Département de formation doctorale en informatique **École doctorale IAE + M**
UFR STMIA

Contribution à la modélisation de l'interaction agent/environnement : Modélisation stochastique et simulation parallèle

THÈSE

présentée et soutenue publiquement le 7 novembre 2001

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Makram BOUZID

Composition du jury

Président :	M. André Schaff, Professeur Université Henri Poincaré - Nancy 1
Rapporteurs :	M. Rachid Alami, Directeur de Recherche LAAS-CNRS M. Abderrafiaa Koukam, Professeur Université de Technologie de Belfort - Montbéliard
Directeur :	M. François Charpillet, Directeur de Recherche INRIA Lorraine
Examinateurs :	M. Vincent Chevrier, Maître de Conférences Université Henri Poincaré - Nancy 1 M. Stéphane Vialle, Chef de travaux SUPELEC Campus de Metz
Invité :	M. Richard Washington, NASA Ames Research Center

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Résumé

Mots-clés: Simulation, interaction, stochastique, agent, environnement, parallélisme.

Ce mémoire décrit les travaux de recherche effectués au cours de ma thèse, au sein de l'équipe MAIA du laboratoire LORIA / INRIA Lorraine, en collaboration avec SUPELEC, et avec le soutien du CCH. Nos travaux de recherche se situent à l'intersection des systèmes multi-agents (SMA) et du parallélisme, et concernent plus précisément la simulation parallèle des SMA. Un système multi-agent consiste en un ensemble d'agents qui agissent et interagissent au sein d'un environnement. Un agent est une entité autonome qui perçoit partiellement son environnement et sur lequel elle agit localement (dite "agent situé").

Dans cette thèse, nous abordons principalement deux problèmes. Le premier est la modélisation des incertitudes et des erreurs qui peuvent se produire au niveau des capteurs et des effecteurs des agents au moment de leurs interactions avec l'environnement, en vue d'une simulation multi-agent. De telles modélisations sont rarement présentes dans un SMA, et il en découle des difficultés à retrouver les résultats obtenus au cours de la simulation lors du passage à l'expérimentation sur le système réel. Le second problème concerne l'exploitation du parallélisme inhérent des systèmes multi-agents (SMA), afin d'obtenir de bonnes performances parallèles (en termes de réduction du temps d'exécution et/ou de traitement de problèmes de plus grandes tailles). En effet, la plupart des implantations parallèles de SMA proposées dans la littérature ne sont pas convenables pour obtenir de bonnes performances parallèles.

Nous proposons donc un modèle formel de systèmes multi-agents, qui comprend un modèle stochastique d'interaction entre l'agent et l'environnement, et qui s'inspire des Processus de Décision Markoviens Partiellement Observables (POMDP). Notre modèle permet de reproduire les incertitudes et les erreurs des capteurs des agents, ainsi que celles de leurs effecteurs, au cours d'une simulation multi-agent. Nous proposons aussi un modèle de simulation parallèle basé sur la répartition des conflits survenant entre les agents, et sur un équilibrage dynamique de charge entre les processeurs, s'appuyant sur un mécanisme de double *work-pools*.

Afin de valider ces deux modèles, nous les avons utilisés pour réaliser un simulateur parallèle de robots mobiles dans un environnement structuré. Ce simulateur constitue une première version d'une plate-forme de simulation multi-robot, permettant de tester et de valider divers comportements pour les robots. Le simulateur réalisé est générique, extensible, facile à utiliser et s'exécute en parallèle sur diverses architectures, ayant des systèmes d'exploitation différents : la SGI-Origin 2000 du CCH et la SGI-VWS 540 de SUPELEC.

Nous avons mené quelques expériences bien spécifiques, afin de montrer l'importance de la modélisation des incertitudes et des erreurs des capteurs et des effecteurs des agents au niveau d'un simulateur multi-agent, et de confirmer ainsi nos hypothèses de départ. Nous avons aussi vérifié l'efficacité de notre modèle de simulation parallèle, au travers des expériences menées sur deux machines parallèles différentes. Nous avons obtenu des

accélérations satisfaisantes sur les deux machines. Nous avons enfin discuté l'apport de nos modèles et de notre simulateur pour la simulation multi-agent en général, ainsi que pour la planification et l'apprentissage multi-agent en particulier.

