs spécialistes pour établir un diagnostic, la répartition des charges sur plusieurs processeurs, etc...
L'interaction résulte de l'aspect pluriel des systèmes multi-agents, et apporte de ce fait, une dimension supplémentaire à l'agent. Ce dernier n'est plus considéré comme le "centre de l'univers" (comme c'est le cas en IA), mais comme un nœud d'un ensemble d'échanges et d'interdépendances. Ainsi, un agent sans interactions avec d'autres agents n'est plus qu'un système d'informations dépourvu de propriétés adaptatives. De ce fait, l'interaction est considérée comme un élément essentiel et fondateur des formes d'intelligence.
une situation d'interaction suppose la réunions de certains critères:

- La présence d'agents capables d'agir et de communiquer.
- Des situations susceptibles de servir de points de rencontre entre agents: collision de véhicules, utilisation de ressources partagées, régulation de la cohésion d'un groupe.

Dans [Fer95], J. Ferber a défini la notion de *situation d'interaction* comme suit:

"On appellera situation d'interaction un ensemble de comportements résultant du regroupement d'agents qui doivent agir pour satisfaire leurs objectifs en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent et de leur compétences individuelles".

La notion d'interaction permet de définir des catégories abstraites indépendantes de leurs réalisations concrètes, en identifiant d'une part les principaux invariants que l'on retrouve partout et, d'autre part, les rapports de différence qu'ils entretiennent.

B.1.1 Critères de classification

Afin d'appréhender les potentialités et les problèmes que présente la notion d'interaction, il convient de classifier les différents types d'interaction. Aussi, il est nécessaire de définir des paramètres d'interaction permettant de classer les situations. Les principales situations d'interaction sont classées par rapport à trois critères: les objectifs ou intentions des agents, les relations que les agents entretiennent vis à vis des ressources qu'ils détiennent, ainsi que les compétences dont ils disposent pour atteindre leurs buts.

Compatibilité des buts

La question qui se pose est de savoir si les agents ont des buts concordants qui vont dans le même sens, ou bien leurs objectifs sont contradictoires voire opposés.
D'une manière générale, on dira que les buts sont incompatibles si la satisfaction de l'un entraîne l'insatisfaction de l'autre, la réciproque étant évidemment vraie.

Définition: *le but d'un agent A est incompatible avec celui d'un agent B si les agents A et B ont comme buts respectifs d'atteindre les états décrits respectivement par p et q et que* $p \Rightarrow q$, *c'est à dire que :* $satisfait(but(A,p)) \Rightarrow \neg satisfait(but(B,q))$.

La distinction entre buts compatibles et buts incompatibles a été à l'origine d'une première classification. Les agents sont dans une situation de coopération si leurs buts respectifs

sont compatibles. Ils sont dans une situation de compétition si leurs buts respectifs sont contradictoires. Dans une situation de coopération, la viabilité d'un agent est renforcée par le comportement de l'autre agent.
L'exemple du modèle Proie-Prédateur est une bonne illustration des mécanismes de coopération et d'antagonismes [Fer91]. Les prédateurs forment un groupe et coopèrent pour chasser leur proie. Néanmoins, ils se situent dans un rapport antagonique avec les proies. La satisfaction du but des prédateurs entraîne la mort des proies. Inversement, la survie des proies entraînent l'échec des prédateurs.

Gestion des ressources

Un aspect important des interactions se rapporte aux ressources dont a besoin un agent pour accomplir sa tâche. Les quantités de ressources sont souvent limitées, et sont donc à l'origine des divers conflits. Ces derniers apparaissent lorsque plusieurs agents ont besoin des mêmes ressources en même temps et au même endroit, c'est à dire lorsque les zones correspondant aux ressources dont ils ont besoin se chevauchent. Par exemple, deux programmes s'exécutant sur le même processeur, deux robots qui s'approvisionnent à la même prise de courant sont des cas de conflits causés par manque de ressources.
La réponse aux situations conflictuelles est apportée par le biais de mécanismes de coordination d'actions et de résolution de conflits. Des techniques basées sur la granularité de l'agent en terme de capacités, consistent à donner la priorité à l'agent le plus "fort" au détriment du plus "faible". Il existe, aussi, des techniques basées sur la négociation qui consistent à établir un compromis entre les agents et de satisfaire, ainsi, modérément les besoins des différents agents. Les techniques de coordination anticipent sur les situations de conflit. Elles postulent que les conflits doivent être gérés avant même leur apparition. Elles comportent un ensemble de mécanismes, de règlements et d'actions supplémentaires qu'il est nécessaire d'accomplir pour que les actions soient possibles. Un exemple intéressant qui illustre parfaitement ces techniques de coordination sont les feux de signalisation et le code de la route qui sont des exemples de dispositifs et de règlements qui régulent le trafic routier et diminuent les risques de conflits entre automobilistes. En l'absence de systèmes de régulation de trafic, si une voiture tente de passer sans se soucier des autres voitures, des conflits apparaîtront. Ces conflits peuvent conduire jusqu'à un accident. Les conflits d'intérêts provoqués par l'accident peuvent être résolus par un constat amiable (négociation) ou devant un tribunal, auquel cas le conflit est résolu par l'arbitrage.

Capacités des agents

Le troisième critère concerne l'aspect relation entre les capacités des agents et les tâches que ces agents sont censés accomplir. Dans la conception d'un système multi-agent, il serait important de savoir si un agent peut réaliser seul une tâche ou bien il a besoin des autres pour atteindre son but. Pour une tâche simple, comme par exemple *aller chercher une bière dans le réfrigérateur*, un seul agent suffit pour l'accomplir. Cependant, la construction d'une navette spatiale impose la conjugaison et la coordination de plusieurs individus pour aboutir à l'accomplissement de la tâche commune. Finalement, le rapport capacité/tâche est essentiel car il détermine s'il y a interaction entre agents ou non en fonction de la complexité de la tâche à réaliser et des capacités des agents.

Buts	Ressources	Compétences	Types de situation	Catégorie
Compatibles	Suffisantes	Suffisantes	Indépendance	Indifférence
Compatibles	Suffisantes	Insuffisantes	Collaboration simple	Coopération
Compatibles	Insuffisantes	Suffisantes	Encombrement	Coopération
Compatibles	Insuffisantes	Insuffisantes	Collaboration coordonnée	Coopération
Incompatibles	Suffisantes	Suffisantes	Compétition individuelle pure	Antagonisme
Incompatibles	Suffisantes	Insuffisantes	Compétition collective pure	Antagonisme
Incompatibles	Insuffisantes	Suffisantes	Conflits individuels pour des ressources	Antagonisme
Incompatibles	Insuffisantes	Insuffisantes	Conflits collectifs pour des ressources	Antagonismes

TAB. B.1 - *Classification des situations d'interaction*

B.1.2 Typologie des situations d'interaction

Les critères définis dans les sections précédentes permettent d'établir une classification des situations d'interaction (voir Tableau B.1, repris de [Fer95]).

