Maîtrise d'Informatique Option Tranversale
Université Paris 8

L'intelligence artificielle distribuée appliquée aux jeux d'équipe situés dans un milieu dynamique : l'exemple de la RoboCup

Benjamin DRIEU

11 octobre 2001

<bdrieu@april.org>
2, rue de la Liberté
93526 - Saint-Denis
France

Résumé

Ce mémoire décrit la conception et l'implémentation d'une équipe de robots footballeurs simulés par un ordinateur. Celle-ci est gérée au moyen d'un système multi-agents dans lequel chaque agent modélise un robot footballeur. Le cadre utilisé pour implémenter cette simulation est la RoboCup, une compétition internationale de football robotique et de football simulé. L'implémentation de cette équipe de robots simulés tient compte des problématiques du milieu (fortement dynamique et bruité) dans lequel les agents évoluent. Cette implémentation fournit un mécanisme pour contrôler les erreurs dans l'exécution des plans des agents, ainsi qu'un mécanisme de visualisation des actions en cours et un mécanisme de modélisation par hypothèses du comportement des coéquipiers.

Ce mémoire traitera aussi des problématiques de la conception et de l'implémentation d'une équipe en général ainsi que d'un système multi-agents dans un milieu dynamique. La RoboCup servira ici de cadre de test d'implémentations des systèmes multi-agents.

Table des matières

Remerciements

Nous remercions Nadia ABCHICHE pour la supervision de ce travail.

Nous souhaitons également remercier chaleureusement tout ceux qui ont rendu ce travail possible par leurs conseils, remarques et encouragements. Plus spécialement, un grand merci à Vanessa CONCHODON, à Mathieu IGNACIO, à Christelle KLOCK, à Arnauld MICHELLIZA et à Moïse VALVASSORI pour les relectures de ce mémoire à différentes étapes de son accomplissement et pour leurs témoignages d'amitié.

Également, un remerciement tout particulier à Mélanie CLÉMENT-FONTAINE, à Frédéric COUCHET et à Farzad FARID pour leur confiance, leur patience et leur ténacité.

Chapitre 1

Introduction

Ce mémoire décrit la conception et l'implémentation d'une équipe de robots footballeurs simulés. Cette simulation est réalisée par un système multi-agents dans lequel chaque agent modélise un robot footballeur. Son implémentation tient compte des problématiques du milieu (fortement dynamique et bruité) dans lequel les agents évoluent. Elle fournit de plus un mécanisme pour contrôler les erreurs dans l'exécution des plans des agents, ainsi qu'un mécanisme de visualisation des actions en cours et un mécanisme de modélisation par hypothèses du comportement des coéquipiers.

1.1 Présentation

La simulation décrite dans ce mémoire d'une équipe de football par un système multi-agents fait intervenir des agents autonomes et coopératifs, dont l'activité est tournée vers un but commun par l'accomplissement d'objectifs intermédiaires (nous parlerons en fait de *plans* intermédiaires). Le système multi-agents défini fait intervenir un mécanisme de coopération volontaire entre les agents d'une équipe, par le biais d'actions concertées.

Le cadre dans lequel cette simulation a lieu est celui d'un projet de recherche scientifique mondial : la Robocup. Le but de celle-ci est d'arriver à développer d'ici 2050 une équipe de football constituée de robots humanoïdes totalement autonomes et capable de battre l'équipe humaine championne du monde selon les règles de la fé-

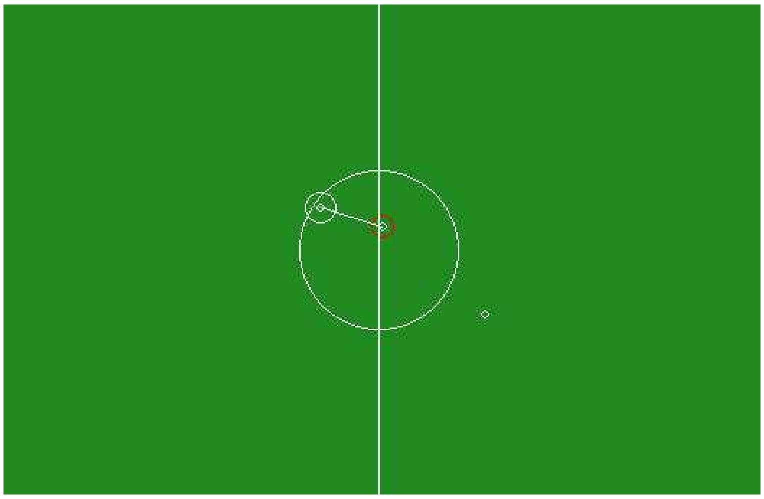

FIG. 1.1 – Visualisation de l'exécution du plan « prendre la balle ».

dération internationale de football.

Plusieurs catégories de rencontres existent actuellement : les robots de petite taille, les robots de taille moyenne, les robots quadrupèdes et enfin les programmes informatiques de simulation. Chaque catégorie fournit une base commune d'expérimentation pour toutes les équipes. Les trois premières catégories mettent en œuvre des robots autonomes et se focalisent principalement sur des problèmes physiques (robotiques). La dernière catégorie se focalise elle sur les problèmes d'intelligence artificielle et met les différents programmes informatiques concourant en relation au moyen d'un serveur de simulation et d'un protocole réseau de communication spécifique à la compétition.

Les contraintes de l'implémentation (pour la RoboCup) d'une équipe simulée par des programmes informatiques restent proches de celles de l'implémentation d'une équipe de robots réels afin d'avoir la simulation la plus réaliste possible. En effet l'environnement dans lequel les programmes évoluent est fortement dynamique (il évolue de manière asynchrone par rapport à l'agent) et bruité (les capteurs de l'agent sont imparfaits et incomplets). L'équipe simulée est donc confronté aux mêmes problèmes d'imperfection et d'incomplétude des capteurs que l'équipe réelle. De même, la communication entre agents est coûteuse et non fiable. L'implémentation décrite dans ce mémoire prend en compte ces paramètres et propose un mécanisme de contournement des problématiques de communication.

La motivation de la mise en place d'une simulation de robots footballeurs est l'abstraction des contraintes matérielles (financières ou techniques) de conception et d'implémentation de robots. De même, l'utilisation d'une simulation logicielle de robots permet d'expérimenter des techniques d'intelligence artificielle plus profondément que ne le permet la technologie robotique actuelle.

Le choix de la RoboCup comme base d'expérimentation tient à ce que la RoboCup est un terrain d'application remarquable des technologies multi-agents. En particulier, elle offre de nombreuses possibilités d'expérimentation : la conception de paradigmes robustes de coopération en milieu dynamique, le développement de protocoles de tolérance de pannes, l'adoption d'un comportement consistant, etc. La simulation d'une équipe de football permet de se confronter notamment aux nombreuses contraintes dynamiques et aux forts besoins d'interactions entre agents logiciels se partageant une faible bande passante. Ces problèmes sont ceux auxquels nous nous intéressons dans cette implémentation.

L'architecture du système multi-agents détaillé dans ce mémoire est basée sur la coopération de plusieurs agents. Son implémentation repose sur les processus des systèmes d'exploitation de type UNIX : chaque agent sera contenu dans un processus UNIX différent et sera donc autonome et distinct. Les interactions entre processus sont assurés par le serveur de simulation, qui contient une référence de l'état réel du monde et qui gère les communications entre agents.

Le comportement des agents est basé sur l'exécution de *plans*. Un plan est un enchaînement d'actions atomiques exécutées par un agent dans le temps et dans un objectif précis. Chaque instance de plan fait partie d'une classe de plans et définit des tâches qui sont exécutées selon une heuristique de classe. Les plans sont dans notre architecture des modules distincts, placés dans un agenda. Ces plans sont des processus capables de contrôler leur exécution et d'invalider leur existence. L'exécution des plans est effectuée sur un mode asynchrone (c'est-à-dire non synchronisé avec l'exécution du programme). L'utilisation d'un agenda permet d'exécuter le plan le plus important (c'est-à-dire en première position dans l'agenda) à chaque cycle d'exécution de l'agent. De plus, chaque plan comporte des informations visuelles lui permettant de produire une représentation symbolique à destination de l'opérateur en se basant sur l'affichage des symboles utilisés pour l'exécution du plan. Cette représentation est rendue possible par la définition d'une couche graphique de haut niveau, indépendante du support d'affichage.

Prenons pour exemple un agent ayant le plan de prendre la balle. À chaque cycle d'exécution, il met à jour l'état du monde en fonction des informations reçues par le serveur de la RoboCup. Puis il exécute un plan, qui produit une action atomique le rapprochant de la balle, en utilisant un effecteur de déplacement (avancer d'un pas, par exemple). Enfin, il met à jour la visualisation de son plan à destination de l'opérateur.

Les instances de plans en cours permettent de plus à un agent d'estimer le comportement de ses coéquipiers par l'utilisation d'une heuristique de classe. Cette heuristique reproduit l'exécution réelle d'un plan tel que l'ensemble de l'équipe l'exécute. L'implémentation d'un tel mécanisme d'évaluation comportementale structurellement partagé permet l'économie de communications (coûteuses en milieu dynamique bruité) au profit d'une synchronisation implicite basée sur des connaissances déduites.

En outre, nous proposons un mécanisme de contrôle du comportement des agents implémentés par l'exécution de scripts administratifs. Ces scripts sont écrits en langage Scheme et sont évalués à l'initialisation de l'agent ainsi qu'à la demande de l'opérateur.

1.2 Organisation de ce mémoire

Ce mémoire est organisé selon le plan suivant :

– le chapitre 2 présente les particularités d'un milieu fortement dynamique tel que celui dans lequel nos agents évoluent. La nature asynchrone de la relation des agents à leur environnement leur impose un traitement partiel des informations perçues ainsi que des stratégies d'action se basant sur des connaissances incomplètes ou des croyances non vérifiées. Nous commençons par décrire la nature dynamique du jeu d'équipe et en quoi une approche multi-agents peut aider à résoudre ses contraintes. Puis nous introduisons les différentes problématiques inhérentes au jeu d'équipe dans un milieu dynamique ainsi que les différentes solutions couramment apportées ;

– le chapitre 3 expose l'histoire et le cadre de la RoboCup ainsi que les moyens d'atteindre son objectif (produire d'ici 2050 une équipe de football robotique capable d'affronter et de vaincre l'équipe humaine championne du monde) par le biais d'échanges entre chercheurs du monde entier. Nous avons en effet utilisé cette compétition comme base d'expérimentation pour notre propre implémentation d'agents logiciels. Ce chapitre traitera de manière approfondie les différents aspects couverts par la catégorie simulation de la RoboCup, notamment les aspects opérationnels ;

– le chapitre 4 décrit la conception et l'implémentation du système multi-agents que nous avons expérimenté dans le cadre de la RoboCup. Ce système met en œuvre des agents logiciels écrits dans les langages C++ et Scheme. Elle est de plus basée sur les couches basses de l'équipe *CMU-2000*, décrite dans [25]. Ce chapitre décrit de plus les fonctionnalités (implémentées dans nos agents logiciels) de visualisation de l'état du monde et de compréhension des activités des partenaires ;

– le chapitre 5 définit la notion d'*agent*. En effet, la polysémie du terme *agent* complexifie la description d'architectures d'agents. Ce chapitre commence ainsi par définir le mot agent dans ses significations usuelles, puis il met en place une taxinomie des différents types d'agents existants. Il introduit ensuite quelques exemples de modèles d'agents et enfin détaille les composants habituellement trouvés au sein d'une architecture individuelle ;

– enfin, le chapitre 6 décrit les différents aspects d'une activité coordonnée entre agents (communication, compréhension et modélisation du partenaire, partage de buts, partage de tâches, planification, adoption d'une attitude, tolérance aux pannes). Il présente ensuite les aspects sociaux propres au jeu d'équipe ou particulièrement cruciaux pour les applications basées sur le jeu d'équipe ;

– enfin, nous concluons par l'analyse de notre implémentation et par le détail des perspectives envisagées et des issues des expérimentations conduites.

Chapitre 2

Contraintes d'un environnement dynamique

Nous appelons *milieu dynamique* un environnement dont les propriétés physiques ou structurelles sont susceptibles d'être modifiées immédiatement à tout moment. Un milieu dynamique est modifié du fait des agents qui y évoluent (actions des agents sur leur milieu) ou du fait de sa propre dynamique interne (évolution de ses composants). Dans le cas d'un environnement informatique, un milieu dynamique est souvent traité en *temps réel*. Le temps réel est un mécanisme où le déroulement des processus en cours ne découle pas d'une relation synchrone avec les agents situés dans l'environnement. C'est à dire que les propriétés d'un environnement évoluent sans être soumises à un cadençage imposé par l'agent.

Les contraintes d'un agent (qui est processus autonome) situé dans milieu dynamique sont particulières. La nature asynchrone de sa relation à l'environnement lui impose un traitement partiel des informations perçues, notamment en raison de leur obsolescence rapide. De même, les connaissances partielles et les croyances non vérifiées sur leur milieu mènent les agents situés dans un milieu dynamique à formuler des stratégies d'actions incomplètes ou basés sur des hypothèses.

Nous allons décrire dans ce chapitre la nature dynamique du jeu d'équipe et en quoi une approche multi-agents peut aider à résoudre ses contraintes. Puis nous passerons en revue les différentes problématiques inhérentes au jeu d'équipe dans un milieu dynamique ainsi que les différentes solutions apportées à celles-ci.

2.1 Le jeu d'équipe : une nature dynamique et distribuée

Nous définissons le jeu d'équipe comme l'affrontement de deux groupes de joueurs, au sein desquels ces derniers agissent simultanément et partagent un but commun : faire échouer le but de l'équipe adverse. Nous posons comme principe que le jeu d'équipe s'inscrit dans un environnement dynamique, en raison de la nature intrinsèquement dynamique des relations entre équipes. En effet, une équipe n'ayant aucun intérêt à adopter une démarche passive, la stratégie gagnante pour une équipe consiste à s'adapter aux changements du milieu, c'est à dire à faire évoluer chacun de ses membres au cours du temps afin de pouvoir atteindre le but fixé dans le nouvel environnement, de profiter d'une opportunité offerte par l'adversaire ou de s'adapter à une nouvelle stratégie adverse.

Au sein d'un milieu dynamique, une approche distribuée, qui consiste à répartir les organes de traitement de l'information au sein d'un système (ici, l'équipe), offre plusieurs avantages non négligeables par rapport à une approche purement centralisée, qui centralise les organes de traitement à un point donné du système :

– une expertise locale à un joueur où à une localisation est nécessaire au sein d'une équipe pour profiter des opportunités locales. La conception et l'utilisation d'agents logiciels permet d'encapsuler les expertises et de les utiliser dans un contexte local là où elles seront utiles ;

– la communication est plus coûteuse dans un milieu dynamique (l'établissement d'un canal de communication peut être problématique, la bande passante est généralement faible et les canaux sont souvent non fiables). La conception et l'utilisation d'agents permet d'économiser des communications grâce à leur autonomie ;

– la paradigme de la programmation agent s'applique au paradigme stratégique des équipes, où chaque joueur possède sa propre personnalité et ses propres buts, mais où il contribue à l'accomplissement d'un but commun.

2.2 Problématiques

Les milieux dynamiques apportent des contraintes de conception ou de fonctionnement supplémentaires au développement de systèmes multi-agents y agissant. Cette section va identifier des problématiques courantes, liées en partie à la dynamique de l'environnement et à son caractère bruité, ainsi que les moyens de les résoudre.

2.2.1 L'état du monde

Un agent interagissant avec un monde incomplètement connu doit être capable de raisonner sur les effets de ses actions et être capable d'obtenir des informations supplémentaires sur ce monde. Cette problématique est particulièrement sensible dans le cas d'un environnement dynamique. Malheureusement, les récepteurs d'un agent (robotique notamment) sont très souvent imprécis et ne couvrent qu'une fraction du monde. Dans l'exemple de la RoboCup, les robots footballeurs simulés ne possèdent qu'une vision partielle, bruitée (la précision de leur vue est faible) et conique (tout comme pour un oeil réel) du monde.

Pour résoudre ce problème, une des solutions possibles est la mise en place d'un mécanisme de quantification de la validité de croyances (connaissances non vérifiées) : l'agent décide que les informations sensitives reçues ne sont pas sûres et convergera vers une valeur précise et sûre au fur et à mesure que des informations sensitives confirment sa mesure initiale. Cette solution règle en partie le problème de l'imprécision des récepteurs.

Le problème de la perception partielle du monde est résolu par la mise au point d'hypothèses sur l'état de ce dernier. Lorsqu'un objet sort du champ de perception de l'agent, ce dernier émet une hypothèse sur sa position ainsi que sa nature, note la date d'émission de l'hypothèse et lui affecte une probabilité de vérité. Au fur et à mesure que le temps passe, la probabilité baisse graduellement jusqu'à atteindre un seuil en dessous duquel l'agent supprime cet objet de sa mémoire. C'est le cas des informations potentiellement changeantes comme la position de la balle sur le terrain dans le cas de la RoboCup. Les propriétés invariantes, telles que la taille d'un terrain de football dans l'exemple de la RoboCup, n'ont bien évidemment pas besoin d'un tel mécanisme de contrôle. De plus, une clause d'invalidation peut permettre de considérer une information comme obsolète et de supprimer l'objet de sa mémoire. C'est par exemple le cas de la trajectoire estimée d'un mobile sorti du champ de vision : si l'agent calcule que la trajectoire estimée de ce mobile devrait le faire réapparaître dans son champ de vision mais que ce n'est pas le cas, il en conclut donc que la trajectoire a été modifiée et que cette information n'est plus consistante. On trouve un exemple de ce mécanisme dans [29].

2.2.2 Le temps réel

Le « temps réel » est une conséquence quasi inévitable du milieu fortement dynamique. Le temps réel est un mécanisme causant une évolution du milieu sur une base temporelle non synchronisée avec l'évolution de l'agent Si un agent décide de ne pas agir pendant une période de temps ou s'il n'en a pas la possibilité, son environnement continue cependant à évoluer pendant cette période. Dans le cas d'une équipe, ce type de conséquences est dramatique car l'efficacité de l'équipe toute entière est diminuée quand un agent cesse de travailler. Les calculs liés au traitement des heuristiques ne doivent donc pas bloquer les agents et doivent être les plus limités dans le temps possibles afin de ne pas pénaliser l'efficacité de l'agent. De plus, certaines infrastructures de systèmes multi-agents définissent une notion de *cycles* (ou d'itérations), pendant lesquels un agent a une opportunité d'action. Le temps nécessaire à la formulation d'une action et le temps d'exécution de cette action ne doivent donc pas excéder la durée de ce cycle.

Le problème du temps réel est résolu par des méthodes logicielles de développement rigoureuses : l'utilisation d'une horloge envoyant des signaux (impulsions) réguliers à l'agent permet par exemple de cadencer son cycle de traitement. De même, l'agent peut tirer profit du parallélisme de leur architecture s'il en est pourvu et exécuter plusieurs tâches indépendantes simultanément (effectuer un long calcul ou une longue intégration d'informations pendant qu'il agit ailleurs).