Abstract

Keywords: Simulation, interaction, stochastic, agent, environment, parallelism.

This thesis describes my PhD work carried out within the MAIA research group of the LORIA / INRIA Lorraine laboratory, in collaboration with SUPELEC, and the support of the CCH center. The reported work belongs at the same time to the multi-agent system (MAS) and parallelism domains, and more precisely to the parallel simulation of MAS. A multi-agent system consists in a set of agents which act and interact within an environment. An agent is an autonomous entity which perceives its environment partially and on which it acts locally (called "situated agent").

In this thesis, we mainly tackle two problems. First one concerns the modeling and simulation of situated agents, together with the unreliability of their sensors and effectors. The majority of the proposed multi-agent simulators do not take into account the errors and uncertainties of the agent sensors and effectors, and this explains why they have difficulties to keep the obtained simulation results true when passing to the real system experimentation. The second problem relates to the exploitation of the inherent parallelism of multi-agent systems, in order to obtain good parallel performances (i.e. reducing the execution time and/or processing problems of bigger sizes). Indeed, the majority of the parallel implementations of MAS proposed in the literature are not suitable to obtain good parallel performances.

We thus propose a formal model of multi-agent systems, including a stochastic model of the agent/environment interaction, inspired by the Partially Observable Markov Decision Processes (POMDP). Our interaction model reproduces the errors and uncertainties which may occur at the agent sensor and effector levels, during a multi-agent based simulation. We also propose a parallel simulation model for multi-agent systems based on the distribution of the conflicts occurring between the agents, and on a dynamic load balancing mechanism between the processors, using two *work-pools*.

In order to validate these two models, we used them to build a parallel simulator of mobile robots in a structured environment. This simulator represents a first version of a multi-robot simulation testbed, which allows to test and validate various robot behaviors. The built simulator is generic, extensible, easy to use, and it can run in parallel on various architectures, with different operating systems: the SGI-Origin 2000 of the CCH center and the SGI-VWS 540 of SUPELEC.

We made some quite specific experiments, in order to show the importance of the modeling of the errors and uncertainties of the agent sensors and effectors, while simulating multi-agent systems, and to confirm therefore our starting assumptions. We also checked the effectiveness of our parallel simulation model, through the undertaken experiments on two different parallel machines. We obtained satisfactory speed-ups on the two machines. We finally discussed the contribution of our models and of our simulator

for multi-agent based simulations in general, as well as for multi-agent planning and learning in particular.

Remerciements

Je tiens à remercier toutes les personnes qui ont participé de près ou de loin à l'aboutissement de cette thèse. Je vais essayer de présenter la liste de tout le monde, de façon parfois générique (vu la longueur de cette liste), tout en espérant n'oublier personne.

Je tiens tout d'abord à exprimer ma gratitude envers mes trois co-encadrants, François Charpillet, Vincent Chevrier et Stéphane Vialle. A la fin de mon DEA, je me suis adressé à Stéphane Vialle pour chercher un sujet de thèse, et il n'a pas hésité à contacter François Charpillet et Vincent Chevrier pour me proposer un sujet conjoint très intéressant couvrant à la fois les aspects multi-agents et parallélisme.

Je remercie mon directeur de thèse François Charpillet, pour son co-encadrement et son aide précieuse tout au long de ma thèse, qui m'ont permis d'orienter, développer et mettre au point mes idées, et d'améliorer ainsi la qualité de mon travail.

Mes remerciements s'adressent aussi à Vincent Chevrier pour son co-encadrement, sa disponibilité, ses critiques constructives et les nombreuses discussions que nous avons eu ensemble, qui m'ont permis de profiter de son expérience, notamment dans le domaine des agents, et de faire ainsi aboutir ce travail de thèse.

Je remercie également Stéphane Vialle pour avoir co-encadré cette thèse, pour le temps qu'il m'a consacré (merci ainsi à SUPELEC), pour m'avoir fait partager ses connaissances et son expérience dans le domaine du parallélisme, pour m'avoir appuyé, aidé et orienté jusqu'à la fin de ce travail.