La situation d'*indépendance* correspond à la configuration où les agents ont des buts compatibles, des ressources suffisantes et des compétences suffisantes aussi. Dans ce cas, on dit que les agents sont *indifférents* i.e., un agent a tous ce qu'il faut pour accomplir la tâche qui lui est confiée, tout seul.
La situation de *collaboration simple* correspond à la configuration où les agents ont des buts compatibles, des ressources suffisantes, mais des compétences insuffisantes. Elle consiste à additionner les compétences des agents pour atteindre le but. Néanmoins, elle ne nécessite pas une coordination entre les agents. Elle caractérise des systèmes dans lesquels l'interaction correspond à l'allocation de tâches et au partage d'informations. Les systèmes qui mettent en œuvre un ensemble de spécialistes sont un exemple de collaboration simple car ils ne font que partager des connaissances pour aboutir à la résolution d'un problème. L'*encombrement* est caractérisé par la configuration suivante: les buts poursuivis par les agent sont compatibles, les ressources requises ne sont pas suffisantes, alors que les compétences sont suffisantes. Un exemple de ce type de situation est le placement des tâches sur un processeur, ou encore la régulation du trafic aérien. La résolution de ce problème nécessite des techniques de coordination d'actions qui sont détaillées dans les sections ultérieures (section B.3).
La situation de *collaboration coordonnée* a lieu quand les agents ont des buts compatibles, des ressources insuffisantes et des compétences insuffisantes. C'est une situation de coopération très complexe car elle suppose que les agents doivent coordonner leurs actions pour pouvoir

disposer de l'apport mutuel de leur compétences. Comme exemples illustratifs, nous pouvons citer les systèmes de contrôle de réseaux et la conception et la fabrication de produits industriels. La situation *compétition individuelle pure* est représentée par la configuration suivante: les agents ont des buts incompatibles, des ressources suffisantes et des compétences suffisantes. La compétition est pure car les ressources suffisantes et l'accès aux ressources ne provoque pas de conflits. Dans ce cas, les agents doivent négocier pour atteindre leurs buts. Cependant, il n'y a pas de problèmes spécifiques d'interaction liés à ce type de situation. La situation *compétition collective* est identique à la situation précédente sauf que les compétences des agents sont insuffisantes. Dans ce cas, les agents se regroupent en coalitions pour atteindre leurs buts. Ces coalitions sont formées à travers deux mouvements: le premier consiste à regrouper les agents en groupes ayant des rapports de collaboration coordonnés, le second tend à opposer les groupes entre eux.
La situation *conflit individuel pour des ressources* est exprimée par la configuration où les agents ont des buts incompatibles, des ressources insuffisantes et des compétences suffisantes. Cette situation illustre le cas où les ressources ne peuvent pas être partagées et suscitent ainsi des conflits car chaque agent veut disposer de ces ressources à lui tout seul. Enfin la situation de *conflit collectif pour des ressources* où les agents ont des buts incompatibles, des ressources insuffisantes et des compétences insuffisantes. Elle est caractérisée par le fait qu'elle combine la compétition collective aux conflits individuels pour les ressources.

B.2 Coopération dans un SMA

La notion de coopération peut être appréhendée sous plusieurs points de vue. Elle peut être vue comme une attitude des agents qui mènent un travail en commun ou comme une interprétation des comportements des agents en les qualifiant de coopératifs ou non selon certains critères comme l'interdépendance des actions où encore la fréquence des communications effectuées.

B.2.1 La coopération comme attitude intentionnelle

Cette forme de coopération postule que des agents coopèrent s'ils s'engagent dans une action commune après avoir identifié et adopté un but commun. La formation d'un groupe est le fruit de la prise de conscience de l'existence d'un but commun et de l'engagement de chacun des agents du groupe à atteindre ce but.
J. Galliers [Gal90], C. Castelfranchi et R. Conte [CMC91] affirment que le problème de l'engagement envers un but collectif est appelé *adoption de but* qui est considéré comme un facteur essentiel dans l'activité sociale des agents. Particulièrement, J. Galliers suppose qu'il y a coopération si les agents s'engagent dans une action et identifient un but commun, i.e., un agent reconnaît que les agents sont engagés dans le même but. Néanmoins, la question qui se pose est de savoir si la coopération est le fruit d'une intention de coopérer, et si toute intention de coopérer engendre nécessairement la coopération?
En outre, la coopération ne présente pas que des avantages, mais aussi des inconvénients. Deux agents peuvent être dans une situation de coopération car ils sont conscients que leurs actions sont engagées dans un but commun, ou une intention commune. Pourtant, les performances des agents pris individuellement peuvent être plus bonnes que les performances résultant de la coopération. Le second inconvénient est que cette forme de coopération n'offre aucune possibilité de coopération aux agents réactifs qui n'ont pas d'intention explicite, ni

de modèle des autres agents. Cependant, des agents réactifs, à l'image des fourmis qui travaillent ensemble pour récupérer de la nourriture de l'extérieur vers leur nid, coopèrent même s'ils n'ont pas conscience [Fre94]. Finalement, l'attitude intentionnelle des agents n'est pas la seule forme de coopération, mais il existe une autre forme basée sur l'interaction entre les agents. La première forme de coopération caractérise les agents cognitifs, tandis que la seconde caractérise les agents réactifs.

B.2.2 La coopération comme résultat d'une interaction

Des chercheurs, comme Durfee [DLC89], considèrent la coopération comme une qualification de l'activité d'un ensemble d'agents, vue de l'extérieur par un observateur qui fait abstraction des états mentaux de l'agent. Le comportement des fourmis est considéré comme coopératif, car pour un observateur externe, il y a un certain nombre de phénomènes qui sont des indices d'une activité de coopération. Cette forme de coopération ne s'intéresse pas aux caractéristiques internes des agents, elle s'occupe seulement du comportement observable de l'agent.
Dans [Fer95], J. Ferber caractérise l'activité de coopération par deux indices qui semblent nécessaires et suffisants pour savoir s'il y a ou non une activité de coopération. Ces indices sont: l'efficacité du travail en groupe, et l'existence de mécanismes de résolution de conflits d'accès à des ressources.

Définition : *On dira que plusieurs agents coopèrent, ou encore qu'il sont dans une situation de coopération, si l'une des deux conditions est vérifiée:*

1. *L'ajout d'un nouvel agent permet d'accroître différentiellement les performances du groupe.*

2. *L'action des agents sert à éviter ou à résoudre des conflits potentiels.*

Le premier indice signifie que si les agents coopèrent, l'ajout d'un nouvel agent est considéré comme une aide pour l'ensemble du groupe. Dans ce cas, on dit que les agents sont en situation de collaboration.
Néanmoins, si l'introduction d'un agent n'améliore pas les performances du groupe, cet agent suscite une contrainte pour les activités du groupe. Dans ce cas, les agents interviennent pour limiter les dégâts. Ils coopèrent pour faire en sorte que la diminution des performances ne soient pas substantielle.

B.3 Les méthodes de coopération

Après avoir examiné les conditions de coopération, il convient d'analyser les méthodes mises en œuvre pour susciter un processus de coopération. Ils existent six méthodes de coopération: le regroupement et la multiplication, la communication, la spécialisation, la collaboration par partage des tâches et des ressources, la coordination d'actions, la résolution de conflits par arbitrage et négociation. La première méthode consiste à rapprocher physiquement les agents pour former un bloc homogène dans l'espace. Le bloc constitué est similaire à un réseau de communication permettant à plusieurs agents de se comporter comme s'ils étaient physiquement les uns à côté des autres. D'autre part, l'augmentation quantitative des agents dans un système présente des avantages considérables, aussi bien du point de vue de l'augmentation