2.2.3 L'imprédictibilité de l'environnement

Les milieux dynamiques sont par nature difficilement prédictibles. Des heuristiques peuvent être utilisées pour prédire des futurs locaux (déplacement d'un mobile par exemple), mais elles sont à la merci des interactions entre les conditions locales et les conditions globales (typiquement, un autre mobile vient modifier la course du premier). La perception partielle de l'environnement telle que nous l'avons décrite a de plus une incidence assez importante sur les heuristiques de prédiction d'un agent. Les hypothèses formulées sur l'évolution de l'environnement sont à confronter en permanence avec l'état changeant du monde au fur et à mesure que des croyances sont infirmées ou confirmées. Ainsi, pour accomplir une action d'interception de mobile, un agent aura besoin de formuler une hypothèse sur la course du mobile et le point d'interception supposé et aussi de suivre la course de ce mobile au fur et à mesure de sa course, afin de ne pas suivre un mobile « fantôme » [31].

L'imprédictibilité de l'environnement rend notamment difficile la formulation d'objectifs à long terme. On trouve cependant dans [9] un mécanisme de planification résolvant les problèmes d'imprédictibilité, d'ambiguïté et d'obsolescence dans le cas d'objectifs partagés entre agents. Les membres des systèmes multi-agents décrits par DURFEE et LESSER modélisent l'activité et suivent l'activité de leur système. Un agent de ce système à le pouvoir de décider d'abandonner un objectif s'il s'éloigne trop de l'état du monde tel que perçu. Il peut aussi communiquer ses réserves aux autres membres du système. Ceux-ci ont ainsi la possibilité de modifier leurs objectifs en fonction d'un critère d'importance (pondération) décidé par l'agent émetteur et de l'importance qu'ils accordent à leurs objectifs propres. Ce mécanisme de pondération peut permettre d'éviter de suivre un objectif obsolète ou d'abandonner un objectif valide sur la foi d'indications erronées ou obsolètes d'un agent voisin.

2.2.4 La communication bruitée

La nature changeante des milieux dynamiques peut rendre difficile la communication entre agents. Notamment, les canaux de communications peuvent ne pas être robustes (c'est-à-dire qu'ils sont sensibles aux pannes), ce qui implique un coût de communication important (par exemple, un agent peut être amené à effectuer des traitements supplémentaires pour émettre ou recevoir une information). Ces deux problèmes rendent non fiables les stratégies de coopération et de synchronisation fortement basées sur la communication. En effet, le coût de la dépendance à une information provenant de la communication entre agents augmente avec la non fiabilité de la communication. Des stratégies reposant sur l'autonomie totale ou partielle sont préférables.

On trouve dans [27] une méthode où les agents d'un système peuvent évoluer en quasi totale autonomie et utiliser des périodes courtes où ils disposent d'une bande passante totale pour se synchroniser et mettre à jour leurs plans. Notamment, pour s'assurer que les buts suivis par les agents sont tous compatibles. Cette méthode, appelée *Periodic Team Synchronisation* (*PTS*), a été développée à l'origine dans le cadre de la catégorie simulation de la RoboCup et prend tout son sens dans le jeu d'équipe : pour que l'attente de périodes de synchronisation à forte bande passante soit une stratégie réaliste, il est nécessaire que les rencontres durent suffisamment longtemps.

La technique du *set-play* est une alternative qui consiste à factoriser les efforts de coordination nécessaires au traitement des situations répétitives (dans le cas de la simulation de matchs de football, cela peut être le coup d'envoi, le hors-jeu, le dégagement, la touche, etc.). Il est payant de déterminer à l'avance des stratégies globalement optimales. Les *set-plays* sont des ensembles de rôles prédéterminés que chaque agent se chargera de remplir s'il le peut (un rôle est comportement guidé par des buts définis s'insérant dans la stratégie du groupe). Afin d'éviter les problèmes introduits par la communication non fiable, les *set-plays* doivent pouvoir se déclencher sans nécessiter de communication de la part des agents concernés. Ils peuvent donc être exécutés par le biais d'une condition activatrice, qui doit être connue par tous les agents du système et déterminée avant le début de la rencontre ou lors de la dernière période de synchronisation (*locker-room agreement*).

Chapitre 3

La RoboCup : un terrain d'application de la programmation agent

La RoboCup est un projet de recherche scientifique mondial, dont le but est de concevoir et de développer d'ici 2050 une équipe de football constituée de robots humanoïdes totalement autonomes et capable de battre l'équipe humaine championne du monde selon les règles de la fédération internationale de football. Les implications du projet sont transversales aux domaines de la recherche informatique et réunissent chaque année des centaines de chercheurs. Une compétition internationale a lieu tous les ans et elle donne l'occasion aux participants de confronter leurs équipes de joueurs robotiques. Plusieurs catégories de rencontres existent : rencontres de robots de petite taille, rencontres de robots de taille moyenne, rencontres de robots quadrupèdes et enfin rencontres de programmes informatiques.

Le projet de la RoboCup couvre un vaste champ d'applications, notamment en matière de robotique et d'intelligence artificielle :

– en développement logiciel, la mise au point de programmes intelligents capables de développer des compétences complexes en milieu dynamique et en temps réel (maintien de la balle, poursuite d'objets, etc.) ;

– en intelligence artificielle, la mise au point d'agents cognitifs (se reporter à la section 5.3.2 page 54) agissant en milieu dynamique bruité, capables de travailler de concert avec d'autres agents dans des actions coordonnées, mais aussi capables de prendre des décisions autonomes ;

– en robotique, le développement de robots mobiles, autonomes, complexes et rapides. La mise au point d'effecteurs et de récepteurs fiables malgré les contraintes issues de la nature dynamique du milieu ;

– en traitement d'images, la reconnaissance en temps réel d'informations bruitées provenant de la caméra des robots footballeurs.

Ce chapitre passe en revue l'histoire de la RoboCup, puis il décrit de manière approfondie les différents aspects couverts par la catégorie simulation de la RoboCup, notamment son aspect opérationnel.

3.1 Histoire de la RoboCup

La RoboCup telle que nous la connaissons a été initiée lors du symposium d'intelligence artificielle *JSAI AI-Symposium 95* [14, 21]. Hiroaki Kitano propose dans [14] d'utiliser la RoboCup comme un problème commun tourné vers un objectif compétitif : une compétition lourde mais durable permettrait de développer plus efficacement des domaines de recherche usuellement peu atteints par les compétitions plus modestes. D'un projet initialement Japonais (débuté en 1993), la RoboCup est devenue rapidement un projet d'envergure internationale.

Afin d'éprouver l'avancement des travaux sur la RoboCup, des rencontres ont lieu tous les ans pour confronter les différentes équipes robotiques des participants au projet. Ces rencontres sont accompagnées de conférences permettant d'établir l'état de l'art et de décider des évolutions futures du projet. Les rencontres sont organisées sur le modèle d'une vraie compétition sportive et donnent lieu à la publication d'un palmarès. La compétition mettant en œuvre des programmes informatiques impose de plus que les participants rendent disponibles les sources de tout ou partie de leur programme à la suite de la compétition (nous souscrivons à cette approche et considérons que la disponibilité du code source d'un programme est la meilleure documentation qui existe et la meilleure manière de faire évoluer la science tout entière).

La RoboCup n'est pas une compétition figée. D'une manière générale, la philosophie de la RoboCup est de complexifier graduellement les règles et les principes de la compétition au fur et à mesure que les techniques de base sont maîtrisées. Ainsi, en 1998, la compétition s'est étendue pour faire concourir des robots quadrupèdes, développés par *Sony*. La mise au point d'une compétition de robots bipèdes est en cours et le nombre de robots impliqués dans chaque aspect de la compétition augmente graduellement au fil des ans, jusqu'à atteindre le nombre de onze par équipe.

Les différentes catégories de compétitions affiliées à la RoboCup, existantes ou proposées, sont :

- la catégorie des petits robots (moins de 180 centimètres carrés), comportant jusqu'à cinq robots par équipe ;
- la catégorie des robots de taille moyenne (d'environ 250 centimètres carrés), comportant jusqu'à onze robots par équipe ;
- la catégorie des robots Sony quadrupèdes, comportant trois robots par équipe ;
- la catégorie des robots humanoïdes, en cours de définition ;
- la catégorie simulation, comportant onze programmes par équipe.

3.2 La catégorie simulation

La catégorie simulation a pour but de modéliser des rencontres entre deux équipes de onze footballeurs. La raison d'être de cette catégorie est de s'affranchir des contraintes matérielles de la mise en place d'une équipe de robots footballeurs (reconnaissance de la vision, problèmes physiques, coûts de mise en place, etc.) et de se concentrer sur l'intelligence artificielle, les paradigmes de communication ainsi que les modèles de représentation du monde extérieur.

Les champs de recherche couverts par la catégorie simulation de la RoboCup sont parmi les suivants :

– la modélisation d'agents autonomes et la sélection d'une action pour réagir à une situation imposée par un milieu dynamique ;

– la modélisation par un agent de la structure de son équipe et des interactions sociales entre joueurs, le travail coopératif entre les joueurs et la formation de groupes de travail au sein d'une équipe ;

– l'apprentissage automatique et l'entraînement de ses compétences ;

– la formulation de plans, la tolérance à l'échec et la modélisation des plans de ses adversaires ;

– la mise à l'épreuve des architectures d'agents dans des milieux multi-agents fortement dynamiques.

3.2.1 Le simulateur

Les équipes de programmes footballeurs s'affrontent par le biais d'un serveur de simulation développé par Itsuki NODA, [20]. Chacune des équipes est distribuée en plusieurs processus et est composée d'autant de programmes que de joueurs, se connectant tous sur le serveur par le biais de connexions de type UDP/IP. Chaque joueur est donc un processus isolé des autres, n'ayant aucun moyen de communiquer directement avec les autres ni de partager sa mémoire. Chacun des joueurs est représenté identiquement par le serveur, sauf le gardien de but qui possède un effecteur supplémentaire pour pouvoir attraper la balle. Un agent spécial est également disponible, l'entraîneur. L'entraîneur a une vision extérieure du jeu mais ne peut pas y agir physiquement.

Le but du serveur de simulation de la RoboCup est d'éviter les problématiques de mise au point mécanique. La simulation de la RoboCup conserve néanmoins la complexité du traitement de l'information et de la perception du monde. Le serveur approxime les conditions d'un milieu réel : la perception de l'environnement tout comme les communications sont bruitées. L'environnement simulé n'est pas discrétisé et les effecteurs de chacun des agents sont proches de leurs équivalents physiques réels. Par exemple, malgré leur volonté, les agents se déplacent moins vite lorsqu'ils sont fatigués. Enfin, l'architecture des canaux de communication informatiques entre les joueurs et le serveur (utilisation d'UDP, mécanisme rapide mais non fiable) rend impossible la prédiction de la réussite ou de l'échec de la communication avec le serveur.

Le simulateur et les joueurs communiquent grâce à un protocole réseau défini dans [26]. Le serveur envoie à intervalles réguliers des informations sur l'état du monde au client et ce dernier agit sur le monde en envoyant des requêtes définies par le protocole de la RoboCup au serveur. Il est intéressant de noter que le serveur n'accepte qu'une et une seule action par intervalle de temps déterminé. Dans le cas où un client en enverrait plusieurs, le serveur en choisi une au hasard.

3.2.2 Les récepteurs simulés

Le monde simulé est un terrain de football de dimension standard et dont les coordonnées sont cartésiennes. L'origine du repère est le centre du terrain et la perception de son orientation est différent pour les deux équipes. La simulation prend en compte deux dimensions, ce qui simplifie le jeu par rapport aux problématiques de la troisième dimension. Voir la figure 3.1 page suivante, pour un exemple de rencontre simulée et visualisée par le programme observant le déroulement de la simulation.

Les agents connectés sur le serveur de simulation reçoivent régulièrement des informations visuelles (en moyenne toutes les cent cinquante millisecondes) des informations sur leur état corporel (en moyenne toutes les cent millisecondes) et des informations auditives (dès qu'une telle information est disponible, c'est-à-dire lorsqu'un autre agent s'exprime ou lorsque l'arbitre fait circuler un message).

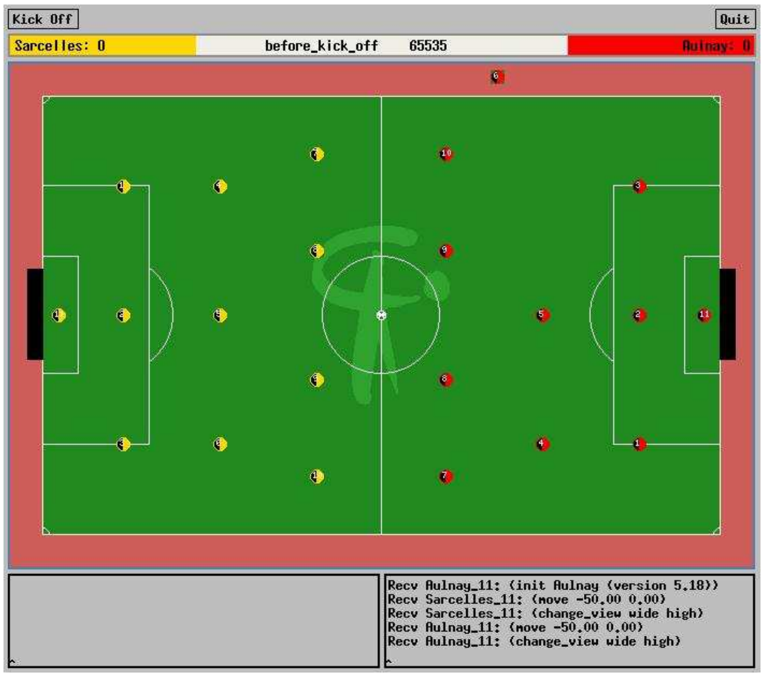

FIG. 3.1 – Le moniteur

Les informations visuelles reçues sont relatives et leur précision décroît linéairement en fonction de la distance à un objet perçu (il est par exemple possible de ne pas reconnaître le numéro d'un coéquipier éloigné). Les informations visuelles sont composées de l'ensemble des coordonnées polaires des objets se situant dans le cône de vision de l'agent. Ce mécanisme reproduit assez fidèlement les contraintes d'une vision réelle et impose l'utilisation de mécanismes de mémorisation des informations perçues tout comme de suivi d'objets mobiles.

Les informations auditives ne sont pas circonscrites à une direction mais sont de portée limitée (50 mètres). De plus, l'utilisation concurrente du canal sonore introduit une incertitude sur la destination d'un message : seul un message (déterminé aléatoirement parmi les messages émis concurremment) est entendu par les agents présents dans un cercle d'audition. Cette problématique implique une bande passante très faible pour le transfert d'informations, tout comme une fiabilité très relative de ce canal de communication.

Le joueur reçoit régulièrement des informations sur son propre corps : la position de sa tête par rapport à son corps (il est possible de faire tourner sa nuque) et surtout son état de fatigue. L'efficacité des mouvements des joueurs est en effet pondérée par la quantité de fatigue physique accumulée. Dans la pratique, les joueurs se fatiguent d'autant plus vite qu'ils accomplissent des efforts soutenus. Il est donc important pour chaque joueur d'avoir une idée assez précise de son propre état de fatigue : les informations corporelles envoyées au client sont des informations de première importance pour l'établissement d'une politique de gestion des efforts et du passage de la balle.

3.2.3 Les effecteurs simulés

Les joueurs ont la possibilité de manipuler le monde par le biais de plusieurs effecteurs. Lorsqu'ils ont le désir d'accomplir une action, ils envoient une requête au serveur, qui se charge de déterminer sa validité et de l'accomplir le cas échéant. Le résultat perçu d'une action peut survenir ultérieurement (en raison, par exemple, du bruit affectant les percepteurs, qui limite la perception des conséquences de cette action), les agents simulés ont donc besoin d'agir de manière asynchrone et de prendre des initiatives sans attendre de retour des actions précédentes.

Il est à noter que le serveur de simulation discrétise le temps en cycles d'un dixième de secondes. Un agent ne doit normalement effectuer qu'une action par cycle. Si un agent tente d'exécuter plusieurs actions par cycle, le serveur en choisit une aléatoirement parmi celles envoyées et exécute uniquement celle dernière sans prévenir le client.

Les agents simulés de la RoboCup disposent des effecteurs suivants pour manipuler le monde qui les entoure ou se déplacer au sein de leur environnement :

– attraper la balle (uniquement pour le gardien de but). Le gardien de but est capable de déterminer une zone rectangulaire en face de lui, dans laquelle il a une certaine probabilité d'attraper la balle ;

– avancer d'un pas. Les agents peuvent avancer vers la direction dans laquelle ils sont tournés. Cette avancée est fonction d'un paramètre (de vitesse) spécifié lors de la transmission de la requête au serveur. De la fatigue est produite à chaque utilisation de cet effecteur et elle dépend de la vitesse utilisée. Plus un joueur est fatigué, moins il avancera vite et moins il sera efficace dans la manipulation de la balle ;

– frapper la balle. Si la balle est dans une zone proche du joueur (qui est représentée par un cercle centré sur le joueur), il peut la manipuler et lui imprimer une accélération dans une direction spécifiée. L'accélération apportée est spécifiée au serveur par un paramètre et dépend de la fatigue du joueur ;

– se déplacer immédiatement à un point donné du terrain. Avant le début du match, le joueur peut se *téléporter* automatiquement et immédiatement à un point donné du terrain, afin de faciliter le placement des joueurs. En cours de jeu, cette manœuvre n'est pas possible ;

– émettre un message sonore (parler). Le message est entendu par défaut dans un rayon de 50 mètres, y compris par les adversaires ;

– tourner le corps dans une direction. Lorsque le joueur avance, il le fait vers la direction dans laquelle il est tourné. Pour modifier la direction de sa course, il est donc nécessaire de faire tourner le corps ;

– tourner la tête vers une direction. Tourner la tête dans une direction qui n'est pas celle du corps permet de suivre un objet à la trace tout en avançant dans une direction différente (par exemple dans le cas d'une interception de balle).

3.2.4 L'agent entraîneur

L'agent entraîneur (*coach*) simule l'entraîneur d'une équipe de football. Il s'agit d'un agent spécial qui a une vision plus globale du jeu que les joueurs eux-mêmes. Il peut de plus donner des ordres aux joueurs afin de les diriger, tout comme le ferait un véritable entraîneur.

L'entraîneur s'utilise de deux manières différentes, paramétrées au démarrage du serveur :

- en mode *coach*. Le *coach* a simplement une vision plus large et non bruitée du jeu et peut envoyer des indications, des conseils et des ordres aux joueurs (via l'utilisation d'un langage de communication généraliste et indépendant de la structure et de la conception de l'équipe). De plus, il peut nommer des zones de la surface de jeu et les communiquer aux clients ;

- en mode *entraineur* (ou *coach off-line*). L'*entraineur* a les mêmes options que le coach mais peut intervenir sur le déroulement du jeu, siffler les pénalités au même titre que l'arbitre et peut déplacer les joueurs où bon lui semble lors de la partie (ce qui est très utile pour entraîner les joueurs en dehors du terrain, par exemple pour les équipes utilisant l'apprentissage automatisé ou *machine learning*).

Le coach est un mécanisme introduit dans la compétition RoboCup de 2001 pour développer un client axé sur le développement de stratégies. En effet, le coach est moins sollicité physiquement que les joueurs en raison de sa position externe au jeu. Il peut donc consacrer plus de temps à la réflexion. De plus, il possède une vision complète et non bruitée du jeu, ce qui lui permet de formuler des stratégies beaucoup plus fines. Par ailleurs, l'utilisation d'un langage de communication indépendant des paradigmes de l'équipe simulée permet de développer des coachs indépendants d'une équipe et de les utiliser pour l'une ou l'autre des équipes développées.