Mes remerciements et ma reconnaissance s'adressent ensuite au Centre Charles Hermite (CCH) et ses dirigeants, pour le support qu'ils m'ont offert pour le bon déroulement de ma thèse, sans lequel je n'aurais pu la mener.

Je remercie aussi tous les membres de mon jury qui ont accepté de rapporter et d'évaluer ce travail, à savoir André Schaff, Rachid Alami, Abderrafiaa Koukam et Richard Washington.

Je tiens à remercier Alain Dutech pour avoir relu et critiqué ce manuscrit, et d'améliorer ainsi sa qualité.

Mes remerciements s'adressent aussi à tous les membres de l'équipe MAIA, et de l'ancienne équipe RFIA, en particulier à Jean-Paul Haton, pour leurs soutien et encouragements, ainsi qu'à tout le personnel du LORIA, en particulier notre assistante Martine Kuhlmann pour leurs accueil et disponibilité.

Un grand merci à Hend Koubaa pour son soutien et sa participation à la relecture et à la critique constructive de ce document.

De la même manière, je remercie Karima Oucherif pour son aide et le temps qu'elle m'a consacré pour relire et corriger ce manuscrit.

Je tiens aussi à remercier les membres du Centre de Recherche de Motorola à Paris qui m'ont aidé à préparer ma soutenance de thèse.

Mes vifs remerciements à mes ami(e)s du LORIA et à tous mes ami(e)s de façon générale, dont la liste est bien longue, mais qui se reconnaîtront certainement, pour leurs encouragements et support pour réussir ma thèse.

Enfin, mais loin d'être le moins important, je tiens à exprimer ma gratitude et mes remerciements envers mes parents et tous les membres de ma famille, pour leur soutien

permanent qui a participé fortement à l'aboutissement de cette thèse.

A mes parents,
à mes frères et sœurs,
à mes beaux-frères et belles-sœurs,

je dédie cette thèse.

Table des matières

Table des figures

Liste des tableaux

Introduction

1 Cadre de la thèse

Cette thèse se déroule au sein de l'équipe MAIA (MAchine Intelligente Autonome) du laboratoire LORIA / INRIA Lorraine, en collaboration avec SUPELEC, et avec le soutien du CCH (Centre Charles Hermite). Elle se situe à l'intersection des domaines des systèmes multi-agents et du parallélisme. Nous nous intéressons plus exactement à la simulation parallèle des systèmes multi-agents.

Une simulation multi-agent consiste à distribuer les données, les connaissances et le contrôle sur un ensemble d'entités appelées agents. Les agents évoluent simultanément et interagissent au fil du temps pour la réalisation d'une ou plusieurs tâches complémentaires ou antagonistes. Un système multi-agent (noté SMA) est défini par la donnée d'un environnement, dans lequel sont plongés des agents situés, c.à.d qui ont une vision locale de l'environnement via leurs capteurs et sur lequel ils agissent localement via leurs effecteurs. Une simulation multi-agent fait intervenir un ensemble d'agents situés, un modèle de comportement pour les agents, un modèle de l'environnement, un modèle de temps et un modèle de l'interaction entre l'agent et l'environnement et éventuellement entre les agents eux mêmes, pour la description de la dynamique du système.

Le modèle de l'interaction agent/agent n'est pas obligatoire puisque cette interaction peut se dérouler indirectement via l'environnement. Il suffit donc d'intégrer un mécanisme de gestion des actions simultanées des agents au niveau du modèle d'interaction agent/environnement.

Enfin, étant donné que dans un système multi-agent les agents agissent simultanément au sein de l'environnement, un modèle de simulation parallèle tirant profit de ce parallélisme inhérent des SMA paraît intéressant. La simulation parallèle va nous permettre d'offrir un cadre intuitif pour l'utilisateur, pour l'expression des comportements des agents. Elle peut aussi nous permettre de réduire le temps de simulation et/ou de traiter des problèmes de plus grandes tailles (ce que nous appelons les performances parallèles).