des performances que de celui de la fiabilité. Le cas des robots explorateurs est édifiant, car l'existence d'un grand nombre de robots permet d'accroître la fiabilité du système. La seconde méthode correspond à la *communication.* Cette dernière étend les capacités perceptives des agents en leur permettant de bénéficier des informations et du savoir-faire des autres agents. Dans les système d'agents cognitifs, la communication se fait par le biais de l'envoi de message, alors que dans les systèmes d'agents réactifs, la communication résulte de la diffusion d'un signal dans l'environnement. La communication constitue le support privilégié pour assurer la répartition des tâches et la coordination d'actions. Il y a aussi la *spécialisation* qui est un processus à travers lequel des agents deviennent plus adaptés à accomplir leurs tâches. Une forme d'adaptation individuelle permet d'associer des rôles temporaires à des agents afin de leur permettre de se spécialiser dans l'accomplissement de leurs tâches. Cette spécialisation aura des répercussions au niveau des performances de la collectivité. A. Drougoul [Dro93] a montré que, si l'on utilise des agents capables de renforcer leur tendance à s'occuper préférentiellement d'une couleur, alors une société spécialisée est plus efficace qu'une société d'agents classiques non spécialisés, à cause de la réduction des errements aléatoires résultant d'une meilleure sensibilité à un type particulier d'objets.
La *collaboration par partage de tâches et de ressources* est une méthode qui consiste à faire travailler plusieurs agents sur une tâche commune. La notion de collaboration sous-entend un ensemble de techniques permettant à des agents de (se) répartir, des tâches, des connaissances et des ressources afin de participer à la réalisation d'une œuvre commune. Dans des systèmes d'agents cognitifs, l'allocation des tâches est basée sur des mécanismes d'offre et de demande. Il existe deux types de mécanismes de répartition de tâches. D'un coté, il y a les mécanismes centralisés, dans lesquels un agent coordinateur centralise les offres et les demandes et les répartit ensuite au mieux, d'un autre coté, il y a les mécanismes distribués qui postulent qu'un agent peut être à la fois offrant ou demandeur. Il existe deux techniques: celles qui s'appuient sur des modèles d'accointance [GBh87], et celles qui sont basées sur la notion d'appel d'offre dont la plus connue est le "réseau contractuel" (contract net) [DS83].
La *coordination d'actions* est une méthode de coopération qui suppose l'exécution de certaines tâches supplémentaires qui n'interviennent pas directement dans la résolution du problème. L'exécution de ces tâches a pour but l'accomplissement des actions de résolution dans les meilleures conditions. Ces tâches sont appelées *tâches de coordination.* L'ultime méthode restante est *la résolution de conflits par arbitrage et négociation.* La résolution des conflits dans un contexte multi-agent utilise deux techniques qui sont l'arbitrage et la négociation. L'arbitrage suppose la définition de règles de comportement qui agissent comme des contraintes sur l'ensemble des agents susceptibles de se retrouver en situation de conflit. Le résultat est global car il permet de réduire les conflits et de préserver les groupes d'agents.
Lorsque des agents cognitifs entrent en conflit d'objectif ou de ressource, la technique d'arbitrage n'est pas utilisée. Les agents s'engagent dans un processus de négociation afin de résoudre eux-mêmes le conflit par la recherche d'un accord. Des chercheurs américains comme V. Lesser [CML88], E. Durfee [DLC89] et K. Sykara [Syc89] se sont penchés sur les problèmes de négociation dans un contexte multi-agent.

B.3.1 Avantages

La coopération met en avant les performances collectives des agents et offre, ainsi, certains avantages sous forme d'améliorations individuelles ou collectives comme la réalisation d'une

tâche impossible à réaliser par un seul agent. On peut citer les avantages suivants:

1. Accomplissement des tâches impossibles à réaliser par un seul agent. C'est une situation où la collaboration permet d'accomplir l'action désirée.

2. Améliorer la productivité de chacun des agents. Il existe un accroissement considérable de la productivité en fonction du nombre d'agents.

3. Augmentation du nombre de tâches réalisées dans un délai imparti, ou encore, diminution du temps de réalisation d'une tâche.

4. Amélioration de l'utilisation des ressources. Ainsi, une meilleure gestion de l'utilisation des ressources entraîne une amélioration des performances du système.

B.4 Communication

Il est clair que sans communication, les agents ne peuvent guère coopérer, ni coordonner leurs actions et encore moins réaliser des tâches en commun. Il existe un grand nombre d'approches de la communication. Dans ce contexte, les experts en linguistique et en philosophie du langage ont développé une panoplie de concepts qui ont servi de fondements pour les principales approches de la communication dans les systèmes multi-agents.

B.4.1 L'intention de communiquer

La question qui est souvent posée est de savoir si l'action de communication est liée à une intention de l'agent émetteur désirant obtenir quelque chose de l'agent destinataire, ou s'agit-il d'un processus indépendant de la volonté de l'agent émetteur?
Dans le premier cas, on parle de communication *intentionnelle*, tandis que dans le second cas, on parle de communication incidente. Ces deux formes de communication caractérisent les systèmes multi-agents. Les communications intentionnelles expriment une volonté d'action. En revanche, les communications incidentes sont effectuées sans que l'émetteur y prenne une part active. Elles présentent deux caractéristiques intéressantes: (i) la sémantique de la communication est liée uniquement à l'état de l'émetteur; (ii) l'interprétation est variable et dépend de la signification que le destinataire va attribuer à cette communication. Donc, les communications incidentes sont totalement liées aux inférences interprétationnelles de l'agent destinataire. Bien qu'il existe une sémantique sous forme de signes référant à un état de l'émetteur, il n'y a pas de pragmatique intrinsèque, i.e., le signe peut engendrer des comportements différents chez les agents receveurs.
L'intention de communiquer est un système gradué qui dépend des capacités cognitives de l'émetteur. D. Dennett [Den83] a proposé un ensemble de niveaux d'intentionnalité hiérarchisés selon une échelle de complexité qui part de 0 jusqu'à un niveau théorique n. Par exemple, le niveau 0 décrit des situations où l'émetteur envoie un signal parce qu'il présente un certain état interne (par exemple, crier parce qu'il a faim), même s'il n'existe aucun récepteur pour ce signal. Il peut s'exprimer de la manière suivante:

X envoie le message **M** parce que **X** perçoit **S**.

L'intentionnalité d'ordre 1 suppose la volonté de l'émetteur de susciter un effet au sein du récepteur. Par exemple, crier pour faire quitter les lieux à un animal. La relation entre le message et l'intention peut être exprimée de la manière suivante:

X envoie le message **M** à **Y** parce que **X** veut que **Y** fasse **P**.

L'intentionnalité de niveau 2 suppose que l'agent émetteur ne veut plus d'une réponse directe de l'agent destinataire, mais une croyance sur un état du monde. Sa forme logique est la suivante:

X envoie le message **M** à **Y** parce que **X** veut que **Y** croie **P**.

B.4.2 Les actes de langage

La publication en 1962 par Austin du livre *How To Do Things With Words* (Quand dire, c'est faire) a suscité un intérêt crucial au sein de la communauté des philosophes linguistes. Pour les disciples de Ferdinand de Saussure, étudier le langage revient à appréhender le sens des phrases en indiquant comment à partir d'une combinaison de mots il est possible de donner un énoncé ayant un sens. Les théoriciens de la sémantique du langage s'intéressent à la manière dont une phrase rend compte de la réalité, de façon satisfaisante. Austin et ses successeurs, à l'image de John Searle, affirment que l'énonciation est un acte qui a pour rôle, avant tout, de produire un effet sur son destinataire. Par exemple, lorsqu'on dit "ramène moi une bière du réfrigérateur", on ne s'intéresse pas à la valeur de vérité de la phrase, mais à l'effet suscité chez l'interlocuteur. La phrase énoncée n'est ni vraie ni fausse, elle renvoie à une action qui peut réussir ou échouer. Ainsi, l'aspect vérité de la phrase perd son intérêt au profit du critère de succès de la communication, i.e., la pragmatique des communications.

Définition: un acte de langage désigne l'ensemble des actions intentionnelles effectuées au cour d'un processus de communication. Searle [Sea69] et Vanderveken [Van88] ont distingué les actes de langage suivants:

1. Les *assertifs* ont pour rôle d'informer sur le monde en donnant une information sur quelque chose (exemple: il fait beau, etc...).
2. Les *directifs* consistent à donner des directives à l'agent destinataire (exemple: prepare toi à manger,...).
3. Les *promissifs* engagent le locuteur à accomplir certaines tâches dans l'avenir (exemple: je donnerai un cour demain matin à 8 heures, etc...).
4. Les *expressifs* consistent à donner à l'agent destinataire des indications concernant l'état mental du locuteur (exemple: je suis de bonne humeur, etc...).
5. Les *déclaratifs* consistent à accomplir un acte en l'énonçant (exemple: je déclare la séance ouverte,...).