3.2.5 L'agent arbitre

L'arbitre est lui aussi un agent logiciel. Il est chargé du bon déroulement de la rencontre et de la gestion du temps de jeu. L'arbitre surveille le déroulement de la partie et peut réguler le jeu par exemple lorsque la balle sort de la surface de jeu, lorsqu'il y a hors-jeu (passe à un joueur trop avancé sur le terrain), lors qu'un but est marqué, etc.

L'arbitre est secondé par un humain, qui peut décider de placer la balle à un endroit déterminé du terrain ou d'accorder une pénalité à un joueur (à cause d'un comportement de son adversaire que l'arbitre informatique ne peut pas automatiquement déterminer, par exemple l'obstruction de la balle). Lorsque l'arbitre prend une décision, le serveur notifie tous les joueurs de son effet via le protocole de communication. Ces décisions peuvent entraîner le déplacement de joueurs sur le terrain ou l'interdiction de s'approcher de la balle pour une équipe tant que l'autre équipe n'a pas remis la balle en jeu.

3.2.6 Déroulement d'une rencontre

Les rencontres de football simulées se déroulent selon une procédure proche de celle d'un match réel. Une rencontre simulée dure deux mi-temps de cinq minutes de temps réel chacune. Avant le coup d'envoi, les joueurs peuvent se déployer où bon leur semble sur leur moitié de terrain et s'échanger des messages comme bon leur semble. Une fois le coup d'envoi donné, il ne leur est plus possible de se déplacer que par la marche. Lors de la mi-temps, les joueurs sont à nouveau placés dans des conditions proches de celles du début de la rencontre jusqu'au coup d'envoi de la deuxième mi-temps. Lors des compétitions officielles, c'est aussi le moment où les agents peuvent se resynchroniser et remettre à jour leurs tactiques.

Lorsqu'une faute est commise, l'arbitre informatique siffle l'arrêt momentané de la partie et les joueurs sont avertis de sa décision. L'équipe ayant droit au ballon a l'initiative de se déplacer jusqu'à lui et de le remettre en jeu (et donc de sélectionner une tactique de remise en jeu ainsi que de déterminer le joueur allant chercher le ballon).

3.3 Évolution de la RoboCup

Une des caractéristiques de la RoboCup est sa forte évolution. En effet, les équipes qui y participent sont fortement encouragées à publier tout ou partie de leurs sources ainsi qu'une version utilisable de leurs programmes. Ainsi, les participants peuvent confronter leurs équipes aux équipes des années précédentes et profiter de leur expérience pour améliorer leurs bases.

Les premières rencontres de la RoboCup ont été caractérisées par une évolution importante des activités basses des agents footballeurs (par exemple, une équipe a acquis un avantage énorme lors d'une des premières confrontations en mettant au point un mécanisme de contrôle de balle lui permettant de la manipuler comme si ses joueurs la tenaient entre leurs mains), jusqu'à atteindre un niveau d'équilibre. À partir de 1998, les principales évolutions ont eu lieu dans les aspects coopératifs et de représentation des connaissances. L'utilisation d'un mécanisme de modélisation de l'adversaire et du coéquipier (voir [30]) a par exemple permis à l'équipe CMU-2000 de marquer des buts dans des conditions où la prise de décision est déterminante.

3.4 La RescueCup

La RescueCup est un projet dérivé de la RoboCup. Son origine est tirée de la constatation de l'inefficacité des secours actuels en cas de désastre naturel important, comme cela a été illustré lors de la catastrophe de Kobé au Japon (où un tremblement de terre à causé la mort immédiate de plus de 6000 personnes). Notamment, les limites suivantes ont été identifiées dans la technologie de l'information telle qu'elle est appliquée aux situations d'urgence :

- erreurs d'estimation d'échelle ;
- dégâts dans les centres de réponse d'urgence et sur leur personnel ;
- coupure ou congestion des canaux de communication ;

FIG. 3.2 – La RescueCup : Simulation de catastrophe

– non-information des civils et des volontaires ;

– support d'information insuffisant en cas de décision.

Ces problèmes sont très proches de ceux auxquels sont confrontés les systèmes multi-agents en milieu fortement dynamique. Le projet de simulation RoboCup-Rescue est une tentative de résolution de ces problèmes. Il se concentre principalement sur :

– l'acquisition, l'accumulation, le relais, la sélection, l'analyse, le résumé et la distribution des informations nécessaires ;

– le support informatif nécessaire à la décision ;

– la distribution des systèmes pour augmenter leur fiabilité et leur robustesse ;

– la continuité opérationnelle, des conditions normales aux situations d'urgence.

Le but de la RescueCup est similaire en esprit à celui de la RoboCup : en 2050, la technologie humaine devra être capable de mettre en place des équipes de robots qui sauveront des vies en situation réelle et lors d'une catastrophe de première ampleur. Actuellement, la RescueCup se situe en marge de la RoboCup et fait intervenir des équipes d'agents simulés (voir la figure 3.2, page précédente).

Chapitre 4

Une implémentation d'équipe pour la simulation RoboCup

Afin de mettre en pratique la théorie acquise en matière de systèmes multi-agents, nous avons implémenté une équipe d'agents footballeurs, conformément aux spécifications de la catégorie simulation de la RoboCup. Cette équipe d'agents nous permettra d'expérimenter les différentes techniques nécessaires au jeu d'équipe et de la confronter à des problématiques réelles.

Notre équipe est basée sur l'implémentation de l'équipe *CMUnited 2000* (*CMU-2000*) décrite dans [25, 28, 29], développée à l'université *Carnegie Mellon* et vainqueur des compétitions de 1998 et de 1999. En tant que programme vainqueur, l'équipe CMUnited a eu l'obligation de fournir une partie de son code source tout comme un manuel détaillant les techniques utilisées et les domaines de recherche explorés par l'équipe. Notre choix est motivé par le fait que l'équipe CMU-2000 et ses versions précédentes se sont distinguées des autres équipes par une série d'effecteurs et de capacités bas niveau conçue de manière efficace. De plus, CMU-2000 est développé avec une volonté de réutilisation et une architecture modulaire, ce qui facilite son intégration dans un programme quelconque. Nous avons utilisé les couches basses (couche et protocole réseau, analyse des informations sensorielles, effecteurs bas niveau, exécution temporelle asynchrone) de CMU-2000 afin de nous concentrer sur les parties comportementales du programme.

4.1 Langage d'implémentation

Notre implémentation utilise un mélange des langages *C++* et *Scheme*. La motivation du choix du langage C++ pour le corps du programme est un compromis entre rapidité d'exécution (le programme doit être capable de se comporter en temps réel) et confort de programmation. Le langage C++ offre des possibilités objet intéressantes, utiles dans l'implémentation d'un système multi-agents [32] : les fonctionnalités objet standard (héritage, surclassage, encapsulation, etc.), une bibliothèque de manipulation de données de haut niveau (*STL*), un symbolisme de niveau intermédiaire, etc. La nature intrinsèquement modulaire de l'architecture d'un agent impose de fait l'utilisation d'un langage objet, une partie de la méthodologie objet étant pratiquement identique à celle de la méthodologie agent [37]. Par ailleurs, malgré des déficiences connues dans l'implémentation de certaines composantes de la norme C++, le compilateur *g++* de GNU fournit un bon environnement de développement multi-plates-formes. De plus et ce n'est pas une mince motivation, l'utilisation du langage C++ nous permet de réutiliser plus étroitement les composants définis par l'équipe CMU-2000.

La motivation de l'utilisation du langage Scheme est l'ajout de fonctionnalités de script, d'évaluation à la volée d'instructions symboliques de haut niveau et de manipulation de structures de données complexes par les agents composant l'équipe de football simulée. Cette fonctionnalité est déjà utilisée lors du démarrage du programme (lecture d'un fichier d'initialisation), où elle permet de faire exécuter des opérations initiales à chacun des agents (se placer sur le terrain en fonction de la stratégie d'équipe par exemple) et en cours de jeu à la lecture de l'entrée standard. Nous envisageons de plus l'utilisation de ce langage de script pour symboliser des actes de langage des agents de notre système, tout comme pour formaliser les arbres décisionnels des organes de décision. Le choix de Scheme en tant que langage d'extension interprété a été effectué sur la base de notre expérience avec ce langage, sur les possibilités offertes (aspect fonctionnel, représentation de structures de données complexes, etc.) mais aussi sur l'implémentation Scheme de GNU (*guile*), dont la qualité est un argument déterminant. Guile est composé d'une bibliothèque de primitives permettant d'inclure un interpréteur Scheme dans un programme compilé (C, C++, etc.). Cette inclusion permet d'accéder réciproquement aux données de l'interpréteur Scheme et du programme et de définir des méthodes Scheme exécutant du code compilé. Une problématique reste cependant soulevée : une correspondance fine entre la structure objet de notre programme et le paradigme fonctionnel du langage Scheme reste à implémenter.

4.2 Architecture de l'agent

L'architecture utilisée pour notre agent est de type horizontale (voir la section 5.2.1, page 48). Nous avons notamment défini des modules indépendants pour l'ordonnancement des tâches, pour la représentation du monde (tableau noir). Les récepteurs ainsi que les effecteurs sont distincts du reste de l'agent. La figure 4.1, page suivante résume l'architecture utilisée.

Le déroulement de l'exécution de l'agent est conditionné par l'exécution de plans. Les plans sont des modules dont chaque instance possède des informations sur sa représentation graphique ainsi que sur sa propre validité. Une instance de plan est capable de générer une autre instance de plan, dédiée à l'atteinte des conditions nécessaires à son exécution. À chaque cycle d'exécution, déterminé par une « horloge » et calqué sur le cycle d'exécution du serveur de simulation (voir la section 3.2.3, page 23), un « top d'horloge » (un signal UNIX) est envoyé au programme, ce qui déclenche une action de l'*ordonnanceur*. L'ordonnanceur est l'organe chargé de choisir le plan à exécuter. Dans l'implémentation actuelle il choisit le plan le plus urgent, c'est-à-dire celui qui se trouve en première position de l'agenda. La conservation de l'état du monde est, quant à elle, assurée par un mécanisme de « tableau noir » décrit dans la section 4.4, page 35.

Le serveur de simulation de la RoboCup interagit avec les clients de manière asynchrone, c'est-à-dire sans synchronisation de leurs interactions. Le serveur communique par l'émission à intervalles réguliers de messages sensitifs ou informatifs ainsi que par l'évaluation régulière des requêtes des clients. La programmation d'un agent simulé de la RoboCup nécessite donc d'implémenter une boucle basée sur la réception de messages et sur leur traitement. À chaque fois que l'agent reçoit des infor-

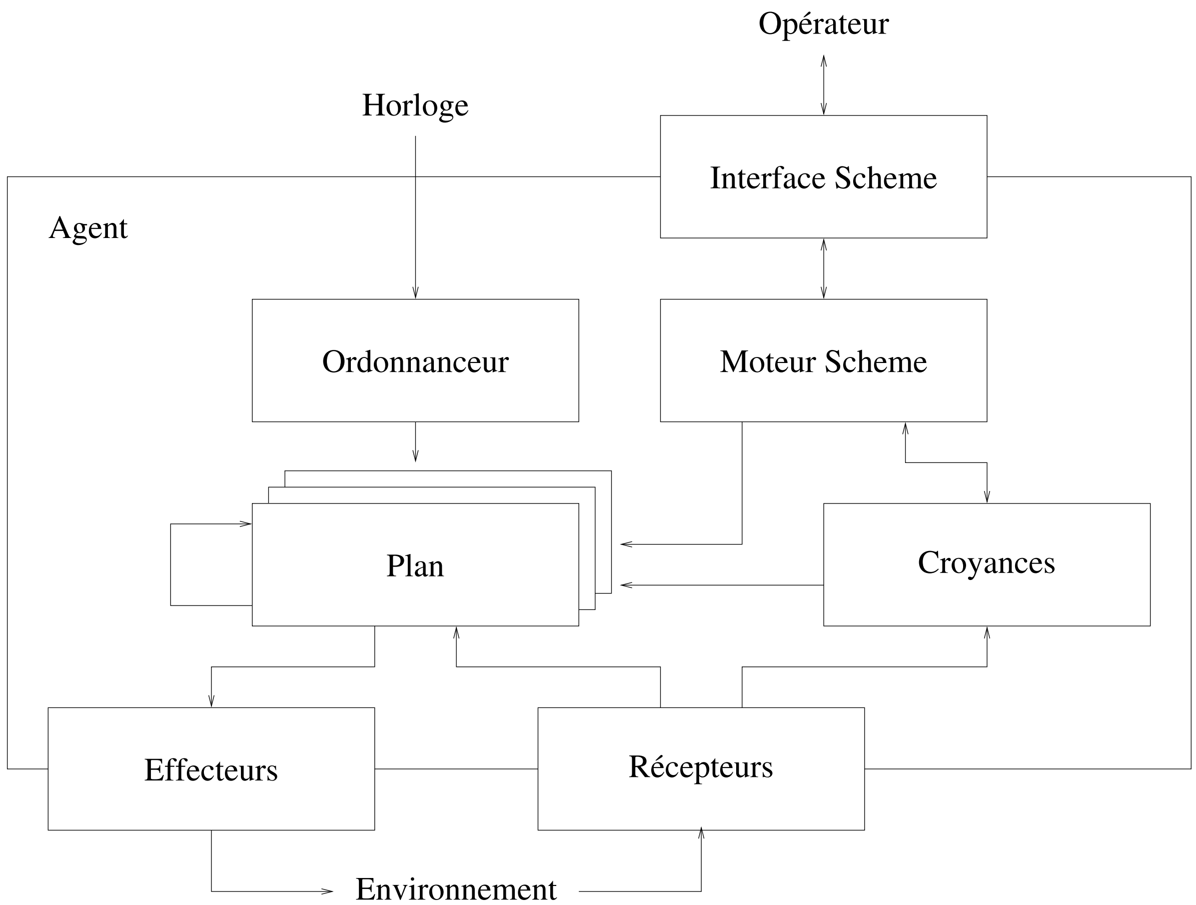

FIG. 4.1 – Architecture de notre implémentation

mations sensorielles, il met à jour l'état du monde qu'il avait maintenu jusque là, réévalue son action actuelle en fonction du nouvel état obtenu et exécute une action en fonction de ses buts mis à jour. Il s'agit d'une contrainte de programmation assez forte, qui oblige à maintenir d'un cycle à l'autre des informations dans un contexte global. Ainsi, les méthodes utilisées pour accomplir une action doivent se passer au maximum de résultats intermédiaires, être réentrantes (pouvoir s'exécuter plusieurs fois dans des conditions différentes) autant que possible. Cette problématique est particulièrement cruciale et limitante pour le mécanisme d'extension basé sur l'inclusion d'un interpréteur *Scheme* (voir la section 4.1, page 31), que nous avons développé au sein de notre agent. Cependant, elle peut être limitée par l'utilisation de variables d'instances afin de sauvegarder les états intermédiaires des plans de l'agent et ainsi limiter les problèmes de l'exécution sans état.

4.3 Interfaces avec l'environnement

Les agents implémentés sont en interaction avec le monde par le biais de leurs effecteurs, de leurs récepteurs et d'une interface d'administration utilisant un moteur Scheme.

Les effecteurs définis sont des actions atomiques calquées sur le protocole de communication du serveur de la RoboCup. En effet, toutes les primitives d'action définie dans le protocole sont considérées comme un effecteur. Les différentes primitives disponibles sont détaillées dans la section 3.2.3, page 23. Comme le serveur ne renvoie pas d'information sur l'effet de l'action d'un effecteur, c'est à l'agent d'estimer la réussite de l'action d'un effecteur en fonction de l'évolution d l'état du monde.

Les récepteurs définis sont eux aussi calqués sur les primitives définies dans le protocole. Ainsi, les informations perçues sur l'environnement sont de type visuelles et sonores. Ces sources d'information sont bruitées, comme décrites dans la section 3.2.2, page 21.

Enfin, une interface avec l'opérateur permet à l'agent d'exécuter des instructions écrites en langage Scheme. Cette interface textuelle est assurée par l'entrée clavier des processus des agents. L'opérateur peut ainsi contrôler l'exécution du comportement de l'agent a tout moment.

4.4 Représentation de l'environnement

Le CMU-2000 fournit une classe *PositionInfo*, qui hérite des méthodes et des variables de plusieurs autres classes de l'implémentation (*PositionInfo*, *PlayerInfo* et *OptionInfo*). La classe *PositionInfo* contient toutes les informations déduites de la communication avec le serveur de la simulation, soit une image symbolique mais brute de l'état du monde. Le CMU-2000 modélise les objets de l'environnement par une suite de classes géométriques, décrites dans la figure 4.2, page suivante. Parmi les informations contenues dans la classe *PositionInfo* :

– les coordonnées absolues de l'agent ainsi que l'orientation de son corps et de sa nuque ;

– des pointeurs vers les différents objets représentant les joueurs et le ballon (certains de ces objets pouvant ne pas être identifiés en raison de l'imprécision des récepteurs) ;

– des informations temporelles sur les différents éléments de l'environnement (par exemple, la date de dernière vision d'un objet) ;

– différentes informations sur la configuration du serveur et du client (la taille du terrain ou le nom des deux équipes par exemple).

La classe *PositionInfo* fournit de plus de nombreux accesseurs (méthodes permettant d'accéder aux données de la mémoire) et de nombreux facilitateurs (méthodes permettant de simplifier l'accès et la modification de structures de données complexes en mémoire). Notamment, elle fournit des facilitateurs définissant des prédicats simples sur les objets de l'environnement (distance à la balle, calcul de la vitesse d'un objet, vérification de la validité de la position d'un coéquipier, etc.). Elle propose

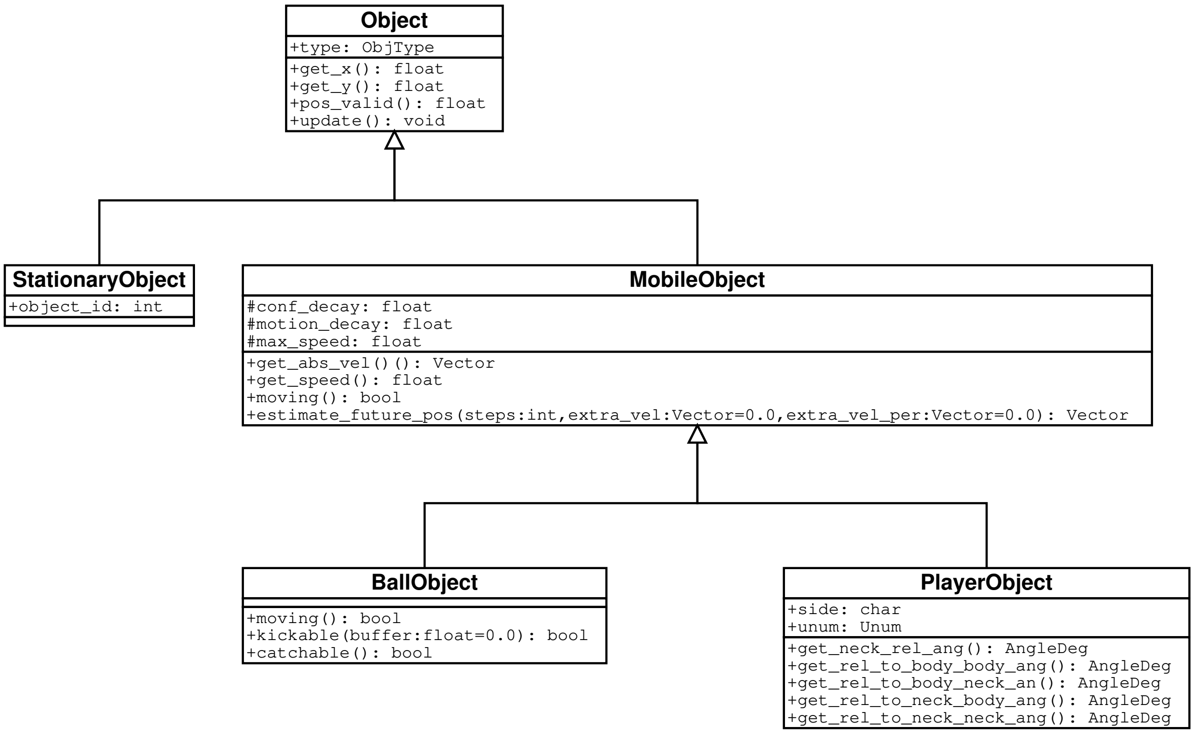

FIG. 4.2 – Hiérarchie des objets géométriques de CMU-2000

de plus des prédicats plus complexes, utilisables directement pour faciliter l'évaluation des actions en cours (par exemple le nombre de joueurs adverses dans un cône donné, ce qui est utile pour évaluer les chances de réussite d'une passe).