2 Sujet de recherche

Le problème de la plupart des simulateurs multi-agents existants est qu'ils ne reproduisent pas les incertitudes et les erreurs des capteurs des agents du système réel, ni celles de leurs effecteurs. Il en découle des difficultés à transposer les résultats obtenus

par simulation dans le monde réel.

Nous désignons par incertitudes et erreurs, ou non-fiabilité, des capteurs et des effecteurs : leur imprécision, leur non-résistance aux bruits et aux perturbations de l'environnement, et les pannes qui peuvent éventuellement se produire à leur niveau. En ce qui concerne les capteurs, l'incertitude englobe aussi l'incomplétude des informations récupérées, ne permettant pas aux agents de faire une parfaite distinction entre les différents objets du système.

Un autre problème des simulateurs multi-agents proposés dans la littérature, est que la simultanéité des activités des agents était peu ou mal exploitée. En effet, la plupart des simulateurs existants se sont limités à une implantation séquentielle de l'évolution simultanée des agents. Néanmoins, quelques travaux ont essayé d'exploiter cette simultanéité en lançant tous les agents en parallèle, sans se soucier des problèmes de synchronisation entre les agents pour la résolution de conflits ou la coopération. Malheureusement, peu de mesures de performances, d'un point de vue parallélisme, ont été communiquées à propos de ces simulateurs.

Nous nous sommes donc fixés deux objectifs pour nos travaux de recherche, qui sont les suivants :

- le premier objectif est d'étudier et de proposer un modèle de simulation d'agents situés, qui reproduit les erreurs et les incertitudes des capteurs et des effecteurs des agents. Cela va nous permettre d'obtenir des simulateurs multi-agents dont les résultats seront assez proches de ceux que nous pouvons obtenir avec le système réel,
- le second objectif est de concevoir un modèle de simulation parallèle pour des agents situés, tirant profit du parallélisme intrinsèque des SMA, afin d'avoir une expression naturelle de l'évolution des agents au cours de la simulation, ainsi que de bonnes performances parallèles.

Nous choisissons aussi d'implanter et de vérifier la validité de nos idées à travers la réalisation d'une plate-forme de simulation de robots mobiles (une première version). Nous avons choisi une telle application pour deux raisons. La première est que les applications de robotique mobile font partie des activités de recherche de notre équipe MAIA, et le développement d'un simulateur efficace, d'un point de vue parallélisme, qui reproduit fidèlement la réalité s'avère fondamental, notamment pour l'apprentissage de comportements pour les robots qui demande beaucoup de temps de calcul. La seconde raison est que ce genre d'application représente toujours un défi pour tester et valider divers mécanismes d'interaction entre les agents.

3 Choix de conception

Dans la littérature, les modèles de décision stochastiques offrent un bon formalisme pour modéliser et gérer les erreurs et les incertitudes des capteurs et des effecteurs d'un agent, lors de la résolution ou de l'optimisation d'un problème, et en particulier les Processus de Décision Markoviens Partiellement Observables (POMDP).

Nous avons proposé un modèle formel de SMA, pour la simulation multi-agent, intégrant un modèle d'interaction agent/environnement qui s'appuie sur les POMDP,

et qui reproduit les erreurs et les incertitudes des capteurs et des effecteurs des agents. Ce modèle a été implanté au travers d'un simulateur d'agents représentant des robots mobiles. Ce simulateur a été conçu de façon générique afin de pouvoir être modifié facilement et de réaliser différentes applications d'agents situés.

Nous avons aussi étudié les implantations parallèles des systèmes multi-agents qui existent dans la littérature, et nous avons proposé un nouveau modèle de simulation parallèle pour des agents situés. Celui-ci effectue un équilibrage dynamique de charge entre les processeurs, en utilisant plusieurs mécanismes de *work-pool*, et réalise un découpage des calculs basé sur les conflits entre agents. Le simulateur de robots mobiles développé utilise également le modèle parallèle que nous avons proposé, et il nous a permis de prouver son efficacité au travers des expériences menées et des résultats obtenus.

4 Plan du mémoire

Cette thèse comporte six chapitres. Dans le premier chapitre nous introduisons les agents, les modèles stochastiques proposés comme modèles de décision pour les agent, et les systèmes multi-agents, afin de familiariser les lecteurs externes à ces domaines avec ces différentes notions.