Malgré son intérêt pratique, la classification précédente n'est pas considérée comme étant exhaustive. Elle pose encore de nombreux problèmes qui font l'objet de recherches dans le domaine de la pragmatique linguistique [Réc81] [Moe89].

B.4.3 Actes locutoires, illocutoires et perlocutoires

Selon Austin [Aus62] et Searle [Sea69], les actes de langage sont définis comme des structures complexes formées de trois composantes élémentaires:

1. La composante *locutoire* correspond à la production matérielles des énoncés par écriture de caractères, ou par émission d'ondes sonores, etc...

2. La composante *illocutoire* se rapporte à la réalisation de l'acte effectué par le locuteur sur le destinataire de l'énoncé. Ces actes peuvent être représentés sous la forme **F(P)** où **F** est la *force illocutoire* et **P** est le contenu propositionnel. Par exemple, affirmer qu'il fait beau est représenté comme suit:

   Affirmer(il fait beau)

   L'interrogation correspondante à l'affirmation précédente est représentée par:

   Questionner(il fait beau)

   Les actes illocutoires sont souvent introduits par un verbe appelé *performatif* qui désigne le type d'acte.

3. La composante *perlocutoire* est définie comme étant une conséquence des actes illocutoires. Elle porte sur les effets que les actes illocutoires peuvent avoir sur l'état du destinataire, ses actions, ses croyances. L'action de convaincre, d'inspirer, d'effrayer est un acte perlocutoire.

B.4.4 Le langage KQML

Dans la perspective du développement d'un standard de communication de haut niveau, basé sur des actes de langage, la DARPA[1] lança un projet de recherche qui donna naissance au langage KQML[2]. Bien qu'il présente un grand intérêt aux yeux des utilisateurs, le langage KQML présente plusieurs lacunes. Cohen et Levesque ont relevé les inconvénients suivants:

1. *Ambiguïté et imprécision:* la description des performatifs est exprimée sous forme de simples descriptions en langage naturel, ce qui les rend vagues et confus. Les performatifs **achieve, broker** et **stream-all** ne sont pas considérés comme des actes de langage indirects car ils satisfont des buts appartenant aux autres agents.

2. *Performatifs inutiles et incohérents:* certains performatifs ne sont pas véritablement des actes de langage, car ils n'engendrent pas la satisfaction des buts de l'agent émetteur.

3. *Performatifs manquants:* il existe une catégorie de performatifs qui font défaut à KQML, ce sont les *promissifs*. Par exemple, il n'est pas possible en KQML de dire que l'on s'engage, auprès d'un tiers, à accomplir une action.

1. Defense Advanced Research Projects Agency
2. Knowledge Query Manipulation Language

A travers ces inconvénients, il faut remarquer que KQML manque de spécification et de formalisation. En conséquence, un utilisateur quelconque pourra prétendre qu'il utilise KQML parce que ses agents sont capables de s'envoyer des requêtes et des informations. Après avoir fait le constat de ces critiques, Cohen et Levesque proposent de définir un ensemble minimum de performatifs en leur donnant des propriétés de compositionnalité, afin de pouvoir définir de nouveaux performatifs à partir d'actes de langage initialement définis.

B.5 Le langage APRIL

Les langages de programmation adaptés à l'implémentation de systèmes multi-agents doivent couvrir une collection variée de domaines. Ils doivent avoir un puisssant support pour la résolution de problèmes et la représentation de la connaissance, mais aussi fournir un support d'exécution sur un réseau d'ordinateurs et répondre à des événements en temps réel. April[3] est un langage symbolique orienté processus. Il fournit un ensemble de facilités ayant trait à la définition et à la communication, d'une manière uniforme, de processus dans un environnement distribué. Il dispose aussi de puissants outils lui permettant d'avoir un haut degré d'expressivité et de manipulation des structures de données comme c'est le cas des langages de programmation symboliques de haut niveau. Les structures symboliques d'April sont basées sur des tuples, utilisables comme des listes, des enregistrements ou encore des ensembles. En plus de ses aspects "computationnels", April incorpore aussi certaines propriétés syntaxiques comme son fort typage. Il fournit des propriétés de haut niveau et il a aussi une syntaxe de précédence d'opérateurs liée avec un sous langage de macro-traitements.
Les propriétés d'April sont utilisées pour la structuration de programmes (les modules sont des objets de haut niveau) et autorise les fonctions et les procédures d'être passées d'un processus à un autre.
Le macro-traitement permet une facilité importante au niveau de l'utilisation du langage. Ainsi, avec ce macro-traitement, il est possible de construire des couches au dessus du langage de base et d'incorporer des propriétés supplémentaires. Parmi les améliorations qui ont été faites à base de cette facilité, on peut citer une extension d'April afin de supporter les aspects objets [CM94]. Particulièrement, un package de macro et de procédures forme le langage MAIL [SBKL92]. Le langage April est conçu pour servir comme langage d'implémentation de MAIL.

L'une des caractéristiques d'April réside au niveau du système de nommage des processus. Ces derniers ont des noms appelés *handles* qui leur sont associés. Ainsi, un programmeur doit associer un nom à un processus au moment de sa création. Ces noms peuvent être passés comme arguments de fonctions ou de procédures, ou encore dans des messages envoyés aux processus. Les noms assignés par le programmeur aux processus sont locaux au système April. Néanmoins, il est possible de rendre les noms publiques. Pour ce faire, il suffit d'enregistrer le nom du processus avec le serveur des noms auquel est lié l'invocation du système April. Le serveur de noms d'April a exactement le même rôle qu'un serveur de noms du domaine Internet. Par conséquent, ce modèle de nommage des processus permet de construire des applications globales sous April.

La communication sous April repose sur un simple mécanisme d'envoi de messages entre

3. Agent PRocess Interaction Language

processus. Les messages pouvant avoir une structure symbolique complexe, sont échangés entre des processus indépendemment de leur location. Dans April, chaque processus a un buffer pour recevoir les messages qui lui sont envoyés. La primitive d'envoi de messages permet d'envoyer un message à un seul ou à plusieurs processus. Les processus utilisent un mécanisme leur permettant de déterminer les messages qu'ils souhaitent recevoir. Typiquement, ce filtrage des messages s'appuie sur un dispositif permettant de filtrer les messages à recevoir. L'ordre d'arrivé de deux messages M1 et M2 dans le buffer des messages du processus n'est pas nécessairement équivalent à l'ordre de transmission. April n'est pas en mesure de préserver l'ordre des messages envoyés par des processus différents car il n'y a pas d'horloge de synchronisation globale. Cependant, l'ordre des messages échangés entre une paire donnée de processus est préservé. Cette propriété est importante pour les applications client/serveur.
April est destiné à être utilisé dans un environnement hétérogène où des applications sont typiquement développées dans des langages distincts. En effet, des fonctions et procédures externes peuvent établir des liens avec une application April. En outre, un programme April peut communiquer avec d'autres applications en utilisant un protocole TCP/IP.
Les aspects temps réel sont aussi pris en charge par le langage. April permet à un programme d'être synchronisé avec les événements temps réel. Dans ce contexte, des processus April peuvent être programmés afin d'être activés par exemple le 1 Janvier 2000 à minuit. Néanmoins, comme April est un langage symbolique, il n'est pas toujour possible de prédire précisément la durée que va prendre l'exécution d'une opération. Ce qui rend April non adapté pour des applications temps réel où les temps de réponse doivent être garanties avec une certaine tolérance. En plus de la limite précédente, April ne supporte pas les aspects multi-threadés qui sont d'un grand apport sur le plan de l'efficacité car les processsus qu'il manipule sont des processus Unix.