Notre implémentation utilise intensivement CMU-2000 pour gérer la couche protocole (attente de l'arrivée de messages, interprétation des messages reçus, mise à jour de l'état du monde en fonction de la réponse du serveur, etc.) et pour les primitives effectrices de bas niveau (avancée d'un pas, impression d'une accélération sur la balle, etc.). De même, nous utilisons certains facilitateurs de la classe *PositionInfo* pour simplifier des calculs géométriques et pour factoriser les calculs de position.

4.5 La planification

Nous définissons une architecture d'agents basée sur l'exécution de plans, conçus sous la forme de modules spécifiques. Leur exécution est autonome mais contrôlée

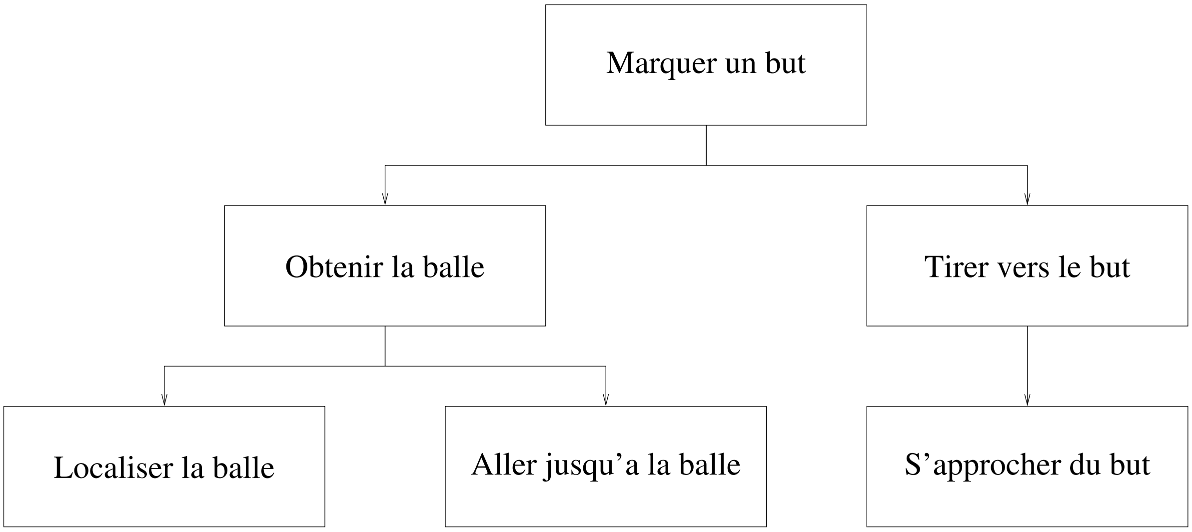

FIG. 4.3 – Exemple de plans hiérarchiques

par un ordonnanceur. Ces plans sont des objectifs à atteindre pouvant contenir à leur tour et de manière hiérarchique des sous-plans intermédiaires : nous représentons ainsi l'enchaînement des étapes nécessaires à l'exécution d'un plan (voir la figure 4.3, de la présente page). Les plans sont représentés par des objets de classes dérivant de la classe *Plan* (par exemple, *PlanTakeBall* ou *PlanWatch*), comportant toutes les données attachées à un plan, notamment les données nécessaires à la visualisation. Nos plans possèdent un mécanisme de contrôle d'exécution et sont capables de tolérer des états inattendus (pannes) ou de déclencher des actions intermédiaires pour arriver à leurs fins.

Nous considérons que les plans définis doivent prendre le plus d'initiatives qu'ils peuvent et qu'ils doivent contrôler le déroulement de leur action, car ils sont en effet les plus à même de le faire. Déléguer le contrôle de l'exécution d'une action à un autre organe équivaut à transporter une partie de la symbolique du plan vers ce dernier afin qu'il soit capable de modéliser et de comprendre le déroulement de cette action. De même, cela nécessite la mise en place d'un mécanisme de contrôle par un organe superviseur. La complexité induite est sensible aux pannes et alourdit le traitement du contrôle des actions entreprises alors qu'elle n'est pas nécessaire. Nous considérons les plans de notre architecture comme des effecteurs complexes et non

```
(if (equal? team-name "Sarcelles")
  (cond
   ((equal? my-number 1)
    (move -20 -15)
    (graphic-init)
    (add-plan "go-to-ball"))
   ((equal? my-number 2)
    (move -5 30)
    (graphic-init)
    (add-plan "find-ball")
    (add-plan "watch" 1))
   ((and
     (> my-number 2)
     (< my-number 11)) #t)))
```

FIG. 4.4 – Exemple de script d'initialisation

atomiques, leur exécution asynchrone étant très similaire à celle d'un effecteur atomique.

L'initialisation de l'agent et l'assignation d'un plan de démarrage est assuré par le biais d'un script Scheme. Nous avons défini une primitive Scheme (`add-plan`) permettant d'ajouter un plan dans les motivations d'un agent. Par exemple, « `(add-plan "find-ball")` » permet d'ajouter dans l'agenda la volonté de localiser la balle sur le terrain. À l'initialisation de l'agent, une option permet d'évaluer un script pouvant comporter des initialisations globales à l'équipe et des branchements conditionnels (« utiliser ce plan uniquement pour tel agent »). Ainsi, les procédures de tests sont facilitées par l'évaluation de scripts tels celui présenté dans l'exemple 4.4, de la présente page. Ce dernier permet de mettre en place deux joueurs, dont l'un tentera de s'emparer de la balle et dont l'autre observera le terrain et tentera d'analyser le plan du premier agent.

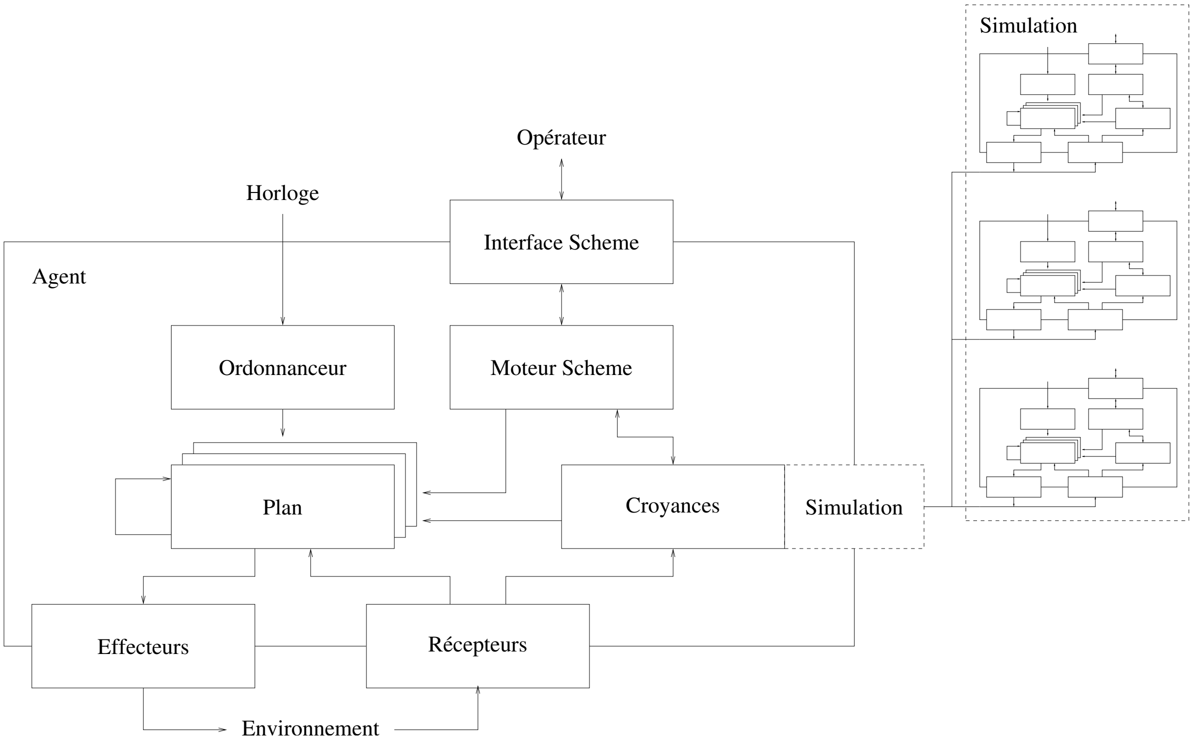

FIG. 4.5 – Proposition de modélisation d'agents simulés

Chacun des plans possède une clause d'invalidation qui est propre à une instance. Il s'agit d'une condition qui invalide le plan et le rend obsolète. L'obsolescence d'un plan peut être déclenchée par deux conditions : sa réussite (par exemple, atteindre la balle pour le plan *take-ball*) ou l'existence de conditions rendant impossible son exécution (par exemple, un autre joueur possède la balle pour le plan *take-ball*). Nous considérons en effet qu'un plan doit être rendu obsolète et abandonné dès qu'une clause rend son exécution difficile, quelle que soit la motivation existante à l'exécuter. Cependant, les plans doivent être responsables de la maintenance ou de la création des conditions nécessaires à leur bon fonctionnement : par exemple, le plan *take-ball* est responsable de la maintenance de la connaissance de la position de la balle en mémoire via l'exécution de plans intermédiaires. Les clauses d'invalidation sont des méthodes évaluées à chaque cycle avant l'exécution de chaque instance de plan.

Les plans développés contiennent de plus une clause de validité. Cette clause permet de déterminer si une instance de plan est valide pour un joueur ou non, en se basant sur son comportement apparent. Le mécanisme que nous proposons tente de vérifier de manière asynchrone et par hypothèse si une instance de plan correspond au comportement d'un joueur connu, c'est à dire si son état actuel correspond à une phase du plan (par exemple, si un joueur est le plus proche de la balle et qu'il est orienté vers elle, il peut être en train d'exécuter le plan *take-ball*). Ce mécanisme est imparfait en plusieurs points : il est sans état, c'est-à-dire qu'il ne tient pas compte des actions passées d'un joueur (ce qui interdit donc de dégager une intentionalité d'un processus) et de plus, il nécessite une heuristique distincte de celle de l'exécution du plan. Nous proposons une solution, qui serait d'abandonner la modélisation actuelle des autres agents, incluse actuellement dans la base des croyances et accessible par le biais de méthodes facilitatrices, au profit de la simulation de ces agents via une représentation mémoire similaire à celle de l'agent « réel » (l'agent exécuté par le processus et communiquant avec le serveur de la RoboCup) et une exécution en parallèle des modèles simulés. Voir par exemple la figure 4.5, page précédente, qui détaille le mécanisme de modélisation proposé. L'agent réel s'interfacerait avec les agents simulés par des canaux de communication semblables et un protocole identique à celui utilisé pour communiquer avec le serveur et se chargerait d'envoyer des informations sensorielles aux agents simulés en se basant sur ses croyances propres. Ainsi, l'agent réel simulerait par ce biais le comportement de ses partenaires en utilisant les heuristiques même de leur comportement réel. Les actions des agents simulés seraient perçues via leurs effecteurs et permettraient à l'agent réel de déterminer les actions probables de ses coéquipiers et adversaires en se basant sur les heuristiques et la représentation mêmes de leur exécution. La communication entre de tels agents n'étant pas soumise à des contraintes de bande passante, elle pourrait se faire de manière privilégiée et pourrait permettre de s'échanger des plans et d'estimer le comportement de ses coéquipiers (cette hypothèse assume cependant que la structure de tous les agents simulée est connue et reproductible).

Les plans actuellement implémentés sont orientés vers la manipulation et le maintien de la balle, mais l'implémentation de plusieurs plans sensitifs complexes simplifie le traitement des informations par l'agent. Les plans dont nous disposons sont pour le moment :

– *face_ball* : trouver la localisation de la balle sur le terrain et se tourner vers elle lorsqu'elle se déplace ;

– *find_ball* : trouver la localisation de la balle sur le terrain. Cela peut impliquer des mouvements pour l'obtenir dans son champ de vision ;

– *find_teammate* : trouver la localisation d'un de ses coéquipiers sur le terrain. Cela peut impliquer des mouvements pour l'obtenir dans son champ de vision. Une des ambiguïtés simulées par le serveur de la RoboCup est d'ailleurs l'incapacité de déterminer le numéro de maillot d'un coéquipier visible mais éloigné, ce qui peut amener à une situation de *dead-lock*. Il s'agit typiquement d'un cas où un plan sera le plus à même de contrôler l'échec de son action ;

– *pass_ball* : passer la balle à un partenaire. Cela implique de se rapprocher de la balle puis de la tourner en direction d'un partenaire et enfin de lui imprimer un mouvement afin de la déplacer jusqu'à ce dernier ;

– *take_ball* : aller à la position de la balle (plus précisément, s'en rapprocher jusqu'à ce qu'elle soit à portée). Si elle n'est pas dans le champ de vision, ce plan déclenche un plan *find_ball* pour la localiser ;

– *watch* : observer le déroulement du jeu de son point de vue (notamment, fixer la balle) et tenter de déterminer les plans des joueurs.

Chacune des instances de plan possède une représentation graphique propre, qui utilise la couche graphique développée pour notre implémentation (voir la section 4.6, page suivante pour plus de détails sur l'implémentation graphique). Les instances de plans utilisent une table de hachage pour stocker les objets graphiques utilisés et affichés à chaque cycle. À chaque exécution asynchrone du plan, les objets graphiques sont mis à jour par l'instance elle-même afin qu'ils soient les plus proches possible de la réalité et des symboles contenus dans l'état du monde.

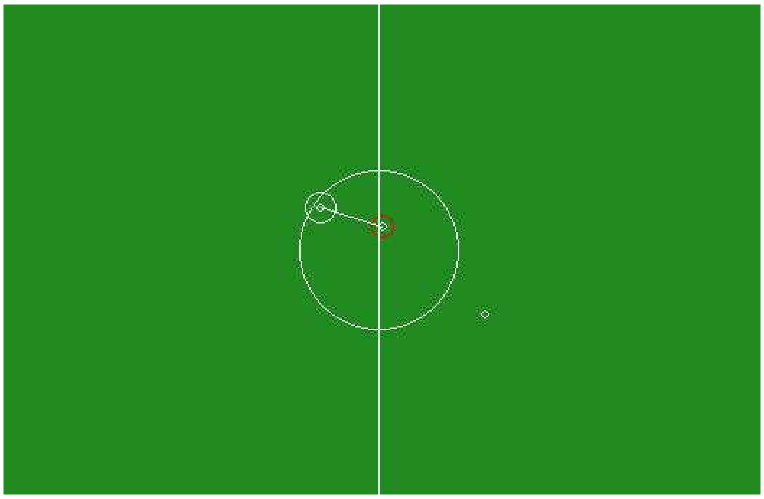

FIG. 4.6 – Visualisation des plans de l'agent

4.6 La visualisation

Nous avons été assez rapidement confrontés au problème de la visualisation des actions de nos agents : dans un milieu dynamique où les conditions de jeu sont rapidement changeantes, il est nécessaire à l'expérimentateur de posséder une bonne idée des buts de ses agents et des paramètres mis en œuvre dans les actions entreprises (notamment, les paramètres physiques de localisation). Typiquement, il est nécessaire de connaître le chemin qu'un agent désire emprunter pour se rendre à un point donné et la représentation textuelle seule est insuffisante pour ce type d'informations. Dans un univers discrétisé, la visualisation de telles informations est relativement aisée. Mais dans un environnement non discrétisé comme c'est le cas pour la simulation de la RoboCup, il est nécessaire d'opter pour une visualisation imparfaite (complétée au besoin par des informations textuelles) et spécifique ou alors de développer un mécanisme de visualisation vectorielle et symbolique (le symbolisme est notamment nécessaire pour garantir la généricité de la visualisation et le maintien d'une information dynamique). Nous avons opté pour l'implémentation d'un mécanisme de définition, de maintenance et d'affichage d'objets vectoriels, qui peuvent être initialisés et utilisés par l'un ou l'autre des composants de notre agent.

En raison de l'aspect dynamique et mobile des symboles à visualiser (joueur, balle,

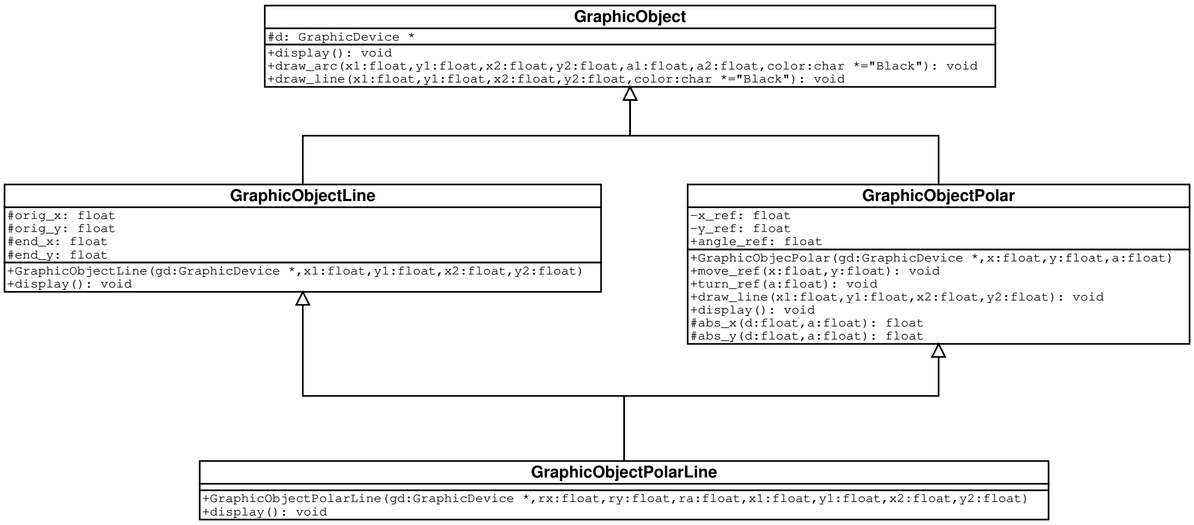

FIG. 4.7 – Exemple de hiérarchie d'objets graphiques

chemins, vecteurs, angle de vision, etc.), nous avons mis en place deux systèmes de coordonnées, le premier étant cartésien et absolu (l'origine étant le centre du terrain) et le deuxième étant polaire et relatif à une origine. Un objet graphique utilisant des coordonnées polaires doit hériter de la classe *PolarObject*, qui possède toutes les méthodes nécessaires à la maintenance d'une origine et à la transformation des coordonnées relatives polaires en coordonnées absolues cartésiennes.