Nous commençons par donner quelques définitions pour préciser la notion d'agent, et nous indiquons celle que nous retenons pour cette thèse. Nous décrivons par la suite quelques travaux préliminaires effectués dans le domaine des agents autonomes et nous indiquons les problèmes qui ont été à l'origine des limites de ces travaux. Cela va nous permettre de motiver la présentation des modèles stochastiques, en particulier les modèles de décision de Markov, et des systèmes multi-agents dans la suite du premier chapitre. Nous définissons et introduisons la problématique des SMA, en précisant le point de vue de cette thèse. Nous présentons les différentes architectures d'agents proposées dans la littérature, et nous terminons ce chapitre par un survol des applications multi-agents réalisées, afin de montrer leur diversité et le besoin qui s'avère en simulation multi-agent.

Le chapitre 2 est la suite directe du premier. Son but est de familiariser les lecteurs avec la simulation multi-agent et le parallélisme, et de dégager les problèmes auxquels nous nous intéressons au cours de cette thèse.

Nous présentons la simulation de façon générale, et celle concernant les SMA en particulier. Nous décrivons quelques travaux qui ont été effectués dans le domaine de la simulation multi-agent et nous mettons en évidence leurs limites et les problèmes qu'ils soulèvent. Etant donné que les systèmes multi-agents présentent un parallélisme inhérent, nous présentons dans le même chapitre les environnements parallèles, ainsi que les implantations parallèles de SMA proposées dans la littérature. Nous signalons aussi les limites des performances parallèles de ces travaux.

Dans le chapitre 3, nous présentons les modèles d'interaction stochastique entre l'agent et l'environnement, et de simulation parallèle que nous proposons pour faire face aux problèmes évoqués dans le second chapitre.

Le chapitre 4 décrit une application de nos deux modèles, consistant en un simulateur de robots mobiles appelé PIOMAS. Nous présentons aussi les différents choix que nous

avons pris pour la conception finale du simulateur.

Nous décrivons dans le chapitre 5 l'implantation de notre simulateur, en passant par la description des machines utilisées et des différents choix techniques effectués.

Dans le chapitre 6, nous présentons les tests et l'évaluation du simulateur PIOMAS, aussi bien du point de vue plate-forme de simulation, que des points de vue agent et parallélisme. Nous discutons aussi l'apport de notre simulateur pour le domaine multi-agent.

Enfin, nous terminons cette thèse par une conclusion sur nos travaux et une description des différentes extensions et perspectives possibles.

Chapitre 1

Agent et systèmes multi-agents

1.1 Introduction

Notre travail se situe à l'intersection de deux domaines, à savoir les systèmes multi-agents (SMA) et le parallélisme. Nous présentons dans ce chapitre, de façon générale, les notions de base à propos des agents et des systèmes multi-agents, et dans le chapitre suivant celles concernant la simulation et le parallélisme. Ceci est dans le but de familiariser les lecteurs d'un domaine avec les différentes notions de l'autre et vice-versa. Nous nous focalisons aussi dans ce chapitre sur certains points concernant les agents et les systèmes multi-agents, que nous pensons être nécessaires pour la compréhension de nos travaux.

Les concepts "agent" et "multi-agent" sont relativement récents et jusqu'à maintenant nous n'avons pas de définition formelle de ce que sont un agent et un système multi-agent, qui soit acceptée par tout le monde [Jennings *et al.*, 1998]. Néanmoins, nous commençons par donner quelques définitions proposées concernant les agents situés, ainsi que quelques travaux qui leurs sont relatifs, afin de montrer leurs apports et leurs limites. Ces derniers vont nous permettre, par la suite, d'introduire les systèmes multi-agents ainsi que les modèles de décision stochastiques.

1.2 Agent situé

Comme nous venons de le signaler, nous présentons dans cette partie la notion d'agent situé de façon générale, c'est-à-dire l'aspect mono-agent ou encore agent isolé. Après un bref historique, nous donnons quelques définitions proposées dans la littérature pour le concept d'agent, ainsi que la définition que nous retenons au cours de cette thèse. Nous décrivons par la suite quelques travaux qui ont été effectués dans le domaine des agents, afin de montrer les raisons qui ont été à l'origine de l'apparition des systèmes multi-agents, et de l'utilisation des modèles stochastiques comme modèles de décision pour les agents isolés.