B.6 Conclusion

Dans cette annexe l'importance des aspects interaction, coopération et communication dans un univers multi-agent est examinée. Ces aspects sont mis en évidence durant le processus de résolution de problèmes pour que les agents puissent coordonner leurs actions afin de résoudre des conflits ou d'accéder à certaines ressources, ou encore de coopérer en vue d'accomplir la tâche qui leur est confiée. La coopération est à l'origine de l'accroissement des performances collectives des agents et offre, ainsi, certains avantages sous forme d'améliorations individuelles ou collectives comme la réalisation d'une tâche impossible à réaliser par un seul agent. Les modèles de communication dans un système multi-agent reposent souvent sur les actes de langage qui tirent leur fondements de la philosophie du langage et qui sont connus pour leur intérêt pratique. La pertinence des travaux sur les actes de langage qui s'est matérialisée par leur forte influence sur les modèles de communication dans les systèmes multi-agents a fait qu'il étaient à la base du protocole de communication de l'architecture MAGIQUE. Cependant, dans ce dernier, les communications entre les agents spécialistes sont symétriques, par contre celles entre un spécialiste et un superviseur ne le sont pas. Cette asymétrie des communications est due à l'aspect hiérarchique de l'architecture.

Table des figures

Liste des tableaux

Bibliographie

[AC87] P.E. Agre and D. Chapman. Pengi: an Implementation of a Theory of Activity. In *Proceedings of AAAI'87*, pages 268–272. Morgan Kaufmann, 1987.

[Agh86] Gul A. Agha. ACTORS: A Model of Concurrent Computation in Distributed Systems. Technical report, MIT Press, Cambridge, Massachusetts, 1986. Series in Artificial Intelligence.

[Aus62] J.L. Austin. *How to Do Things With Words.* Clarendon Press, 1962. Trad. fr. Quand dire c'est faire, Le seuil, 1970.

[Bac90] Bruno Bachimont. *Cohèrence et convergence dans un tableau noir: Organisation, formalisation et semantique de l'architecture de contrôle ABACAB.* PhD thesis, Université de Paris VI, Paris-France, 1990.

[Bal97] Tucker Balch. Learning Roles: Behavioral Diversity in Robots Teams. In *AAAI Workshop on Multiagent Learning*, 1997.

[BC86] J-P. Briot and P. Cointe. The OBJVLISP Model: Definition of a Uniform, Reflexive and Extensible Object Oriented Language. *Rapport n 1. CMI*, 1986.

[BCM93] F. Bousquet, C. Cambier, and C. Mullon. Simulating the Interaction between a Society and a Renewable Resource. *Journal of Biolog. Systems*, 1(2):199–214, 1993.

[BE92] H. Bachatène and P. Estraillier. A Modular Design on Stepwise Refinements of Coloured Petri Nets. Technical report, MASI, Université Paris 6, 1992.

[BEL95] Claire Beyssade, Patrice Enjalbert, and Claire Lefèvre. Cooperating Logical Agents. In *Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages*, pages 299–314, Montréal-CANADA, 1995.

[BF81] A. Barr and E.A. Feingenbaum. *Handbook of Artificial Intelligence*, volume 1. Morgan Inc., 1981.

[BG88] A.H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, 1988.

[BHD94] R. Beckers, O.E. Holland, and J.L. Deneubourg. From Local Actions to Global Tasks: Stigmergy and Collective Robotics. In Proceedings of the Fourth International Workshop on the Synthesis and Artificial Life IV Simulation of Living Systems, editors, *R. Brooks and P. Maes*, 1994. MIT Press.

[BIP87a] M.E. Bratman, D.J. Israel, and M.E. Pollack. Toward an Architecture for Resource-Bounded Agents. Technical report, Center for the Study of Language and Information, SRI and Stanford University, 1987. CSLI-87-104.

[BIP87b] M.E. Bratman, D.J. Israel, and M.E. Pollack. Toward an Architecturefor Resource-Bounded Agents. Technical report, Centre for the Study of Language and Information, SRI and Stanford University, 1987. CSLI 87-104.

[BIP88] M.E. Bratman, D.J. Israel, and M.E. Pollack. Plans and Resource-Bounded Practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

[BLC90] T. Berners-Lee and R. Cailliau. World Wide Web for a HyperText Project. Technical report, CERN European Laboratory for Particle Physics, Genève, Suisse, 1990.

[BM95] Nourredine Bensaid and Philippe Mathieu. Un modèle d'architecture multi-agents entièrement écrit en Prolog. In *IV Journées Francophones de Programmation Logique, JFPL'95*, pages 381–385, Dijon-France, 1995. teknea, Toulouse-France.

[BM97a] Nourredine Bensaid and Philippe Mathieu. A Framework for Cooperation in Hierarchical Multi-Agent Systems. *Mathematical Modelling and Scientific Computing*, 8, 1997. ISSN 1067-0688.

[BM97b] Nourredine Bensaid and Philippe Mathieu. A Hybrid and Hierarchical Multi-Agent Architecture Model. In *Proceedings of the Second International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology, 21st-23rd April 1997, PAAM'97*, pages 145–155, London-United-Kingdom, 1997. The Practical Application Company Ltd.

[BM97c] Nourredine Bensaid and Philippe Mathieu. A Hybrid Architecture for Hierarchical Agents. In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications, 10-12 February 1997, ICCIMA'97*, pages 91–95, Gold-Coast, Australia, 1997. GRIFFITH UNIVERSITY.

[BM98] Nourredine Bensaid and Philippe Mathieu. An Autonomous Agent System to Simulate a Set of Robots Exploring a Labyrinth. In *Proceedings of the 11th International FLorida Artificial Intelligence Research Society Conference, FLAIRS'98*, pages 384–388, Sanibel Island, FLORIDA-USA, May 1998. AAAI Press.

[Bra92] C. Brassac. Analyse de conversation et théorie des actes de langages. *Cahier de linguistique française*, 1992. 13.

[Bro86] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, RA-2 (1), 1986.

[Bro90] Rodney A. Brooks. Elephants Don't Play Chess. *Robotic and Autonomous System*, 1990. 6.

[Bro91] Rodney A. Brooks. Intelligence without Representation. *Artificial Intelligence*, pages 139–159, 1991. 47.

[BS84] B.G. Buchanan and E.H. Shortliffe. *Rule-based Experts Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, Reading, Massachussets, 1984.

[BS92] B. Burmeister and K. Sundermeyer. Cooperative Problem-Solving guided by Intentions and perception. In *Y. Demazeau and E. Werner, Decentralized Artificial Intelligence 3*, volume 3, North Holland, 1992.

[Cam94] C. Cambier. *SIMDELTA : un système multi-agent pour simuler la pêche sur le Delta Central du Niger.* PhD thesis, Université de Paris 6, Paris-France, 1994.

[Car92] C. Carle. *Un language d'acteur pour l'intelligence artificielle distribuée intégrant objets et agents par réflexivité compilatoire.* PhD thesis, Université de Paris 6, Paris-France, 1992.

[CC93] Y. Cosmadopoulos and D. Chu. *IC Prolog, version 0.95, for Sun Workstations.* Department of Computing, Imperial College, London, 1993.

[CC94] F.G. M Cabe and K.L. Clark. APRIL: Agent PRocess Interaction Language. In *Proceedings of the 1994 Workshop on Intelligent Agents; Agent Theories, Architectures, and Languages*, pages 324–340, Amsterdam-The Netherlands, 1994.

[CD90] Brahim Chaib-Draa. *Contribution à la Résolution distribuée de problèmes: Une approche basée sur les états intentionnels.* PhD thesis, Université de Valenciennes et du Hainaut Cambrésis, Valenciennes-France, 1990.

[CGM86] D.C. Corkill, K.Q. Gallagher, and K.E. Murray. GBB: A Generic Blackboard Development System. In *Proceedings of the AAAI'86 Conference*, volume Vol. 2, pages 1008–1014, 1986.

[Chu92] D. Chu. IC-Prolog II : a Language for Implementing Multi-Agent Systems Language Features. In *Proceedings of the Special Interest Group on Cooperating Knowledge Based Systems*, 1992. Keele.