Nous avons souhaité introduire une fonctionnalité supplémentaire à notre implémentation de la visualisation graphique : celle de l'indépendance du médium. Les mécanismes de visualisation se limitent souvent à un seul paradigme de représentation (écran d'ordinateur ou image *bitmap*, affichage vectoriel PostScript, sortie brute des coordonnées dans un fichier texte, etc.). Nous considérons ce mécanisme comme limitatif, notamment car la transformation d'un paradigme à l'autre est souvent destructive ou approximative. Ainsi, nous avons développé un mécanisme d'abstraction du support d'affichage nous permettant :

1. de définir des primitives géométriques de bas niveau, indépendantes du support d'affichage ;

2. de définir des interfaces entre le support d'affichage (ou *périphérique*) et les primitives graphiques de bas niveau : nous avons défini des correspondances entre les primitives d'affichage bas niveau et les primitives d'affichages spécifiques, mais aussi des mécanismes d'initialisation du support graphique, d'allocation et de cache de couleurs ;

3. de définir des objets graphiques de haut niveau, symboliques et faisant appel aux primitives graphiques de bas niveau pour s'afficher sur le support graphique.

Nous avons déjà implémenté un périphérique d'affichage basé sur les primitives graphiques de la bibliothèque *Xlib*, qui permet de visualiser les objets graphiques dans une fenêtre X11. Nous envisageons le développement d'un périphérique PostScript (dont l'utilité prend tout son sens dans la production de documents écrits), d'un périphérique MPEG (par exemple en utilisant la bibliothèque *libfame*, disponible sur le web à l'URL `http://fame.sourceforge.net/libfame/overview/`), afin de produire de manière native des séquences animées (fichiers vidéo) et d'un périphérique OpenGL, permettant une visualisation en trois dimensions.

Chapitre 5

À la base du jeu d'équipe, l'individu

La polysémie du terme *agent* induit une complexité sémantique dans la description d'architectures ou de structures d'agents. Ce chapitre commence par définir le mot agent dans ses significations usuelles, puis il met en place une taxinomie des différents types d'agents existants, il introduit ensuite quelques exemples de modèles d'agents et enfin il détaille les composants les plus usuels au sein d'une architecture individuelle.

5.1 Un agent vu comme processus

Pour Marvin MINSKY, un agent est un *processus* pouvant effectuer des « tâches simples ne demandant ni esprit ni réflexion ». De nombreux modèles tentent de donner une définition plus précise du sens exact du mot agent. Le consensus entre ces différentes définitions est résumé dans [12] : pour Stan FRANKLIN, un agent est un objet qui *agit* en plus de raisonner, ce qui est un apport sur les systèmes d'intelligence artificielle classiques.

Le mot *agir* possède quatre sens communs applicables à la définition que nous essayons de donner d'un agent :

– agir, c'est accomplir une action, faire quelque chose (« il est temps d'agir »). Par exemple, se déplacer dans son milieu ou transformer un objet ;

– agir, c'est se conduire d'une certaine façon (« agir en sage »). Par exemple, adopter une attitude de coopération ;

– agir, c'est opérer un effet (« agir sur le moral du groupe »). Par exemple bloquer le passage d'un agent ou rendre possible une action ;

– agir, c'est intervenir auprès d'une personne (« agir auprès d'un supérieur »). Par exemple, demander des ressources à un agent.

Nous définissons un agent comme un système autonome, composé d'organes distincts et possédant des ressources propres. Un agent est situé dans son environnement et son comportement tend vers l'accomplissement conscient ou émergent d'un but. Il peut sentir son environnement à travers ses récepteurs et y agir en utilisant ses effecteurs [17].

Le dictionnaire définit un *processus* comme un « développement temporel de phénomènes marquant chacun une étape. » Cette définition convient à notre propre définition d'un agent. Nous considérons qu'un agent effectue des actions dans le temps et sur son environnement et qu'il subit de plus des phénomènes internes propres à son architecture. L'architecture d'un agent décrit ses mécanismes internes de traitement des informations et d'interaction avec son environnement.

La technologie agent permet dors et déjà de résoudre de manière opérationnelle des problèmes complexes. Les applications des systèmes à base d'agents sont multiples. Par exemple :

– la résolution collaborative de problèmes et l'intelligence artificielle distribuée [6, 9]. L'intelligence artificielle distribuée est centrée sur les macro-phénomènes d'un système distribué (interactions) et non sur les micro-phénomènes d'un tel système (architecture d'un agent), cependant les applications d'intelligence artificielle distribuée se basent sur la technologie agent. C'est notamment le cas des applications de contrôles de flux (surveillance de trafic aérien [24], résolution de contraintes horaires, suivi médical, etc.) ;

– les agents d'interface [19]. Il s'agit de programmes qui coopèrent avec l'opérateur dans son utilisation d'une application informatique. Leur interaction avec l'utilisateur (ainsi qu'avec d'autres agents) leur permet de formuler des plans dont la finalité est d'assister l'utilisateur dans son travail. Ces agents peuvent par exemple se charger de filtrer les informations qui pourraient être utiles à l'utilisateur ;

– les agents d'information [5]. Ces agents ont accès à une ou plusieurs sources d'information, potentiellement distribuées et hétérogènes. Ils sont chargés de répondre aux questions d'un utilisateur. Il peut s'agir d'agents chargés de parcourir le web à la recherche d'informations sur un sujet précis ou d'agents collectant et agrégeant des informations provenant de bases de données hétérogènes ;

– les agents « réalistes ». L'application de tels agents est directe dans le domaine des jeux vidéo ou de la réalité virtuelle. L'inclusion de personnages réalistes dans un monde simulé implique de leur insuffler un comportement proche de celui des êtres réels, notamment par la simulation de « sentiments » et de buts. L'abstraction agent et la définition de plans propres le permet de manière efficace.

5.2 Les architectures individuelles

Nous définissons un agent comme un système autonome, composé d'organes distincts (opérationnels ou cognitifs par exemple) et possédant des ressources propres. Définir la notion d'agent nécessite de se pencher sur les architectures existantes agençant ces organes tout comme de détailler la nature de chacun de ces organes. Cette section commence donc par passer en revue différentes architectures individuelles d'agents (nous appelons « architecture individuelle » un modèle utilisé pour décrire le fonctionnement interne d'un agent, par opposition aux architectures sociales décrivant les macro-phénomènes d'un système d'agents et les interactions entre les membres le composant), puis elle décrit les dispositifs composant un agent. Enfin, nous concluons en proposant une classification d'architectures d'agents.

5.2.1 Quelques exemples d'architectures individuelles

Malgré une évolution continue et un progrès récent considérable, les architectures d'agents utilisées actuellement sont pour la plupart basées sur des modèles anciens et elles dérivent de quelques familles de modèles largement décrites. Nous allons détailler l'aspect fonctionnel des architectures d'agents les plus utilisées dans la littérature :

- l'architecture modulaire horizontale. Cette architecture consiste en un assemblage de modules ayant chacun la charge d'une fonction particulière (par exemple la perception, la communication, la maintenance de la base des croyances, la gestion des engagements, la gestion des buts, la prise de décision, la planification, etc.). voir par exemple la figure 5.1, page suivante. Les canaux de communication entre les composants de cette architecture sont statiques et déterminés par l'architecte. Un des avantages de cette architecture est qu'elle est très simple à mettre en œuvre. L'architecture horizontale est celle pour laquelle nous avons opté dans la conception et l'implémentation de nos agents ;

- l'architecture du tableau noir. Cette architecture comporte trois sous-systèmes, qui sont les sources de connaissances (ou *knowledge sources* ou *KS*), le contrôleur (qui est chargé de gérer les conflits entre les sources de connaissance) et le tableau noir en question (le tableau noir est une base de données partagée des états partiels de l'agent, de ses hypothèses, des résultats intermédiaires, etc.). Les sources de connaissances sont dites *opportunistes*, car chacune d'entre elles décide de s'activer lorsqu'elle le juge pertinent, en fonction de l'état du tableau noir. Ces trois sous-systèmes ne communiquent pas directement mais partagent cependant des informations, ce qui leur permet de s'agencer convenablement. Un exemple d'architecture à base de tableau noir est décrit dans [4] ;

- l'architecture de la subsomption. Cette architecture est organisée de manière verticale. Contrairement à l'architecture horizontale, les modules composant une

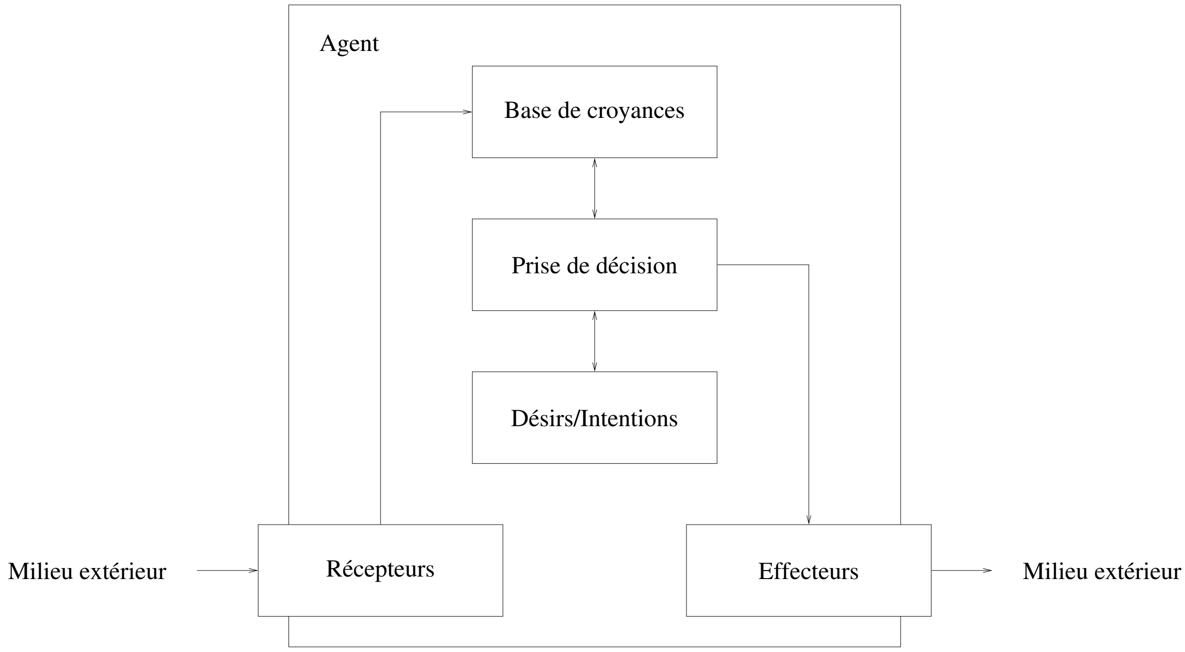

FIG. 5.1 – Un exemple d'architecture d'agent horizontal

architecture à base de subsomption sont des tâches complètes (comportant typiquement une base de croyances, un planificateur, etc.). Les conflits entre les tâches existantes d'un agent subsomptif sont gérés par des règles de priorité et de domination d'une tâche sur une autre ;

– l'architecture des tâches compétitives. Cette architecture est très semblable à la précédente, mais un module à part (*sélecteur de tâches*) se charge de choisir les tâches à exécuter à un moment donné, en fonction de l'état actuel de l'agent et de son environnement (les critères de choix de la tâche prioritaire peuvent se baser sur son adéquation avec les motivations de l'agent) ;

– l'architecture du système de production. Cette architecture repose sur une base de règles, dont la validité est vérifiée régulièrement. Lorsque les conditions qui valident une règle sont réunies, celle-ci s'active et exécute une action donnée (ce système est très semblable à un *moteur d'inférences*) ;

– l'architecture du classifieur. Cette architecture est très semblable à la précédente mais apporte une dimension supplémentaire, celle de l'évolutionnisme. Lorsqu'une règle s'active, son effet est évalué par le classifieur et son adéquation (*fitness*) avec le problème en cours est enregistrée : une règle efficace sera mieux notée qu'une règle peu efficace. Par la suite, les règles les mieux notées auront plus de chance d'être activées et auront plus de poids dans l'évolution de l'agent ;

– les architectures connexionistes. Ces architectures sont basées sur l'utilisation d'un modèle proche des approximations actuelles de l'architecture du cerveau [16]. Les récepteurs et les effecteurs sont reliés par un réseau de neurones artificielles, s'activant chacune lorsque la somme de la stimulation de ses entrées est suffisante. L'approche connexioniste permet de plus la mise en place d'un mécanisme d'apprentissage par propagation ;

– les architectures dynamiques. Dans ces architectures, la liaison entre les récepteurs et les effecteurs est assurée par le biais d'équations logiques. Cette architecture est typiquement utilisée en robotique pour contrôler l'inclinaison des roues et la vitesse d'avancée en fonction de l'état de capteurs.

5.2.2 Les composants d'un agent

Un grand nombre d'architectures d'agents sont organisées de manière modulaire [1, 19, 25, 35]. Ces modules se retrouvent sous des formes proches dans la plupart des architectures d'agents (typiquement, toutes les architectures d'agents comportent les trois volets perception, délibération et action). En revanche, leur organisation au sein de l'architecture d'un agent et les communications et interdépendances entre modules sont différentes d'une architecture à l'autre. Nous allons détailler les différents composants habituellement trouvés dans les architectures d'agents décrites dans la littérature.

On trouve dans pratiquement toutes les architectures d'agents un organe preneur de décisions (ordonnanceur), assimilé par analogie à l'unité de contrôle de l'ordinateur de Von Neumann[1]. La plupart des agents ne peuvent structurellement exécuter qu'un nombre limité d'actions atomiques simultanément (généralement une seule en raison de limitations physiques ou de limitations des langages de programmation ou de systèmes d'exploitation utilisés, tous n'offrant pas facilement de fonctionnalités de parallélisme préemptif ou d'exécution asynchrone). Il est donc nécessaire de choisir parmi les actions à effectuer celle qui pourra s'exécuter en se basant sur des critères tels que l'adéquation aux objectifs, la priorité (pondération), la complexité, la durée, etc. Le choix d'une tâche à exécuter peut être influencé par les autres organes de l'agent. Typiquement, l'ordonnanceur peut faire appel à l'expertise d'autres organes pour évaluer la pertinence d'une tâche.

Les agents de type intentionnel ayant un but à accomplir, il leur est nécessaire de connaître les actions à effectuer pour l'atteindre, c'est-à-dire de disposer d'objectifs intermédiaires à accomplir. De tels agents ont donc besoin de conserver en mémoire une image de leurs objectifs ou au contraire dont la validité n'est pas possible dans l'état actuel du monde. En effet, les cogniticiens considèrent généralement que la décision d'adopter d'une attitude est le résultat d'un calcul basé sur la multiplication de l'importance de cette attitude (selon les critères des motivations de l'agent) par l'estimation des espérances de succès. Une telle base d'objectifs doit être mise à jour de manière régulière pour correspondre à la réalité. Par exemple, une mise à jour est nécessaire à chaque accomplissement d'un objectif intermédiaire ou lorsque la dérivation de la perception du monde dépasse un certain seuil par rapport à l'état du monde lors de l'établissement des objectifs.

[1] L'ordinateur de Von Neumann est l'architecture la plus couramment utilisée pour décrire le fonctionnement d'un ordinateur moderne. Certains y voient des similitudes avec l'architecture d'un agent. L'architecture de Von Neumann est formée de cinq composants qui interagissent les uns avec les autres :

– l'unité arithmétique et logique (UAL) ou unité de calcul ;
– l'unité de contrôle ou unité de commande ;
– la mémoire centrale ou mémoire principale ;
– les unités d'entrée ;
– les unités de sortie.

Afin de connaître la pertinence d'un objectif, c'est à dire de vérifier s'il est toujours valable en fonction de l'état du monde extérieur ou de l'état interne de l'agent lui-même, un agent intentionnel a besoin de disposer d'une vision (ou modèle) symbolique du monde. Ce modèle est une représentation interprétative du monde obtenue à partir d'informations sensorielles ou de croyances structurelles. Il prend la forme d'une base de données de symboles représentant chacun un aspect de l'environnement (par exemple, des symboles décrivant la température, les coéquipiers, la topologie du terrain, etc.) ou des relations entre symboles. Cette représentation étant interprétative, elle peut dériver fortement de la réalité fonctionnelle extérieure. C'est pour cette raison qu'elle est aussi parfois appelée *base de croyances* (nous considérons ici l'adéquation physique et non opérationnelle avec la réalité). De plus, dans le cas particulier d'un milieu dynamique (voir le chapitre 2, page 11), cette représentation du monde doit être mise à jour de manière régulière, voire en temps réel, afin d'être la plus consistante et pertinente possible.

Le planificateur est un organe qui met en place une succession d'opérations, dans le but de remplir les objectifs, en se basant sur l'hypothèse que les actions planifiées vont modifier l'environnement dans un sens positif. Le planificateur intervient à deux niveaux dans le comportement d'un agent : à un niveau supérieur, il détermine quels sont les comportements à suivre sur le long terme pour adopter une stratégie efficace et à un niveau inférieur, il détermine les primitives d'action (*actions atomiques*) nécessaires à l'accomplissement de la prochaine étape planifiée importante. Dans le cas d'un système multi-agents, l'activité planifiée coordonnée nécessite de plus une synchronisation entre les différents agents impliqués et un partage de plans afin d'obtenir un ordonnancement efficace entre leurs actions en vue d'une résolution globale de problèmes [9].

Un agent a besoin de connaître avec une précision suffisante l'état de son environnement. Pour posséder un état du monde adéquat, il doit le *percevoir* directement ou doit obtenir des informations sur celui-ci par le biais d'autres agents (des agents facilitateurs par exemple). Dans les deux cas, il a besoin d'organes sensitifs ou *percepteurs*, qui lui envoient des informations sur son environnement, en continu ou sur des bases événementielles. Les récepteurs d'un agent sont typiquement des capteurs dans le cas d'un agent robotique ou des canaux de communication avec son environnement informatique dans le cas d'un agent purement logiciel (cela peut inclure des canaux de communication avec d'autres agents ou des dispositifs de mesure). L'action des percepteurs est renforcée par un travail de reconstruction d'informations à un niveau symbolique ou logique.

De même, un agent doit avoir le pouvoir d'agir sur son environnement afin de résoudre le problème pour lequel il a été créé ou afin de maintenir un équilibre ou un état en fonction de la représentation qu'il se fait de son environnement. Par exemple, il peut s'agir d'assurer sa propre survie ou de maintenir un facteur au dessus d'un certain seuil. Les actions portant sur l'environnement de l'agent sont effectuées par ses effecteurs, qui consistent en des dispositifs mécaniques dans le cas d'un agent robotique (par exemple des roues, un bras articulé, un émetteur radio ou lumineux, etc.) ou en des méthodes logicielles (canal de communication avec un processus, appel système, etc.). Le bon fonctionnement des plans de l'agent nécessite d'obtenir des informations en retour de son action afin d'entreprendre une correction en cas d'échec (par exemple, nos agents évaluent à chaque cycle l'adéquation de la situation par rapport à l'action suivie). Certains effecteurs permettent l'évaluation de l'action entreprise (mesure de la vitesse d'un moteur, code de retour d'une procédure, etc.) ce qui permet d'obtenir immédiatement une mesure de l'incidence de l'action sur l'environnement ou sur son issue. Dans le cas contraire, l'évaluation de l'action d'un effecteur doit se faire par l'étude de l'environnement et de son évolution.