Plusieurs domaines de recherche ont participé à l'apparition de la notion d'agent. Les principaux travaux, qui ont été à l'origine de cette notion, appartiennent aux domaines

suivants [Jennings *et al.*, 1998; Chaib-draa, 1999] :

- l'intelligence artificielle (IA) (voir [Russell et Norvig, 1995] pour un historique plus complet),
- la programmation orientée-objet [Booch, 1994] et les systèmes à base d'objets concurrents [Agha *et al.*, 1993],
- les langages d'acteurs [Hewitt *et al.*, 1973; Agha, 1986],
- la conception d'interfaces homme-machine [Maes, 1994].

Mais, c'est plutôt le domaine de l'IA qui a participé le plus dans l'évolution du paradigme agent. En effet, la plupart des définitions proposées et des travaux effectués ont été établis par des chercheurs en IA. C'est pour cette raison que dans ce qui suit, nous nous basons sur ce domaine pour définir la notion d'agent.

1.2.1 Agent : définitions

L'intelligence artificielle a pour but de reproduire l'intelligence humaine au niveau des entités informatiques. Si ces entités sont capables de percevoir et d'agir au sein d'un environnement, nous pouvons les appeler **agents** [Russell et Norvig, 1995]. L'**environnement** joue un rôle particulièrement important dans la définition de la notion d'agent. En effet, ce dernier peut être vu comme un ensemble de ressources externes dont dispose l'agent et de règles, qui guident son évolution et qui subissent les conséquences de ses actions.

Russell et Norvig définissent donc un agent comme étant n'importe quelle entité qui perçoit son environnement au travers de ses capteurs et sur qui elle agit via ses effecteurs (figure 1.1). Un être humain peut donc être vu comme un agent. Un robot muni de caméras, de capteurs infrarouges, ... , et de différents moteurs pour générer des actions, peut être aussi considéré comme un agent.

Cette définition nous permet d'introduire la notion d'**interaction** entre l'agent et l'environnement. Cette interaction peut être définie comme étant l'influence mutuelle entre ces deux entités, qui est établie via les processus de perception de l'agent de son environnement et de son action sur ce dernier. L'interaction peut être directe, comme dans le cas de l'action de l'agent sur l'environnement ; c'est une modification explicite de l'état de ce dernier. Elle peut aussi être indirecte, comme dans le cas de la perception de l'environnement par l'agent ; l'environnement ne peut pas modifier lui-même l'état de l'agent, mais ce dernier peut changer d'état en fonction des percepts récupérés à partir de l'environnement.

La définition de Russell et Norvig introduit la notion d'environnement sans la définir. Elle est aussi très générale, ce qui permet d'identifier les agents dans plusieurs domaines d'application. Mais, nous pensons que les entités qui sont en interaction avec l'environnement doivent avoir certaines propriétés afin de pouvoir être appelées "agent". Nous présentons donc dans ce qui suit d'autres définitions pour l'agent basées sur les propriétés.

J. Ferber a proposé une définition pour le terme "agent" dans [Ferber, 1995]. Il le considère comme étant une entité physique ou virtuelle :

- qui est capable d'agir au sein d'un environnement E,
- qui peut communiquer directement avec d'autres agents,

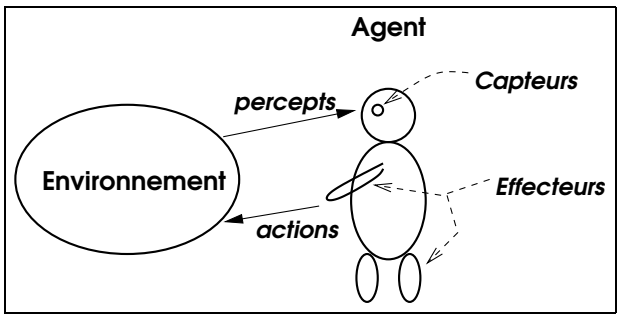

FIG. 1.1 – *La définition d'un agent au sens de Russell et Norvig*

- qui est mue par un ensemble de tendances, sous la forme d'objectifs individuels, d'une fonction de satisfaction ou de survie, qu'elle cherche à optimiser,
- qui possède des ressources propres,
- qui est capable de percevoir, mais de manière limitée, son environnement E,
- qui ne dispose que d'une représentation partielle de cet environnement et éventuellement aucune,
- qui possède des compétences et offre des services,
- qui peut éventuellement se reproduire,
- dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