[Chu93] Dammian Chu. I.C. PROLOG II: A Language for Implementing Multi-Agent Systems. In *Proceedings of the 1992 Workshop on Cooperating Knowledge Based Systems (CKBS-92)*, pages 61–74, DAKE Center, University of Keele, UK, 1993.

[CL90] Philip R. Cohen and Hector J. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, 42:213–261, 1990.

[CM94] K.L. Clark and F.G. McCabe. Distributed and Object Oriented Symbolic Programming in APRIL. Technical report, Dept. of Computing, Imperial College, London-UK, 1994.

[CMC91] R. Conte, M. Miceli, and C. Castelfranchi. Limits and Levels of Cooperation : Disentangling Various Types of Prosocial Interaction. In Distributed Artificial Intelligence 2, editor, *Y. Demazeau et P. Müller*, North-Holland, 1991.

[CMG82] K.L. Clark, F.G. McCabe, and S. Gregory. IC-Prolog Language Features. In K.L. Clark and S.-A. Tarnlund, editors, *Logic programming*, pages 253–266, 1982. Academic Press, London.

[CML88] S. Conry, R.A. Meyer, and V. Lesser. Multistage Negociation in Distributed Artificial Intelligence. In *Readings in Distributed Artificial Intelligence, A. Bond et L. Gasser (Ed)*, 1988. Morgan Kaufman.

[CP79] Philip R. Cohen and C-R. Perrault. Elements of a Plan Based Theory of Speech Acts. *Cognitive science*, 3:177–212, 1979.

[DAG+86] J.L. Deneubourg, S. Aron, S. Goss, J.M. Pasteels, and G. Duerinck. Random Behaviour, Amplification Processes and Number of Participants : How they Contribute to the Foraging Properties of Ants. *Physica*, 22(D):176–186, 1986.

[DB90] M. Drummond and J. Bresina. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. In *Proceedings of the Eight National Conference on Artificial Intelligence (AAAI'90)*, pages 138–144. AAAI Press/MIT Press, 1990.

[Del68] P. Delattre. Structure et Fonction (Biologie). In *Encyclopaedia Universalis*, pages 442–444, 1968. vol. 15, (Ed.).

[Den83] D.C. Dennett. Intentional Systems in Cognitive Ethology : the Panglossian Paradigm Defended. *The Behavioral and Brain Sciences*, 6:343–390, 1983.

[Den87] D. Dennett. The intentional Stance. Technical report, MIT Press, Camridge, MA, 1987.

[DFCF92] A. Drougoul, J. Ferber, B. Corbara, and D. Fresneau. A Behavioral Simulation Model for the Study of Emergent Social Structures . In Toward a Practice of Autonomous Systems, editor, *P. Bourgine et F. Varela*, pages 161–170, Paris, 1992. MIT Press.

[DLC87] E.H. Durfee, V.R. Lesser, and D.D. Corkill. Cooperation through Communication in a Distributed Problem Solving Network. In *Distributed Artificial Intelligence*, 1987. M. Huhns (Ed.), Pitman.

[DLC89] E.H. Durfee, V.R. Lesser, and D.D. Corkill. Cooperative Distributed Problem Solving. In *The Handbook of Artificial Intelligence, A. Barr, P.R. Cohen et E.A. Feigenbaum (Ed)*, volume IV, pages 83–148, Adison-Wesley, 1989.

[Dro93] A. Drougoul. *De la simulation multi-agent à la résolution collective de problèmes. Une étude de l'émergence de structures d'organisation dans les systèmes multi-agents.* PhD thesis, Université de Paris 6, Paris-France, 1993.

[DS83] R. Davis and R. G. Smith. Negociation as a Metaphor for Problem Solving. *Artificial Intelligence*, 20 n° 01, 1983.

[EHRLR80] L-D. Erman, F. Hayes-Roth, V-R. Lesser, and R-D. Reddy. The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing surveys*, 12:213–253, 1980.

[EHRLR88] L.D. Erman, F. Hayes-Rothe, V. Lesser, and D. Reddy. *The HEARSAY-II Speech Understanding System : Integrating Knowledge to resolve Uncertainty. In Blackboard Systems.* Engelmore and Morgan (eds), 1988.

[FB88] J. Ferber and J-P. Briot. Design of a concurrent language for distributed artificial intelligence. *In International Conference on Fifth Generation Computer Systems (FGCS'88)*, volume 2:755–762, 1988. ICOT.

[Fer89] J. Ferber. Eco-Problem Solving: How to Solve a Problem by Interactions. In *Proceedings of the 9th Workshop on DAI*, pages 113–128, 1989.

[Fer91] Jacques Ferber. *Objets et agents: Une étude des structures de représentation et de communication en intelligence artificielle.* PhD thesis, Université de PARIS VI, LAFORIA Paris, Paris-France, 1991. Thèse d'etat.

[Fer92] I.A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents.* PhD thesis, Computer Laboratory, University of Cambridge, UK, 1992.

[Fer95] Jacques Ferber. *Les systèmes multi-agents, Vers une intelligence collective.* InterEdition, Paris-France, 1995.

[FFMM94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*. ACM Press, 1994.

[FHN71] R.E. Fikes, P.E. Hart, and N. Nilson. STRIPS: A New Approach to the Application of Theorem Proving. *Artificial Intelligence*, 2:189–208, 1971.

[FJ91] J. Ferber and E. Jacopin. The Framework of Eco-Problem Solving. In *Y. Demazeau and J.-P. Müller, editor, Decentralized A.I.*, volume 2, pages 181–193, North-Holland, 1991.

[FM94] J. Ferber and L. Magnin. Conception de systèmes multi-agents par composants modulaires et réseaux de Petri. In *Actes de sjournées du PRC-IA*, Montpelier, 1994.

[Fre94] D. Fresneau. *Biologie et comportement social d'une fourmi ponerine néotropicale.* Université de PARIS VIII, Paris, 1994. Thèse d'etat.

[FSH94] A. El Fallah-Seghrouchni and S. Haddad. Représentation et manipulation de plans à l'aide de réseaux de Petri. In *Actes des deuxièmes journées francophones sur l'intelligence artificielle distribuée et les systèmes multi-agents (JFIAD-SMA'94*, Voiron, 1994. Y. Demazeau et S. Pesty (Ed), IMAG.

[Gal90] J.R. Galliers. Modelling Autonomous Belief Revision in Dialogue. In Proceedings of MAAMAW'90 Decentralized Artificial Intelligence 3, editor, *Y. Demazeau et P. Müller*, Elsevier North Holland, 1990.

[Gas91] L. Gasser. Social Conceptions of Knowledge and Action : DAI Foundations and Open Systems Semantics. *Artificial Intelligence*, 47(1-3):107–138, 1991.

[Gau95] Daniel Gauvin. Un Environnement de Programmation Orienté Agent. In *Troisièmes Journées Francophones en Intelligence Artificielle Distribuée et Systèmes Multi-Agents*, pages 15–26, Savoie-France, 1995.

[GBh87] L. Gasser, C. Braganza, and N. herman. Implementing Distributed Artificial Intelligence Systems using MACE. In *Proceedings of the IEEE Conference on Artificial Intelligence Applications*, pages 315–320, 1987.

[GDD94] A. Geist, A. Deguelin, and J. Dongarra. PVM: Parallel Virtual Machine, A user's guide and tutorial for networked Parallel Computing. Technical report, MIT Press, 1994.

[GK94] Michael R. Genesereth and Steven P. Ketchpel. Software Agents. *Communication of the ACM, Intelligents Agents*, 37, 1994.

[GPS90] P. Gaborit, A. Potet, and C. Sayettat. Semantics and Validation Procedures of a Multi-Modal Logic for Formalization of Multi-Agent Universes. In *Proceedings of the European Conference in Artificial Intelligence*, pages 289–291, 1990.