5.3 Une classification

La classification des différentes architectures d'agents, basée sur des critères pertinents et sur un mode naturaliste (*taxinomie*) est un mécanisme nécessaire à la compréhension des architectures existantes. Nous proposons ainsi dans cette section une taxinomie des agents logiciels et définissons des critères de classification non exclusifs basés sur des pistes proposées par [12]. Le choix de critères de classification d'une taxinomie est arbitraire et repose sur un besoin de nommer précisément les architectures existantes en fonction de leur aspect opérationnel ou architectural.

5.3.1 Les agents réactifs

Un agent est réactif s'il répond de manière opportune aux changements de son environnement. Ceux-ci peuvent être constitués de stimuli externes et asynchrones. Un tel type d'agent n'a pas besoin d'une représentation symbolique élevée de la perception son environnement. Pour [3], une telle définition s'applique tout aussi à un thermostat, qui se contente purement d'agir en fonction des stimuli perçus, sans avoir de compréhension de son univers ni de ses buts. On trouve de nombreux agents réactifs dans la littérature, mais peu le sont purement. Pour la plupart, la réactivité n'est qu'une caractéristique et n'interdit pas l'action intentionnelle.

La structure des agents purement réactifs tend à la simplicité, mais ces derniers peuvent être capables d'actions de groupe complexes et coordonnées. C'est le cas d'une société de fourmis, dont la somme des membres est capable d'actions évoluées mais dont chaque individu pris séparément possède une représentation faible de l'environnement et n'a pas de buts globaux. Les activités complexes et tournées vers un but sont dans cette architecture d'agents une propriété émergente des interactions sociales entre les agents constituant le groupe.

On trouve un exemple de définition d'une société d'agents réactifs dans [8]. Alexis DROGOUL y décrit notamment le mécanisme de la *sociogénèse* de colonies de fourmis à travers une approche multi-agents. Une simulation (*MANTA*) met en évidence l'émergence de stratégies à long terme complexes au sein d'une société d'agents pourtant non cognitifs. Cette émergence est aussi mis en exergue dans une simulation de jeu d'échecs, où l'approche distribuée purement réactive produit cependant un comportement consistant sur le long terme.

5.3.2 Les agents intentionnels

Un agent est intentionnel, dirigé par des buts ou encore *cognitif* s'il ne se contente pas d'agir en fonction des conditions de son environnement mais en fonction de ses buts propres et s'il possède une intentionnalité, c'est-à-dire est une volonté consciente d'effectuer un acte. Un tel agent doit posséder un état mental qui l'incite à agir dans un sens particulier et donc une représentation symbolique, de plus ou moins haut niveau en fonction de la complexité de la tâche à accomplir, de son environnement et de ses buts. Cette représentation symbolique lui permet de prendre conscience de l'état d'accomplissement de ses buts et des contraintes restant à soulever, mais aussi de ses motivations propres.

Un agent intentionnel est opposé à un agent purement réactif en ce que la perception de son environnement est symbolique et son comportement conditionné par ses motivations. Cependant, les deux notions ne sont pas antagonistes. Par exemple, un agent intentionnel peut tout à fait réagir à des stimuli de son environnement tout en conservant une haute représentation de celui-ci et en agissant en fonction de ses intentions. Au sein de l'architecture des agents intentionnels, on sépare les agents dont l'intentionnalité est immédiate de ceux dont l'intentionnalité est dirigée vers le futur. Dans le premier cas, l'intentionnalité représente un mécanisme d'exécution des actes de l'agent, alors que dans le deuxième cas, l'intentionnalité fait intervenir une planification des actes de l'agent.

Un modèle d'agent intentionnel est par exemple décrit dans [24] : le modèle *BDI* (pour *Beliefs Desires Intentions*). Ce modèle définit trois composantes : les croyances, les désirs et les intentions. Un agent BDI modélise dans un arbre les mondes possibles selon ses croyances et adapte les mondes possibles selon ses désirs ainsi que les mondes possibles selon ses intentions en fonction des futurs probables. Ces mondes sont modélisés sous forme hiérarchique en fonction d'hypothèses formulées sur le futur de l'environnement. Plusieurs variantes de ce modèle sont proposées, parmi celles-ci : les agents réalistes, qui désirent des propositions mais peuvent les considérer comme irréalisables et les agents non réalistes (opiniâtres), qui refusent de considérer leurs désirs comme irréalisables, tout comme les informations déniant la validité de leurs désirs.

5.3.3 Les agents autonomes

Un agent autonome est un agent qui agit dans et sur son environnement et qui ne se contente pas d'être dirigé par des commandes venant d'un autre agent ou de l'opérateur. Un tel agent doit posséder des effecteurs et être situé dans son environnement. L'autonomie d'un agent ne porte pas seulement sur son comportement mais aussi sur ses ressources internes qui sont elles aussi autonomes : d'un point de vue général et dans le cas d'un agent autonome situé, on définit un agent comme « tout ce qui ne fait pas partie de l'environnement. »

Pour Jose BRUSTOLONI [3], un agent autonome doit savoir comment accomplir ses buts en fonction de l'état de son environnement ou tout du moins, doit être capable de trouver un moyen pour accomplir ses buts. Ces capacités de connaissance ou de recherche impliquent un traitement structurel (c'est le cas des architectures connectivistes) ou symbolique (où l'agent manipule des symboles et détermine son comportement en fonction de ces manipulations) des stimuli reçus.

Nous considérons cependant que l'autonomie est un concept distinct de l'intentionnalité. L'autonomie est une conséquence de l'hermétisme de l'agent à son environnement et de sa prise de décision indépendante de commandes externes. De même, un comportement cohérent et d'aspect extérieur réfléchi peut se passer de cognition. Ainsi, on trouve par exemple des agents autonomes mais purement réactifs dans [8].

5.3.4 Les agents adaptatifs

Un agent adaptatif est capable d'apprendre en fonction de son expérience passée et de son évolution. Les architectures d'agents produisant des unités fonctionnelles dédiées à des tâches précises, les techniques d'apprentissage classique de l'intelligence artificielle s'appliquent particulièrement bien à l'apprentissage d'un agent (qu'il s'agisse de faire évoluer ses capacités d'action, de décision ou d'analyse) [31].

En particulier, plusieurs recherches [2, 31] appliquent les techniques de l'évolution génétique aux effecteurs de bas niveau des robots footballeurs participant à la RoboCup. Les algorithmes génétiques sont aussi utilisés comme support à la décision par [18], qui les applique dans une simulation de marché, contenant des interactions entre demandeurs et fournisseurs. De plus, le paradigme des systèmes multi-agents peut être mis en parallèle avec la notion de population et de génotypes des algorithmes génétiques : par exemple, chaque agent possède un génotype et un agent particulier du système se charge de les évaluer et de favoriser la reproduction des agents adaptés au problème.

5.3.5 Les agents mobiles

Un agent mobile est un agent capable de se déplacer dans son environnement, qui peut être physique (réel ou simulé) ou structurel (niveaux d'exécution par exemple). Un agent mobile dispose donc de dispositifs assurant sa mobilité. Dans le cas d'un agent robotique physique ou simulé, il s'agit bien évidemment d'effecteurs lui permettant de se déplacer. Dans le cas d'un agent logiciel, la mobilité implique le déplacement dans le réseau ou dans l'architecture du système, c'est à dire la mise en place d'un mécanisme de migration de processus [5].

Un agent robotique mobile peut se déplacer vers les ressources dont il a besoin, se rapprocher d'un objectif à accomplir, fuir des prédateurs, etc. Un agent logiciel mobile n'est bien entendu pas concerné par la problématique de la localisation physique, mais en revanche, il peut tirer partie de la topologie du réseau informatique. Ainsi, un agent mobile qui se rapproche d'une ressource (base de données, agent logiciel, périphérique, etc.) pourra travailler de manière plus efficace : plutôt que d'accéder à cette ressources travers un WAN[2], il peut se déplacer à travers le réseau de relais en relais et accéder localement à la ressource désirée en utilisant une bande passante maximale.

Les agents d'information (voir la section 5.1, page 47) sont la plupart du temps des agents mobiles : leur rôle est de parcourir des sources d'information souvent distribuées et donc d'être amenés à se déplacer dans un réseau. Des agents d'information mobiles permettent à l'utilisateur de s'abstraire de la nature des sources d'information, à condition que l'agent mobile assure un rôle de passerelle et soit pourvu d'une interface avec ces sources d'information [5]. De même, grâce à ses ressources et à ses buts propres, un agent mobile peut effectuer sa recherche d'information de manière

[2]WAN : *Wide Area Network*, il s'agit d'un réseau étendu physiquement, typiquement l'Internet. Un WAN a généralement une bande passante limitée.

asynchrone sans que l'utilisateur n'ait à le manipuler.

5.3.6 Les agents flexibles

Les agents flexibles sont des agents capables de modifier leur structure fonctionnelle (code) en cours d'utilisation. Par exemple, les agents flexibles peuvent être capables de charger dynamiquement un script[3] de contrôle envoyé par l'utilisateur. L'exécution de tels scripts nécessite une interface de chargement (typiquement, un service réseau) ainsi qu'un mécanisme d'exécution des scripts reçus : interface avec les données, opérateurs d'exécution, etc. On trouve ainsi par exemple dans [6] des agents flexibles utilisés pour surveiller l'état de ressources distribuées. Nous avons implémenté un mécanisme similaire, basé sur l'utilisation du langage *scheme*. Ce mécanisme est décrit dans la section 4.1, page 31.

5.3.7 Les agents sociaux

Un agent social est un agent qui opère des interactions avec d'autres agents au sein de son environnement. Ces interactions peuvent faciliter la tâche d'un agent (coopération) ou au contraire le gêner dans l'accomplissement de ses buts (encombrement, compétition). Un système possédant plusieurs agents est appelé un *système multi-agents* ou *SMA*. Nous appelons un système possédant plusieurs agents sociaux un *système multi-agents sociaux*. Se reporter à la section 6.1 page 61 pour une description plus complète des systèmes multi-agents.

Les agents avec lesquels un agent social est en mesure d'avoir des interactions peuvent être *hétérogènes*, c'est-à-dire qu'ils peuvent avoir des formes, des interfaces, des buts ou des représentations de l'environnement différentes. La communication est une forme d'interaction nécessitant un protocole et un langage commun aux différentes parties et indépendants des spécificités de ces dernières. De plus, un canal de communication adapté à tous les agents l'utilisant et permettant de transporter les messages échangés est requis [36].

[3]Un script est une suite d'opérations décrites dans un langage textuel adapté à leur sémantique opératoire.

Les interactions entre agents sont possibles sur une base bipolaire (*micro-interactions*) ou sur une base faisant intervenir un grand nombre d'agents : c'est le cas des messages émis par un agent à destination de tous les agents de son environnement (*macro-interactions*). Voir la section 6.1 page 61.

Les interactions et, plus spécifiquement, la communication sont des composantes nécessaires au travail coordonné d'une équipe d'agents. Nous décrivons dans le chapitre 6, page suivante, les fondements du jeu d'équipe et la nature des interactions intervenant au sein d'une d'équipe d'agents.

Chapitre 6

Décisions et plans coopératifs dans le jeu d'équipe

Les systèmes multi-agents sont une abstraction issue directement de l'intelligence artificielle distribuée bien que les systèmes multi-agents se concentrent d'avantage sur les phénomènes internes des individus les composant. Ainsi, un système multi-agents est composé d'un ensemble d'individus autonomes et distincts, ayant chacun leurs propres buts et leurs propres motivations et dont l'action produit un comportement global émergent. C'est notamment le cas des collectivités d'insectes sociaux qui sont considérés comme un exemple de systèmes multi-agents. Les individus les composant n'ont qu'une expertise et des motivations locales, mais leurs comportements sont dirigés inconsciemment vers la résolution d'une partie d'un but complexe : la survie de la colonie [8]. Les systèmes multi-agents sont apparus au début des années 1980 [7] et se sont énormément développés pendant les années 1990. Aujourd'hui, des applications industrielles telles que le contrôle des vols dans les aéroports ou la simulation militaire [33] mettent déjà en œuvre des systèmes multi-agents et le domaine du jeu les utilise intensivement.

Une équipe est définie par le dictionnaire comme un « groupe de personnes collaborant à un même travail. » Dans le cas de systèmes multi-agents, cette collaboration fait intervenir plusieurs agents sociaux, qui non seulement coopèrent et se coordonnent, mais axent aussi leurs efforts coopératifs vers l'atteinte d'un but commun. Cette coopération implique des échanges d'informations, des partages de plans et de buts, mais aussi des mécanismes de synchronisation sociale et de résolution de pannes. Nous allons décrire dans ce chapitre les différents composants nécessaires à une activité coordonnée entre agents sociaux coopératifs, puis nous allons présenter les aspects sociaux propres au jeu d'équipe.

6.1 L'individu vu comme un membre d'une architecture sociale

Un système multi-agents coopératif (ou *SMA* coopératif) est un système qui dispose :

- d'agents autonomes sociaux, qui peuvent être homogènes ou hétérogènes, mais fonctionnant en parallèle, de manière synchrone ou asynchrone ;
- d'un mécanisme d'interactions entre les différents agents composant le système.

Nous divisons de manière logique l'organisation des systèmes multi-agents coopératifs en sociétés non cloisonnées, dont la nature dépend de la somme des agents qui y sont situés : des micro-sociétés (regroupement de deux agents) ou des macro-sociétés (amalgame important d'agents ou de micro-sociétés). Cette séparation est architecturelle et fractale : ce que nous appelons macro-société est un agencement structurel d'entités (ressources) distinctes, qu'il s'agisse d'agents ou de sociétés d'agents.

Une des propriétés émergentes des micro-sociétés est de produire des fonctionnalités (actions de groupe, perceptions locales du problème, etc.) qu'une macro-société peut utiliser pour résoudre un problème global : les agents d'un système se regroupent dans le contexte d'un but plus global poursuivi par « l'entité » groupe. En revanche, les macro-sociétés produisent des contraintes (explicitement décrites ou émergentes elles aussi) à destination des agents et des micro-sociétés qui les composent (contraintes des coordination, conflits, etc.). Ces contraintes tendent à encadrer les agents vers la résolution du problème global et à utiliser leurs capacités de manière coordonnée avec les autres agents du groupe. De telles contraintes décrivent la structure organisationnelle d'une macro-société d'agents.

Les agents sociaux composant un système multi-agents sont en interaction régulière, mais cette interaction peut se révéler conflictuelle : les buts et les moyens des agents peuvent être différents et antagonistes. Ainsi, on dégagera plusieurs formes d'interactions au sein d'un système multi-agents : la collaboration (addition simple des compétences des agents), la coopération (collaboration coordonnée), la compétition (les agents ont des buts antagonistes ou doivent se partager trop peu de ressources), encombrement (les agents se gênent les uns les autres, mais ne sont pas en compétition).

Sous réserve de résolution des coûts supplémentaires induits par la coopération (synchronisation, partage, etc.), la coopération est une attitude que les agents ont tout intérêt à adopter pour plusieurs raisons : les problèmes peuvent être trop complexes ou distribués pour être résolus localement, certaines actions peuvent être trop coûteuses ou trop importantes pour les ressources d'un seul agent, certains agents sont potentiellement plus spécialisés que d'autres pour résoudre un problème, etc. Pour conserver une motivation suffisante à la coopération, les systèmes multi-agents doivent donc garantir un bénéfice supérieur à l'investissement pour les agents qui coopèrent (la théorie de jeux prouve par ailleurs que le bénéfice de la coopération est d'autant plus important que le nombre de participants coopérant est important). Un des points clefs de la recherche en intelligence artificielle distribuée est le nécessaire besoin de trouver des organisations sociales dont l'appartenance impacte le moins possible sur l'accomplissement des buts individuels des agents les composant.

6.2 La coopération comme base du fonctionnement d'un système multi-agents

Si un mécanisme efficace de gestion des conflits est mis en place, les agents composant un système multi-agents ont une motivation très forte pour coopérer entre eux et tenter de résoudre un problème collectivement. Nous allons identifier différents types de coopération tels qu'on les trouve dans la littérature [10] :

- le regroupement et la multiplication. Les agents forment un bloc composé de la totalité des individus, ce qui leur permet d'effectuer des actions impossibles à un agent seul (amalgame) ;

- la communication. Les agents communiquent entre eux par l'envoi de messages. La définition d'un protocole de communication est nécessaire pour que les agents puissent se comprendre et échanger des informations ;

- la spécialisation. Chaque agent du groupe se charge d'une action particulière et acquiert des compétences particulières et utiles au comportement global du système ;

- le partage de tâches et de ressources. Les agents se partagent les tâches à accomplir par accointances (connaissances mutuelles des capacités) ou par un mécanisme de marché et d'appel d'offres ;

- la coordination. Les agents définissent l'ordre de démarrage des actions à accomplir et se synchronisent ;

- l'arbitrage. Les agents résolvent les conflits qui peuvent survenir entre eux. Cette coordination peut faire intervenir une coordination volontaire ou un mécanisme explicite de lois.

6.2.1 La coopération et la communication

Un individu qui ne communique pas est un individu isolé, muet et sourd aux informations potentiellement utiles en provenance des autres agents du système. Un tel individu est inapte à se synchroniser avec ses pairs et donc à former une équipe pour travailler en groupe. La coopération efficace implique que les agents engagés dans une tâche commune communiquent entre eux.

Communiquer, c'est avant tout émettre des signaux. La communication entre plusieurs agents nécessite un mécanisme de transport de l'information ainsi qu'un support sur lequel l'agent émetteur peut écrire et sur lequel un ou plusieurs agents peuvent recevoir des informations. On parle alors de support ou de canal de communication. La définition de canaux de communication permet d'abstraire les paramètres physiques ou fonctionnels d'un médium de transport d'information. On trouve par exemple un tel mécanisme d'abstraction dans [36].

Nous distinguons trois grands types de canaux de communication transportant un signal :

– les canaux de type direct (synchrones ou asynchrones), où l'agent émetteur initie une communication directe entre lui et les destinataires (c'est le cas du téléphone ou du courrier postal) ;

– les canaux propagatoires, où le signal est diffusé dans l'environnement et dont l'intensité décroît avec la distance parcourue (typiquement, la valeur de l'intensité $V(x)$ du signal émis au point x_0 est décroissant en fonction de la puissance n de la distance, voir l'équation 6.1, de la présente page) ;

$$\frac{V(x_0)}{distance(x, x_0)^n} \tag{6.1}$$

– les canaux par voie d'affiche, où un agent désirant communiquer place son message sur un espace accessible par l'ensemble des agents (tableaux noirs) et que les agents concernés consulteront par la suite.

Les robots simulés de la RoboCup utilisent un canal de communication de type propagatoire, dont la portée maximum est de 50 mètres. Les signaux émis sont des messages transmis sous forme de paquets contenant un message et une date d'émission. Par exemple, un agent recevra le message suivant : `(hear 145 "Where's the ball ?")`.

Le choix du destinataire est de plus à la charge de l'émetteur : il peut choisir de communiquer en destination d'un agent unique ou à destination d'un groupe restreint d'agents nommés, à destination d'agents satisfaisant une condition (typiquement, les agents les plus proches de l'émetteur) ou à destination de tous les agents du système. Dans le cas de canaux de communication propagatoires, il se peut que seuls certains agents soient suffisamment proches pour recevoir l'information. Un agent peut ainsi recourir à un intermédiaire pour relayer l'information à tous et demander à un agent à portée suffisante d'émettre le signal reçu à l'identique (ce dernier sera appelé *agent facilitateur*). De même, les canaux de communication propagatoires nécessitent la spécification du destinataire dans les communications particulières.

Communiquer, c'est aussi interpréter les signaux reçus. Un signal prend un sens lorsqu'un agent réunit deux conditions : lorsqu'il perçoit le stimulus en question (son, signal visuel, odeur, paquet TCP, etc.) et lorsque son système interprétatif transforme le signal en sens (symbole) ou en comportement (réaction). Un tel processus d'abduction nécessite l'utilisation d'un formalisme déterminé à l'avance et compréhensible par les deux parties pour décrire la nature des messages envoyés.

KQML [11] est un exemple de langage de communication agent remplissant ce besoin : KQML fournit un symbolisme de haut niveau dans la description de l'acheminement du message vers son destinataire et dans le contenu du message envoyé à l'agent (par le biais de *KIF*, par exemple). Notamment, KQML définit des performatifs de communication de haut niveau permettant à des agents potentiellement hétérogènes d'échanger des informations structurées. Un des désavantages de KQML est cependant son orientation vers la communication point à point entre agents (avec potentiellement l'utilisation d'agents facilitateurs pour relayer un message), ce qui le rend peu adapté à une utilisation dans un contexte de canaux de communication partagés, non fiables et possédant une faible bande passante.

6.2.2 Comprendre son partenaire

L'utilisation d'un langage de communication symbolique de haut niveau implique la compréhension de la nature de son interlocuteur. Dans un environnement où la bande passante des canaux de communication et la capacité de traitement de l'information par le destinataire sont infinies, la synchronisation des états du monde par communication totale serait possible. Mais c'est rarement le cas dans les problématiques du monde réel tout comme dans la simulation (par exemple, les agents de la RoboCup ne peuvent recevoir des signaux que tous les deux cycles en moyenne). Il est donc nécessaire de savoir quelles sont les informations pertinentes à envoyer à un partenaire. Par exemple de savoir en quoi il pourrait être utile (à quels ordres il peut obéir efficacement) tout comme savoir en quoi les informations envoyées peuvent lui servir. Cela implique donc de modéliser son état actuel pour connaître ses capacités, ses besoins et sa connaissance de l'environnement.

La modélisation du partenaire permet de plus de tenter d'obtenir une image de ses buts et de ses croyances. Une telle information est utile pour coopérer de manière plus efficace sur une base temporelle : ainsi, un agent sera à même de reconnaître les plans d'un autre agent, d'anticiper ses réactions et d'ajuster ses propres actions en conséquence. La maintenance de cette information nécessite de prévoir explicitement un modèle de ses partenaires et de les simuler. Dans un système multi-agents, la modélisation respective impacte directement sur le comportement individuel des agents le composant puisque chacun agira en tenant compte des croyances partagées sur l'état de ses partenaires. On trouve dans [35] une architecture résolvant par le partage partiel de modèles le problème de la complexité combinatoire de la modélisation récursive de l'ensemble des composants d'un système multi-agents.

6.2.3 Partager des buts

Dans la plupart des cas, la coopération volontaire implique un but en commun ou tout du moins une alliance objective. Les agents doivent donc tendre individuellement vers un objectif commun, tout en conservant leur identité et leurs buts locaux. Une des solutions les plus simples à cette problématique est le partage de buts : chacun des agents poursuivra un but individuel qui se trouve être celui du reste de l'équipe. Un tel but est dit *persistant* car il est initié par plusieurs agents et sera maintenu tant que tous les agents participant à son accomplissement n'auront pas décidé de son obsolescence. Milind TAMBE définit dans [34] une méthode appelée l'*intention jointe* (*joint intention*), qui permet a plusieurs agents de se mettre d'accord sur un plan, qui sera valide tant qu'une clause d'irrelevance reste invalide. Ce mécanisme est appelé le but persistant joint (*Joint Persistent Goal*).

On trouve aussi dans [15] un exemple de partage de buts. L'architecture qui y est décrite est basée sur celle de RAO et de GEORGEFF [24] (il s'agit d'une architecture BDI, voir la section 5.3.2, page 54). Les agents du système décrit partagent des buts en partageant des éléments extraits des mondes *désirés* par chacun d'entre eux et leurs intentions sont la garantie d'une activité jointe.

6.2.4 Partager des tâches

Le partage des tâches est nécessaire pour garantir un travail coopératif consistant et éviter les conflits et les redondances. Pour partager les tâches à accomplir entre les agents d'un système, il est tout d'abord nécessaire de subdiviser le travail nécessaire en tâches distinctes. Bien souvent, ce découpage est effectué manuellement par le concepteur du système. Le partage de tâches consiste ensuite à trouver un schéma où des agents sont mis en correspondance avec des tâches. Nous dégageons deux types de partage de tâches : l'approche centralisée et l'approche dynamique.

L'approche centralisée peut emprunter deux voies différentes : la voie hiérarchique, où des agents possèdent une autorité directive sur d'autres et peuvent leur demander d'accomplir des tâches. Ces agents décideurs choisissent l'agent le plus à même d'accomplir une tâche en fonction de ses croyances et de ses buts. Une autre voie est la voie de la médiation, typique des systèmes dits égalitaires. Un ou plusieurs agents sont désignés comme *médiateurs* et mettent en correspondance les agents demandeurs et les agents fournisseurs. Ils décident de confier une tâche à un agent fournisseur en fonction des requêtes qui lui ont été faites par les agents demandeurs.

Une approche dynamique peut utiliser un mécanisme d'accointances, où chaque agent possède une matrice mettant en correspondance les compétences requises par les tâches à effectuer et les agents possédant ces compétences. Lorsqu'un agent détermine qu'un autre agent est capable de prendre en charge une tâche, il lui demande de l'effectuer (allocation directe), ce que ce dernier peut refuser. Si un agent ne peut déterminer de destinataire pour une de ses tâches, il délégue ce choix à son *réseau d'accointances*, c'est-à-dire un ensemble d'agents qui à leur tour vont tenter de trouver des agents pour effectuer cette tâche (allocation par délégation).

Un autre mécanisme dynamique est basé sur la métaphore du contrat, qui peut être mise en œuvre pour partager les tâches. Un agent ayant besoin de voir une tâche accomplie émet un appel d'offres à destination de tous les agents qu'il croit pouvoir accomplir cette tâche. Les agents qui reçoivent cet appel et qui peuvent effectivement y répondre envoient une proposition à l'émetteur, qui choisit la proposition la plus adaptée à ses besoins. Ce système présente l'avantage de la simplicité mais multiplie les échanges de messages entre les agents du système et implique un mécanisme de comparaison et de choix des offres.

DURFEE et LESSER proposent dans [9] un modèle apportant une dimension dynamique et une approche à base de plans du mécanisme du contrat et du partage de tâches. Ce modèle considère qu'un contrat n'est qu'un plan : l'engagement qui lie deux parties lors de l'établissement d'un contrat est équivalent au partage d'un plan dont l'objet serait celui du contrat. Ainsi, si l'un des agents d'un système à besoin qu'une tâche soit accomplie (ce modèle s'applique aussi au partage de ressources), il propose un plan à ses voisins. Si un voisin décide de lui venir en aide, il accepte le plan et le partage avec l'émetteur. Les agents peuvent de plus proposer des aménagements aux plans soumis (contre-plans), voir annuler un plan en cours d'exécution s'il ne correspond plus à leurs buts internes (voir la section 6.2.5, de la présente page).

6.2.5 Savoir planifier

La coopération est basée sur une succession d'actions atomiques exécutées en parallèle par les agents d'un système et synchronisées entre elles par l'envoi de messages. Les effets des actions entreprises ainsi que leur perception n'étant pas immédiates, les agents d'un système ont tout intérêt à estimer à l'avance l'évolution de leurs actions et des actions de leurs partenaires afin de déclencher leurs actions au moment opportun. De même, un agent doit être capable d'évaluer les actions futures de ses partenaires pour adopter une bonne attitude passive.

On trouve dans [9] un mécanisme de planification coopérative. DURFEE et LESSER définissent un mécanisme principalement adapté aux systèmes interprétatifs distribués, mais adaptable aux systèmes multi-agents. Ce modèle est basé sur le concept de nœuds autonomes (agents) qui modèlent leur rôle dans le système ainsi que leur comportement attendu. Ces nœuds modélisent les activités du système et utilisent cette modélisation pour proposer des solutions et des hypothèses (ou *Partial Global Plans*) aux nœuds voisins (de même, si un nœud s'aperçoit que ses plans originaux sont obsolètes, il peut décider de communiquer cette information à ses voisins au dessus d'un certain seuil d'obsolescence). Les voisins peuvent modifier leurs plans en fonction d'un critère d'importance décidé par l'agent émetteur et de l'importance qu'ils accordent à leurs plans propres. Des règles d'autorité sont de plus définies entre les nœuds afin de faciliter la mise en place d'autorités locales.

6.2.6 Adopter une attitude

L'idée de l'attitude est en intelligence artificielle très similaire à celle développée par les cogniticiens. L'attitude consiste à adopter une démarche (vide de but) et à se comporter en fonction de règles implicitement fixées par cette démarche. Lorsqu'un agent décide d'adopter une attitude, il modifie son état mental en conséquence afin d'adapter son comportement : l'agent peut par exemple décider de passer dans une attitude de coopération, d'indifférence ou de compétition vis-à-vis d'autres agents. Un tel mécanisme est utile pour améliorer la consistance du comportement social des agents composant un système et éviter les comportements sociaux erratiques entre deux agents (dans notre cas, il pourrait s'agir de deux agents adoptant une attitude respective de passeur et de buteur afin d'éviter des situations de passage de balle non nécessaires entre eux). On trouve dans [15] un mécanisme d'allocation et d'adoption d'attitudes ainsi que de publicité sur ces adoptions : un chef d'équipe permet à chaque agent composant une formation d'adopter une attitude précise et de s'y tenir (*commit*). Chaque agent ainsi engagé coopérera et partagera des buts jusqu'à l'accomplissement de son engagement.

On trouve ainsi dans [13] un exemple d'attitude de coopération passive et altruiste. Le principe est qu'un agent adoptant une attitude altruiste va consacrer une partie de ses ressources (énergie, temps, etc.) à améliorer l'état global du monde, en prévision des actions futures de ses partenaires. Ainsi, si un obstacle gène le passage d'un robot, il pourra décider dans une certaine mesure de le déplacer pour libérer le passage plutôt que de le contourner, même si le contournement est moins coûteux que la libération du passage. En effet, la suppression d'un obstacle est profitable à l'ensemble de la communauté des agents du système. Cette méthode de coopération est issue directement de la théorie des jeux : dans un monde où tous les participants coopèrent, l'ensemble de la communauté profite de cette coopération. Mais dans le cas où peu de participants coopèrent, ceux-ci sont désavantagés. Ainsi, l'adoption d'une attitude altruiste ou égoïste permet de se comporter de manière optimale selon que la tendance est favorable à la coopération ou à l'égoïsme et permet de même de connaître le comportement des agents du système.

6.2.7 Tolérer les pannes

La détection des pannes est un des élément les plus cruciaux pour chacun des agents lorsqu'il s'agit de manipuler son environnement. La détection des pannes et le contrôle des performances d'un agent permet d'éviter les problèmes de *deadlock* (boucle infinie basée sur l'attente d'une condition non réalisable) et d'éviter les stratégies non optimales. Un contrôle régulier et relativement simple de l'environnement suffit à résoudre ces problèmes. Mais dans la problématique des systèmes multi-agents, la détection et le contrôle des pannes repose sur un mécanisme plus complexe. Une panne peut être de plusieurs natures : un agent oublie d'indiquer aux autres qu'il a terminé une action dont les autres ont besoin, un agent oublie d'exécuter une action dont les autres agents attendent le résultat, un agent oublie de demander à un autre agent d'exécuter une action dont il attend le résultat, un agent réussit de manière inattendue une action considérée comme impossible mais les autres ne prennent pas en considération cette réussite et se comportent de manière non optimale, un agent décide d'effectuer une action qu'un autre agent entame au même moment mais tous les deux renoncent à la continuer croyant que l'autre s'en chargera, etc. Milind TAMBE décrit dans [34] quelques unes de ces problématiques appliquées à la simulation militaire.

La détection des pannes est un problème qu'on peut en partie résoudre par une analyse constante du système et par la mesure de ses performances (un agent du système peut être dédié à la détection des pannes). En revanche, la prédiction des pannes est structurellement difficile. En effet, une panne est par nature un échec non prédictible. Ainsi, un système multi-agents aura tout intérêt à être muni d'un système performant de récupération d'échecs et de pannes. Notamment, la granularité du traitement des pannes est un paramètre à prendre en compte : selon la gravité de la panne, un agent tente de la résoudre seul ou il décide que le système tout entier est concerné et doit être mis au courant de la nature de la panne et des remèdes à y apporter. D'une manière générale et à condition que le coût de la communication soit inférieur au bénéfice, lorsqu'un agent du système est incapable d'exécuter sa part du plan global, il est de sa responsabilité d'avertir le reste du système de la nature de l'échec rencontré et ainsi d'initier un processus de recouvrement concerté. Si le coût de l'annonce d'un échec est important, les agents d'un système doivent prendre eux-mêmes en charge la surveillance des actions des membres de leur système.

Le recouvrement implique à la fois une réévaluation totale ou partielle des plans engagés, mais aussi une réponse, soit par le contournement du problème (adoption d'une solution annexe), soit par une nouvelle tentative d'exécution du plan. On trouve dans [34] un exemple de mécanisme de recouvrement de pannes : Milind TAMBE définit des « opérateurs » d'équipe, qui sont des actions jointes dans lesquelles chacun des agents du système joue un rôle précis (par exemple, l'opérateur *engager opposant* trouve sa place dans le cas d'une simulation de combat aérien). Chacun de ces opérateurs est régulièrement évalué : chaque agent engagé dans un opérateur détermine grâce à une heuristique critique s'il est possible de l'achever. Si ce n'est pas le cas et que la raison est l'échec d'un des rôles (typiquement dans le cas de la simulation de combat aérien, un hélicoptère abattu), alors l'agent évaluateur prend en charge ce rôle s'il en est capable et communique ce changement. S'il en est incapable, il attend qu'un autre agent le fasse pour lui (et potentiellement, détruit l'opérateur si aucun n'en est capable).

Dans [22], Lynne E. PARKER définit la méthode *ALLIANCE*, une méthode alternative. Chacun de ses agents passe en revue les comportements possibles et estime le *désir* qu'il a d'adopter chaque comportement (en se basant sur les informations reçues de ses récepteurs, de la communication avec les autres agents, des aspects inhibiteurs de son propre comportement ainsi que de ses motivations et buts). Le désir de chacun de ces comportements possibles augmente en fonction du temps et lorsque ce désir dépasse un certain seuil, le comportement est activé. Dans le cas où un agent A est en charge d'un comportement (ce qui est déterminé par la communication ou par l'observation de son comportement), le désir de l'agent B de voir ce comportement activé croît beaucoup plus lentement. Ainsi, si l'agent A est incapable d'accomplir le comportement sélectionné, l'agent B finira par activer ce comportement. Ce mécanisme est appelé l'impatience et permet une résolution émergente de pannes dans un système multi-agents.

6.3 Le cas du jeu d'équipe

Les jeux sont des domaines de recherche et d'application privilégiés pour l'intelligence artificielle. Tout au long de l'histoire de l'intelligence artificielle, on trouve des exemples de défis posés au travers de jeux. D'abord sous forme de jeux simples (puzzles, casses-têtes mathématiques) adaptés à la logique informatique, puis sous forme de jeux complexes en interaction avec l'esprit humain (jeux d'échecs, de dames, de société, etc.), où la formidable puissance de calcul d'un ordinateur ne le rend pas nécessairement supérieur à l'esprit humain, qui est naturellement capable d'apprentissage, d'initiative, d'analyse et d'adaptation. Les programmes informatiques de jeu profitent des progrès technologiques de l'intelligence artificielle. Ainsi, les approches connexionnistes (réseaux de neurones), évolutionnistes (évolution génétique de programmes) et l'intelligence artificielle distribuée (répartition des centres d'évaluation) sont des techniques utilisées dans l'intelligence artificielle appliquée au jeu.

Le jeu d'équipe introduit deux dimensions supplémentaires, celle de la multiplicité des intervenants (coéquipiers ou adversaires) et celle de la communication entre partenaires. L'intelligence artificielle distribuée est une approche en adéquation directe avec le jeu d'équipe, puisqu'elle permet d'utiliser les compétences locales de chacun des membres de l'équipe. Notamment, la somme des points de vue locaux permet de construire des stratégies émergentes efficaces pour la globalité d'une équipe.

Une équipe est un cas particulier d'organisation des systèmes multi-agents. Il s'agit d'un système multi-agents coopératif dont les membres, coopèrent et se coordonnent mutuellement et tournent de plus leurs efforts coopératifs vers l'accomplissement d'un but commun. Former une équipe implique donc de partager des buts (voir la section 6.2.3, page 66) et d'être capable de se mettre d'accord sur un plan commun. Ainsi, raisonner sur une équipe implique l'hypothèse que les agents la composant sont automatiquement amicaux et coopératifs entre eux et automatiquement hostiles aux plans de l'équipe adverse. En effet, une autre des spécificités du jeu d'équipe est l'existence d'une équipe adverse dont le but est d'empêcher le but partagé opposé de s'accomplir. Nous allons traiter dans cette section des problématiques spécifiquement inhérentes au jeu d'équipe.

6.3.1 Gérer les conflits

La gestion des conflits dans le cas d'un jeu d'équipe permet d'éviter que l'adversaire ne tire parti d'une faille d'organisation. La contrainte adversariale rend d'autant plus cruciale la limitation des redondances et des conflits. L'histoire de la RoboCup a montré qu'une des tactiques gagnantes consiste à provoquer volontairement des conflits parmi les agents de l'équipe adverse.

Une solution adoptée couramment pour régler les conflits est la mise en place de formations et de rôles : un rôle décrit un but et un comportement au sein d'une formation dont tous les membres partagent les buts. La distinction de rôles permet de limiter les conflits, notamment de localisation physique, ainsi que les redondances. Les agents participant à une formation doivent tous en être conscients et connaître leur rôle, ce qui implique l'échange de messages de synchronisation ou l'utilisation de plans préétablis (par exemple via le *locker-room agreement*, voir la section 2.2.4, page 15). D'une manière générique, les équipes d'agents ont tout à gagner à privilégier des équipes flexibles par rapport à des équipes rigides, composées invariablement des mêmes agents. Ainsi, le rôle d'un agent au sein d'une formation est flexible et peut évoluer au fil du temps, en fonction de ses motivations à rester au sein de la formation [27]. De même, une formation peut cesser d'exister lorsqu'une clause d'invalidation est déclenchée (typiquement, lorsque la raison d'être d'une formation a été atteinte ou lorsque son objectif n'est plus atteignable).

Une autre approche consiste à définir un capitaine. Le capitaine est un agent membre de l'équipe qui dispose de privilèges directifs particuliers. La désignation du capitaine peut faire partie de plans prédéfinis ou alors faire l'objet d'une élection en cours de jeu. De même, une stratégie alternative consiste à définir des capitaines dont l'autorité directive sera locale à une formation.