[Gär88] P. Gärdenfors. *KNOWLEDGE IN FLUX, Modeling the Dynamics of Epistemics States.* The MIT Press, 1988.

[GRL87] A. Golding, P.S. Rosenbloom, and J.E. Laird. Learning general search control from outside guidance. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Milan, 1987.

[HBC+91] Jean-Paul Haton, Nadjet Bouzid, François Charpillet, Marie-Christine Haton, Brigitte Laasri, Hassan Laasri, Pierre Marquis, and Thierry-Mondot-Amedeo Napoli. *Le raisonnement en intelligence artificielle.* InterEdition, Paris, France, 1991.

[Hew77] C. Hewitt. Viewing Control Structures as Patterns of Message Passing. *Intelligence Artificial*, Vol. 8, n^{0}3, 1977.

[Hew85] C. Hewitt. The Challenge of Open Systems. *Byte*, 1985.

[HR85] B. Hayes-Roth. A Blackboard Architecture for Control. *Artificial Intelligence*, 26:252–321, 1985.

[Huh87] M.N. Huhns. *Distributed Artificial Intelligence.* Pitman, 1987. Huhns M.N (Ed).

[Ish90] T. Ishida. CoCo : A Multi-Agent System for Concurrent and Cooperative Operation Tasks. In 9th Distributed Artificial Intelligence Workshop, editor, *M. Benda (Ed)*, Orcas, Island, 1990.

[Jen95] Nicholas R. Jennings. Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems Using Joint Intentions. *Artificial Intelligence*, 75:195–240, 1995.

[JML+92] N.R. Jennings, E. Mamdani, I. Laresgoiti, J. Perez, and J. Corera. GRATE: a general framework for cooperative problem solving. *J. Intell. Sys Eng.*, 1(2):102–114, 1992.

[JW98] N.R. Jennings and M. Wooldridge. Applications of Intelligent Agents. In N.R. Jennings and M. Wooldridge, editors, *Agent Technology: Foundations, Applications, and Markets.* Published In Cooperation with UNICOM, 1998.

[Kle86a] Johan De Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[Kle86b] Johan De Kleer. Extending the ATMS. *Artificial Intelligence*, 28:231–272, 1986.

[KR90] L.P. Kaelbling and S.J. Rosenschein. Actions and Planning in embedded agents. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 35–48, 1990. P. Maes (Ed).

[Lal92] P. Lalanda. *Conduite de Raisonnement dans un Système à Base de Tableau Noir Temps Réel.* PhD thesis, Université de Nancy I, Nanacy-France, 1992.

[Lau87] J.L. Laurière. *INTELLIGENCE ARTIFICIELLE: résolution de problèmes par l'homme et la machine.* Eyrolles, Paris, 1987.

[LB92] J.C. Lementec and S. Brunnessaux. Atome-TR: A Real-Time Control for Blackboard Scheduling. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 255–256, 1992.

[LC83] V. R. Lesser and D. D. Corkill. The Distributed Vehicle Monitoring Testbed : A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine*, 4(3):15–33, 1983.

[Len75] D. Lenat. BEINGS: knowledge as interacting experts. In *Proceedings of the 4th IJCAI conference*, 1975.

[LH92] D.M. Lyons and A.J. Hendriks. A Practical Approach to Integrating Reaction and Deliberation. In *Proceedings of the 1st International Conference on AI Planning Systems (AIPS)*, pages 153–162, San Mateo, CA, 1992. Morgan Kaufmann.

[LLL+95] Yves Lespérance, Hector J. Levesque, Fangzhen Lin, Daniel Marcu, Raymond Reiter, and Richard B. Scherl. Fondements d'une Approche Logique à la Programmation d'Agents. In *Troisièmes Journées Francophones en Intelligence Artificielle Distribuée et Systèmes Multi-Agents*, pages 3–14, Savoie-France, 1995.

[LMH87a] Hassan Laasri, Brigitte Maître, and Jean-Paul Haton. ATOME: Outil d'aide au développement de systèmes multi-experts. In *Actes 6ème AFCET-RFIA*, pages 749–759, Antibes, France, 1987.

[LMH87b] H. Lâasri, B. Maître, and J.P. Haton. ATOME: Outil d'aide au développement de systèmes Multi-experts. In *Proceedings of the AFCET'87 Conference*, volume Vol. 2, pages 749–759, 1987.

[LMH88a] H. Lâasri, B. Maître, and J.P. Haton. ATOME: A Blackboard Architecture with Temporal and Hypothetical Reasoning. In *Proceedings of the ECAI'88 Conference*, pages 5–10, 1988.

[LMH88b] H. Lâasri, B. Maître, and J.P. Haton. Organisation, coopération et exploitation des connaissances dans les architectures de blackboard: ATOME. In *Proceedings of the Avignon'88 Conference*, volume Vol. 3, pages 371–390, 1988.

[LMM+87] Hassan Laasri, B. Maître, Thierry Mondot, François Charpillet, and J-P. Haton. ATOME: Another Tool for Developping Multi-Expert Systems. In *Proceedings AAAI-Workshop on Blackboard Systems*, Seatle, Washington-USA, 1987.

[LMM+89] H. Lâasri, B. Maître, T. Mondot, F. Charpillet, and J.P. Haton. Coordination de sources de connaissances opérant dans un environnement incomplet et évolutif: étude et réalisation dans ATOME. In *Proceedings of the Avignon'89 Conference: "Les systèmes experts de seconde génération"*, pages 237–251, 1989.

[LNR85] J.E. Laird, A. Newll, and P.S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1985.

[LRN86] J.E. Laird, P.S. Rosenbloom, and A. Newll. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[Mae80] P. Maes. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back.* MIT/Elsevier, 1980.

[Mat94] M.J. Mataric. Learning to Behave Socially. In Proceedings of the Third International Conference on Simulation of Adaptive Behavior From Animals to Animats 3, editor, *D. Cliff, P. Husbands, J.-A. Meyer et S.W Wilson*, pages 453–462, 1994. MIT Press.

[MC94] B. Moulin and L. Cloutier. Collaborative Work Based on Multiagent Architectures: A Methodological Perspective. In F. Aminzadeh et M. Jamshidi, editor, *Soft Computing: Fuzzy Logic, Neural Networks and Distributed Artificial Intelligence*, pages 261–296. Prentice-Hall, 1994.

[McF90] D. McFarland. *Dictionnaire du comportement animal.* Robert Laffont, 1990.

[MIT90] T. Maruichi, M. Ichikawa, and M. Tokoro. Modeling Autonomous Agents and Their Groups. In Distributed Artificial Intelligence 1, editor, *Y. Demazeau et P. Müller (Ed)*, North-Holland, 1990.

[MLF95] James Mayfield, Yannis Labrou, and Tim Finin. Evaluation of KQML as an Agent Communication Language. In *Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages*, pages 347–360, Amsterdam-The Netherlands, 1995.

[Moe89] J. Moeschler. *Modélisation du dialogue: représentation de l'inférence argumentative.* Hermès, 1989.

[Mor77] E. Morin. *La méthode (1): La nature de la Nature.* Le Seuil, 1977.

[MP93] J. P. Muller and M. Pischel. The Agent Architecture InteRRaP: Concept and Application. Technical Report 93-26, German Artificial Intelligence Research Center (DFKI), Saarbrucken, 1993. Research report.

[Mul96] Jorg P. Muller. *The Design of Intelligent Agents, A Layered Approach.* Springer, 1996.

[Nam97] R. Namyst. *PM²: Un environnement pour une conception portable et une exécution efficace des applications parallèles irrégulières sur architectures distribuées.* PhD thesis, Université de Lille I, Lille-France, 1997.

[NFAR88] H.P. NII, E.A. Feigenbaum, J.J. Anton, and A.J. Rockmore. *Signal-to-Symbol Transformation: HASP/SIAP Case Study. In Blackboard Systems.* Engelmore and Morgan (eds), 1988.