6.3.2 Adopter une attitude passive

Dans la plupart des jeux d'équipe et plus spécifiquement dans les jeux de balle, une minorité des membres d'une équipe agit réellement à un instant donné. Ses coéquipiers ne doivent cependant pas rester inactifs pendant ce temps là : une bonne stratégie de jeu d'équipe consiste à adopter une attitude passive utile. L'attitude passive inclut le positionnement à une localisation stratégique, une attitude de gêne de l'adversaire, la mise de ses compétences à disposition des agents actuellement actifs ou tout simplement le repos. Le choix d'une bonne position peut par exemple s'effectuer au moyen de la modélisation de champs de potentiel (attraction par les adversaires et répulsion par les coéquipiers) [28].

6.3.3 Communiquer en destination de tous

Bien souvent, les canaux de communication utilisés dans les jeux d'équipe sont de type propagatoire et sont partagés entre les différents agents du système, qu'ils soient coopératifs ou antagonistes. Le premier challenge d'un tel canal de communication est d'arriver à identifier l'émetteur du message avec précision : notamment, d'être capable de séparer les messages émis par un membre de son équipe des messages contrefaits émis par l'équipe adverse afin d'induire en erreur la stratégie de communication d'une équipe. Cette stratégie de contrefaçon est contrariée par des mécanismes d'encryption ou de date des messages émis et reçus [28].

Un autre problème est la collision de messages. Dans certains canaux propagatoires, l'émission de plusieurs messages simultanés brouille totalement le canal et empêche la communication. Si de tels accidents sont inévitables, il est tout du moins possible de les limiter. Ainsi, un mécanisme de limitation de collision simple serait que lorsqu'un agent communique, les agents qui reçoivent le message tentent de déterminer si ce message appelle une ou plusieurs réponses de la part de l'équipe. Un tel agent détermine si le message reçu ne nécessite qu'une réponse et s'il lui échoit personnellement de la donner, auquel cas il y répond le plus vite possible. Si la réponse nécessite plusieurs réponses (par exemple, si la question est « qui peut m'aider ? »), les agents concernés ne doivent pas tous répondre le plus vite possible afin d'éviter les collisions. Un mécanisme de délai (*locker-room agreement*, heuristique câblée, tirage aléatoire) permet d'étaler dans le temps les réponses et ainsi de favoriser l'utilisation exclusive du canal de communication. Cependant, en cas de collision, un mécanisme de recouvrement de pannes efficace pourrait être basé sur une idée proche de celle de l'algorithme *CSMA-CD* (*Carrier Sense Multiple Acces with Collision Detect*), utilisé pour gérer les collisions sur un brin éthernet : il s'agit pour chacun des émetteurs de répéter un message endommagé par une collision ultérieurement, après une période d'attente aléatoire.

6.3.4 Coopérer contre un adversaire

Le jeu d'équipe fait intervenir deux équipes opposées, dont le but est de faire échouer celui de l'équipe adverse. Tout comme la modélisation d'un coéquipier facilite la coopération (prédiction de mouvements et élimination de certains besoins de communication), la modélisation de l'adversaire permet de se comporter de manière plus consistante contre lui. Une bonne stratégie est de modéliser l'adversaire et d'estimer qu'il se comportera de manière optimum en fonction de ses critères particuliers et en fonction de l'état du monde actuel supposément connu par lui. Ainsi, la prise de décision effectuée s'effectuera à l'échelle d'une réponse à l'hypothèse d'un comportement optimum et se repose sur la supposition que la décision prise restera acceptable si l'opposant ne joue pas un coup optimal. Cette stratégie est inspirée de l'algorithme *minimax*, très répandu en intelligence artificielle appliquée aux jeux, qui fait intervenir une notation du contexte du jeu en cours et de l'hypothèse que l'adversaire prendra le meilleur choix qui lui est offert. Elle fonctionne aussi dans le cas de la modélisation du comportement coopératif des coéquipiers.

Un exemple d'utilisation de cette modélisation est détaillé dans [30]. La méthode décrite, IMBBOP, consiste à résoudre dans une simulation informatique de football les dilemmes survenant près d'un but adverse. Typiquement, le joueur à le choix entre tenter de marquer un but lui-même ou passer la balle à un coéquipier qui pourrait avoir plus de chances que lui de marquer le but. L'algorithme général consiste à déterminer le moment t où l'équipier X aura la possibilité d'accomplir l'objectif G (par exemple, marquer un but au temps T), que l'agent A à besoin d'accomplir ou de voir accomplir, en considérant que l'agent X adoptera un comportement optimal. Plus la différence $T - t$ est grande, plus l'agent X aura de chances d'accomplir G en adoptant un comportement non optimum. Ainsi, l'agent A décide de déléguer l'action à l'agent X lorsque $T - t$ dépasse un certain seuil.

6.3.5 La programmation orientée équipe

La programmation orientée équipe est une abstraction qui permet au programmeur de ne pas avoir à s'occuper des mécanismes de constitution, de synchronisation et de maintenance d'équipes d'agents : une couche d'abstraction prend en charge ces aspects et les isole des parties fonctionnelles de l'agent. David PYNADATH définit ainsi par exemple dans [23] un tel mécanisme d'abstraction, appelé *TEAMCORE*, qui s'occupe de toutes les couches basses de la maintenance des équipes d'agents. La programmation orientée équipe permet au programmeur de se concentrer sur l'aspect fonctionnel de son équipe et de définir des rôles et des plans pour résoudre le problème présenté. Le principe est de définir une hiérarchie de buts et de sous-buts (nécessaires pour atteindre le but final, typiquement remporter la partie ou remplir une mission), auxquels l'entité « équipe » va réagir de manière réactive et émergente. Pour chacun des plans et des sous-plans à atteindre, la programmation orientée équipe permet d'assigner un ou plusieurs rôles aux membres de l'équipe (la méthode d'assignation de rôles étant assurée par la couche d'abstraction de programmation orientée équipe) par plan actif. La robustesse de l'organisation sociale de l'équipe étant assurée par le dispositif de programmation orientée équipe, les agents en tant qu'individus n'ont plus qu'à s'occuper de la robustesse de leurs propres rôles.

Chapitre 7

Conclusion

Ce mémoire décrit la conception et l'implémentation d'une équipe de robots footballeurs simulés. Cette simulation est réalisée par un système multi-agents dans lequel chaque agent modélise un robot footballeur. Son implémentation tient compte des problématiques du milieu (fortement dynamique et bruité) dans lequel les agents évoluent. Elle fournit de plus un mécanisme pour contrôler les erreurs dans l'exécution des plans des agents, ainsi qu'un mécanisme de visualisation des actions en cours et un mécanisme de modélisation par hypothèses du comportement des coéquipiers.

Notre équipe est basée sur l'implémentation de l'équipe *CMUnited 2000* (*CMU-2000*) décrite dans [25, 28, 29]. L'équipe CMU-2000 est développée à l'université *Carnegie Mellon* et fût vainqueur des compétitions de 1998 et de 1999. Nous avons utilisé les couches basses (couche et protocole réseau, analyse des informations sensorielles, effecteurs bas niveau, exécution temporelle asynchrone, etc.) de CMU-2000.

L'activité comportementale de nos agents est basée sur l'exécution de plans, représentés par autant d'instances différentes qu'il existe de plans en cours. Le choix actuel de l'exécution d'un plan est laissé à l'ordonnanceur des tâches qui exécute la plus prioritaire du cycle en cours. Ainsi, l'ordonnancement des tâches est le mécanisme qui permet de passer de l'exécution d'une tâche à une autre, ce qui produit un mécanisme multi-tâches non préemptif. Un mécanisme plus fin serait l'affectation de quantificateurs de priorité aux différentes tâches. Un algorithme de répartition de ressources proche de celui des ordonnanceurs de systèmes d'exploitation multitâches préemptifs permettrait une réelle exécution concurrente des tâches en cours.

De même, l'extension du mécanisme des plans serait une chose tout à fait souhaitable. Notamment, les conditions d'invalidation actuelles sont communes à toutes les instances, ce qui pourrait être modifié au profit de la définition de conditions d'invalidation propres à chaque instance. Ainsi, un agent pourrait déclencher un plan de type « aller chercher la balle » sur la foi de l'analyse des intentions d'un autre agent et abandonner ce plan lorsqu'il se rend compte que les intentions de cet agent ne sont pas ce qu'il a évalué, bien que l'état du monde rende possible l'exécution du plan.

Les activités de visualisation de l'activité des agents implémentés est assurée par une couche de primitives graphiques symboliques, qui permettent de visualiser et de suivre l'évolution d'un symbole de la mémoire de l'agent. Il est de la responsabilité des plans en cours de décider des symboles à afficher via la couche de primitives graphiques. On peut imaginer l'extension de ce mécanisme : la conception de plans de pure visualisation permettraient à l'opérateur de visualiser des structures de données arbitraires. Nous comptons de plus développer des périphériques de visualisation supplémentaires, notamment un affichage PostScript et un affichage en trois dimensions utilisant OpenGL.

Nous proposons une méthode de modélisation des agents de notre système qui se baserait sur l'adoption de la même modélisation pour les agents de son système que pour l'architecture interne de l'agent. En effet, le système mis en place souffre de la séparation de l'heuristique d'analyse comportementale et de la fonction comportementale réelle. Ceci rend difficile l'analyse du comportement des agents du système, puisque cette dernière se base sur une évaluation externe au processus comportemental. Nous proposons la mise en place d'un mécanisme où l'agent « réel » (l'agent exécuté par un processus UNIX et communiquant avec le serveur de la RoboCup) simule par des structures identiques à la sienne les autres agents du système et les exécute en parallèle à sa propre exécution. L'agent réel s'interfacerait avec les agents simulés par des canaux de communication semblables et un protocole identique à celui utilisé pour communiquer avec le serveur. Il se chargerait d'envoyer des informations sensorielles aux agents simulés en se basant sur ses croyances propres.

Ainsi, l'agent réel simulerait par ce biais le comportement de ses partenaires en utilisant les heuristiques même de leur comportement réel. Les actions des agents simulés seraient perçues via leurs effecteurs et permettraient à l'agent réel de déterminer les actions probables de ses coéquipiers et adversaires en se basant sur les heuristiques et la représentation mêmes de leur exécution. La communication entre de tels agents n'étant pas soumise à des contraintes de bande passante, elle pourrait se faire ainsi de manière privilégiée et pourrait leur permettre de s'échanger des plans et d'estimer ce à quoi les autres sont occupés.

Cette technique se base cependant sur l'hypothèse que la modélisation du comportement utilisée est bien identique aux fonctions comportementales des agents simulés. Ainsi, la simulation des adversaires par cette technique serait approximative car elle se baserait sur des prédicats comportementaux différents. Cependant, une telle solution permettrait d'approximer la modélisation du comportement des adversaires en se basant sur l'hypothèse que les primitives comportementales (aller vers la balle, envoyer la balle dans les buts, etc.) des adversaires sont proches des primitives comportementales de notre équipe.

Une autre problématique est l'aspect récursif des simulations d'agents. Si nous prenons l'hypothèse que l'agent A et l'agent B se simulent mutuellement, A simulera le comportement de B sans tenir compte du fait que son propre comportement est simulé et estimé par B, qui agit donc en conséquence (le même mécanisme est valable pour l'agent B). Il serait donc nécessaire pour une réelle coordination de simuler récursivement les modèles des différents agents. La problématique de ce type de solution est la rapide explosion combinatoire des modèles simulés. Ce problème pourrait être résolu par le partage de modèles ou par un mécanisme de consultation d'états mentaux privilégiés entre les agents simulés.

Le modèle utilisé pour implémenter les agents de notre système laisse de plus volontairement de côté les problématiques de la communication dans un milieu bruité et non fiable comme celui de la simulation de la RoboCup au profit d'une estimation comportementale silencieuse et donc d'une synchronisation implicite. Le coût de l'attente d'une information ou d'une réaction de la communication entre agents pourrait être limité en mettant en place un paradigme de communication informative à « sens unique. » C'est-à-dire un paradigme dont les messages émis n'attendraient pas de réponse mais auraient pour but d'informer les coéquipiers. Par exemple, un agent décidera d'annoncer le déclenchement d'une action importante pour les buts globaux de l'équipe (heuristique rigide) ou il décidera de communiquer à un agent une information invalidant le plan en cours d'exécution. L'utilisation du coach et de la simulation des agents de l'équipe au sein de l'agent coach réel permettrait de tester un tel paradigme de communication.

Nous comptons explorer les possibilités de Scheme dans la modélisation des actes de langage. Les performatifs du langage pourraient être facilement représentés par des procédures Scheme : en effet, le mécanisme de l'analyse et le traitement d'un performatif est très similaire à celui de l'analyse et de l'exécution d'une procédure fonctionnelle. Ainsi, le protocole de communication développé à partir de Scheme pourrait bénéficier de structures de données complexes et des facilités fonctionnelles d'un langage fonctionnel. De même, l'utilisation de Scheme permettrait de factoriser les fonctionnalités du moteur Scheme implémenté dans nos agents tout en simplifiant l'intégration d'informations au sein des instances de nos agents.

Nous avons été confrontés à la problématique des actions complexes en milieu bruité, notamment pour les « effecteurs » non atomiques (tourner la balle, dribbler avec elle jusqu'à un point, etc.). Actuellement, ces effecteurs sont écrits de manière heuristique. Nous envisageons une réécriture de ces effecteurs de manière non statique en nous basant sur des algorithmes d'apprentissage automatique, par exemple en utilisant les algorithmes génétiques. Cette approche a par exemple déjà été étudiée et utilisée avec succès dans [2, 28].

Bibliographie

[1] ABCHICHE, N. *Élaboration, implémentation et validation d'une approche distribuée pour l'intégration de modèles de raisonnement hétérogènes : application au diagnostic de pannes électriques*. PhD thesis, Université Paris 8, 1999.

[2] ANDRE, D., AND TELLER, A. Evolving team darwin united. In *RoboCup* (1998), pp. 346–351.

[3] BRUSTOLONI, J. C. Autonomous agents : Characterization and requirements. Tech. rep., School of Computer Science, Carnegie Mellon University, 1991.

[4] CARVER, N., AND LESSER, V. The evolution of blackboard control architectures. Tech. Rep. 92-71, CMPSCI, 1992.

[5] CHESS, D., GROSOF, B., HARRISON, C., LEVINE, D., PARRIS, C., AND TSUDIK, G. Itinerant Agents for Mobile Computing. *IEEE Personal Communications 2*, 5 (1995), 34–49.

[6] CONRADIE, L., AND MOUNTZIA, M. A relational model for distributed systems monitoring using flexible agents. IEEE Computer Society Press.

[7] CREVIER, D. *À la recherche de l'intelligence artificielle*. Flammarion, 1993.

[8] DROGOUL, A. When ants play chess (or can strategies emerge from tactical behaviours ?). In *From Reaction to Cognition — Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-93 (LNAI Volume 957)* (1995), C. Castelfranchi and J.-P. Müller, Eds., Springer-Verlag : Heidelberg, Germany, pp. 13–27.

[9] DURFEE, E. H., AND LESSER, V. R. Partial global planning : A coordination framework for distributed hypothesis formation. Tech. rep., University of Michigan and University of Massachussetts, 1991.

[10] FERBER, J. *Les systèmes multi-agents*. InterEditions, 1995.

[11] FININ, T., FRITZSON, R., MCKAY, D., AND MCENTIRE, R. KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)* (Gaithersburg, Maryland, 1994), ACM Press, pp. 456–463.

[12] FRANKLIN, S., AND GRAESSER, A. Is it an agent, or just a program ? : A taxonomy for autonomous agents. Tech. rep., University of Memphis, 1996.

[13] GOLDMAN, C. V., AND S., R. J. Emergent coordination through the use of cooperative state-changing rules. Tech. rep., Computer Science Department, Hebrew University, 1994.

[14] HIROAKI, K., AND AL. Robocup : the Robot World Cup Iniative. In *JSAI AI-Symposium 95 Proceedings* (1995).

[15] KINNY, D., LJUNGBERG, M., RAO, A., SONENBERG, E., TIDHAR, G., AND WERNER, E. Planned team activity. In *Pre-Proceedings of MAAMAW'92* (1992), A. Cesta, R. Conte, and M. Miceli, Eds.

[16] KRÖSE, B., AND VAN DER SMAGT, P. An introduction to neural networks. Tech. rep., University of Amsterdam, 1996.

[17] MAES, P. Modeling adaptative autonomous agents. Tech. rep., MIT Media-Laboratory, 1990.

[18] MATOS, N., SIERRE, C., AND JENNINGS, N. R. Determining successful negotiation strategies : An evolutionary approach. Tech. rep., IIIA, Artificial Intelligence Research Institute, 1998.

[19] METRAL, M. E. Design of a generic learning interface agent. Master's thesis, Massachussets Institute of Technology, 1993.

[20] NODA, I. Soccer server : a simulator of robocup. In *JSAI AI-Symposium 95 Proceedings* (1995).

[21] OSAWA, E. Viewing a soccer game as a canonical problem for multiagent research. In *JSAI AI-Symposium 95 Proceedings* (1995).

[22] PARKER, L. E. Alliance : An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation 14*, 2 (1998), 220–240.

[23] PYNADATH, D. V., TAMBE, M., CHAUVAT, N., AND CAVEDON, L. Toward team-oriented programming. Tech. rep., Information Science Instritute and Computer Science Department, 1999.

[24] RAO, A. S., AND GEORGEFF, M. P. Bdi agents : From theory to practice. Tech. rep., Australian Artificial Intelligence Institute, April 1995.

[25] RILEY, P., STONE, P., MCALLESTER, D., AND VELOSO, M. ATT-CMUnited-2000 : Third place finisher in the robocup-2000 simulator league. Tech. rep., Computer Science Department, Canegie Mellon University, 2000.

[26] ROBOCUP TEAM. *Soccerserver Manual*, 4.02 ed., 1999.

[27] STONE, P., , AND VELOSO, M. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence* (1999).

[28] STONE, P., RILEY, P., AND VELOSO, M. The CMUnited-98 champion simulator team. Tech. rep., Computer Science Department, Canegie Mellon University, 1998.

[29] STONE, P., RILEY, P., AND VELOSO, M. The CMUnited-99 champion simulator team. Tech. rep., Computer Science Department, Canegie Mellon University, 1999.

[30] STONE, P., RILEY, P., AND VELOSO, M. Defining and using ideal teammate and opponent agents models. Tech. rep., AT&T Labs, Carnegie Mellon University, 2000.

[31] STONE, P., AND VELOSO, M. Multiagent systems : A survey from a machine learning perspective.

[32] STROUSTRUP, B. *Le langage C++*, 3 ed. CampusPress France, 1999.

[33] TAMBE, M. Implementing agents teams in dynamic multi-agent environments. Tech. rep., University of Southern California, 1997.

[34] TAMBE, M. Toward flexible teamwork. *Journal of Artificial Intelligence Research 7* (1997).

[35] TAMBE, M., AND ROSENBLOOM, P. S. Architectures for agents that track other agents in multi-agent worlds. Tech. rep., University of Southern California, 1995.

[36] VALVASSORI, M. Fungus, une bibliothèque d'aide à la création de simulateurs de systèmes complexes par l'approche multi-agents. Master's thesis, Université Paris 8, 2001.

[37] WOOLDRIDGE, M., AND JENNINGS, N. R. Pitfalls of agent-oriented development. Tech. rep., Queen Mary & Westfield College, University of London, 1998.