[Nii86a] H. Penny Nii. Blackboard Systems: the Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *The AI magazine, Summer*, pages 38–53, 1986.

[Nii86b] H. Penny Nii. Blackboard Systems: the Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *The AI magazine*, pages 82–106, 1986.

[NM95] Raymond Namyst and Jean-François Mehaut. PM²: Parallel Multithreaded Machine. A computing environment for distributed architectures. In *ParCo'95 (PARallel COmputing)*, pages 179–184, 1995.

[NS61] A. Newell and H.A. Simon. GPS: A Programm that Simulates Human Thought. In *Lernende Automaten*, Oldenbourg, KG, 1961.

[OPP99] L. Overgaard, H.G. Petersen, and J.W. Perram. Motion Planning for an Articulated Robot: A Multi-Agent Approach. In Proceedings of MAAMAW'94 Decentralized Artificial Intelligence 3, editor, *Y. Demazeau, J.P. Müller et J. Perram*, Odense Danemark, 1999.

[Pnu86] A. Pnueli. Specification and development of reactive systems. In *Information Processing 86.* Elsevier/North-Holland, 1986.

[Rad83] M. Rady. *L'ambiguïté du lagage naturel est-elle la source du non-déterminisme des procédures de traitement?* Thèse de Doctorat d'Etat de l'Université Paris 6, 1983.

[Rao96] A. Rao. Decision Procedures for Propositional Linear-Time Belief-Desire-Intention Logics. In *Intelligent Agents II: Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, Montréal-CANADA, 1996. Lecture Notes in Artificial Intelligence. Springer-Verlag.

[Réc81] F. Récanati. *Les énoncés performatifs.* Editions de Minuit, 1981.

[RD95] S. Regnier and D. Duhaut. Une approche multi-agent pour la résolution de problèmes robotiques. In *Actes des journées francophones sur l'intelligence artificielle distribuée et les systèmes multi-agents (JFIADSMA'95*, Chambéry, 1995.

[RG91] A.S. Rao and M.P. Georgeff. Modeling Agents Within a BDI-Architecture. In R. Fikes and E. Sandwall, editors, *Proccedings of the 2nd International Conference on Principles of Knowledge representation and Reasoning (KR'91)*, pages 473–484, cambridge, Mass., 1991.

[RG92a] A. Rao and M. Georgeff. An Abstract Architecture for Rational Agents. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992.

[RG92b] A. Rao and M. Georgeff. Social Plans: Preliminary Report. In Proceedings of MAAMAW'91 Decentralized Artificial Intelligence 3, editor, *E. Werner et C. Castelfranchi*, pages 127–146, Elsevier North Holland, 1992.

[RG95] A.S. Rao and M.P. Georgeff. BDI-agents: From Theory to Practice. In *Proceedings of the 1st International Conference on Multiagents Systems*, San Francisco, 1995.

[RZ94] J. Rosenschein and G. Zlotkin. Rules of Encounters: Designing Conventions for Automated Negociation among Computers. Technical report, MIT Press, 1994.

[Sab90a] G. Sabah. A Computational Model of Natural Language Understanding Using a Parallel Implementation. In *Proceedings of the ECAI'90 Conference*, Stokholm, 1990.

[Sab90b] G. Sabah. A Flexible Model for Interaction between the Cognitive Process Underlying Natural Language Understanding. In *Proceedings of the COLING'90 Conference*, volume 3, pages 446–448, helsinki, 1990.

[SBKL92] D. Steiner, A. Burt, M. Kolb, and C. Lerin. The Conceptual Framework for MAIL: An overview. Technical report, DFKI, 1992.

[Sea69] J-R. Searle. Speech acts: An Essay in the Philosophy of Language. *Cambridge University Press*, 1969.

[Sho76] E. Shortliffe. *Computer-Based Medical Consultation: MYCIN.* Elsevier, New York, 1976.

[Sho90] Y. Shoham. Agent-Oriented Programming. Technical report, STAN-CS-1335-90, Computer Science Department,Stanford University, Stanford, CA 94305, 1990.

[Sho93] Yohav Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60:51–92, 1993.

[Sim81] H.A. Simon. The sciences of the Artificial. Technical report, MIT Press, Cambridge, MA, 1981. 2nd edition.

[Smi80] R. G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Trans. on Computer*, 29(12):1104–1113, 1980.

[SP95] Aaron Sloman and Riccardo Poli. SIM_AGENT: A toolkit for exploring agent designs. In *Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages*, pages 292–407, Montréal-CANADA, 1995.

[SR83] G. Sabah and M. Rady. A Deterministic Syntactic-Semantic Parser. In *Proceedings of the IJCAI'83 Conference*, Karlsruhe, 1983.

[Ste90] L. Steels. Cooperation between Distributed Agents through Self-Organization . In Decentralized Artificial Intelligence, editor, *Y. Demazeau et P. Müller*, Elsevier North Holland, 1990.

[Ste94] L. Steels. The Artificial Life Robots of Artificial Intelligence . *Artificial Life*, 1(1), 1994.

[Syc89] K. Sycara. Multiagent Compromise via Negociation. In *Distributed Artificial Intelligence,Pitman*, 1989. Huhns M.N (Ed).

[SZ96] Katia Sycara and Dajun Zeng. Multi-Agent Integration of Information Gathering and Decision Support. In *Proceedings of 12th European Conference on Artificial Intelligence*, pages 549–553, 1996.

[Ter88] A. Terry. *Using Explicit Strategic Knowledge to Control Expert Systems. In: Balckboard Systems.* Engelmore and Morgan (eds), 1988.

[Tho93] S.R. Thomas. *PLACA, An Agent Oriented Programming Language.* PhD thesis, Stanford University, 1993.

[Tok93] M. Tokoro. The Society of Objects. In *Proceedings of the OOPSLA Conference*, 1993.

[TR95a] M. Tambe and P-S. Rosenbloom. RESC: An Approach to Agent Tracking in a Real-Time, Dynamic Environment. In *Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI)*, Montreal-CANADA, 1995.

[TR95b] Milind Tambe and Paul S. Rosenbloom. Architectures for Agents that Track Other Agents in Multi-Agent Worlds. In *Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages*, pages 156–170, Montreal-CANADA, 1995.

[Van88] D. Vanderveken. *Les actes de discours.* Pierre Mardaga, 1988.

[Wer89] E. Werner. Cooperating Agents: A Unified Theory of Communication and Social Structure. In *Proceedings of Distributed Artificial Intelligence*, volume II. L. Gasser et M. Huhns (Ed), 1989. Pitman.

[WJ94] M. Wooldridge and N. Jennings. Towards a Theory of Cooperative Problem Solving. In Proceedings of MAAMAW'94, editor, *Y. Demazeau et P. Müller et J. Perram*, Odense, 1994.

[WJM94] T. Wittig, N.R. Jennings, and E.H. Mamdani. ARCHON- A Framework for Intelligent Cooperation. *IEE-BCS Journal of Intelligent Systems Engineering*, 3(3):168–179, 1994.

[Woo94] Michael Wooldridge. This is MYWORLD: The logic of an Agent-Oriented DAI Testbed. In *Proceedings of the 1994 Workshop on Intelligent Agents; Agent Theories, Architectures, and Languages*, pages 160–178, Amsterdam-The Netherlands, 1994.

[Woo95] Michael Wooldridge. Time, Knowledge, and Choice. In *Proceedings of the 1995 Workshop on Intelligent Agents; Agent Theories, Architectures, and Languages*, pages 79–96, Montréal-Canada, 1995.

[Yon90] A. Yonezawa. ABCL : An Object-Oriented Concurrent System. Technical report, MIT Press, 1990.

[ZR92] G. Zlotkin and J.S. Rosenschein. A Domain Theory for Task Oriented Negociation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Chambery, 1992.