Cette définition introduit des caractéristiques intéressantes pour l'agent, comme la notion d'objectif et de perception limitée ou locale de l'environnement, puisque c'est le cas dans plusieurs applications. En effet, un robot mobile a généralement un certain rayon de perception, pour observer son environnement physique. Un agent mobile dans le réseau Internet ne peut pas non plus observer tout le réseau ; il ne peut en observer qu'une partie. Nous trouvons cependant que certains points dans cette définition sont un peu plus détaillés que nécessaires (comme les notions de tendances et de la reproduction), alors que d'autres qui sont aussi importants pour la définition d'un agent ont été oubliés (décrits ci-dessous).

M. Wooldridge et R. Jennings [Wooldridge et Jennings, 1995; Jennings *et al.*, 1998; Weiss, 1999] ont défini un agent comme étant un système informatique **situé** dans un environnement, et dans lequel il est capable d'agir de façon **souple** et **autonome**, afin de satisfaire les objectifs pour lesquels il a été conçu.

Un agent est dit situé, s'il perçoit son environnement via ses capteurs, et peut en changer la configuration en agissant dessus via ses effecteurs. C'est le cas des deux exemples que nous venons de citer, à savoir l'agent situé dans un environnement physique (comme un robot mobile) ou dans le réseau Internet.

La notion d'autonomie, signifie la capacité d'action sans intervention de l'utilisateur, même dans les cas extrêmes (i.e. dans les situations critiques).

La souplesse au niveau de l'action sera acquise lorsque l'agent vérifie les propriétés

suivantes :

- la réactivité : la capacité de répondre aux événements extérieurs,
- la pro-activité : la capacité de prédire et d'anticiper les changements dans l'environnement et d'agir en fonction d'eux,
- la sociabilité : la capacité de communiquer et d'interagir avec les autres agents. Cette propriété est intéressante uniquement dans le cas où nous nous intéressons à l'aspect multi-agent.

Un agent est donc une entité physique ou virtuelle en situation dans un environnement avec lequel il interagit. Cette interaction est fondamentale dans la thèse que je défends. Nous pensons aussi que la notion de situation est importante pour la définition d'un agent. En effet, cette propriété est nécessaire pour implanter par exemple celle de la réactivité au niveau de l'agent. Cette dernière, à son tour, assure que l'agent réponde dans certaines situations imprévues rapidement et correctement. Une autre propriété importante dans cette définition est celle de l'autonomie, puisque sans cette dernière l'agent se réduit à un simple objet informatique qui possède des données et offre des méthodes.

Nous constatons donc qu'il y a plusieurs définitions de ce qu'est un agent, et que la communauté des SMA n'est pas d'accord sur une seule définition [Jennings *et al.*, 1998]. En effet, étant donné la diversité des domaines d'application des agents, ces définitions ont été généralement introduites dans le cadre d'un domaine d'application bien spécifique [Franklin et Graesser, 1996] (la robotique, les applications de réseau, ...). Néanmoins, ces définitions ne nous paraissent pas contradictoires, elles présentent plutôt certaines similitudes, comme l'action au sein d'un environnement et sa perception. Ces définitions nous paraissent aussi complémentaires, comme la notion de comportement en fonction d'un objectif dans la définition de Ferber et celle de Wooldridge et Jennings, qui est intéressante à introduire dans celle de Russell et Norvig. Les notions d'autonomie et de réactivité, introduites par Wooldridge et Jennings sont aussi importantes à inclure dans les autres définitions. C'est aussi le cas de la perception partielle de l'environnement évoquée par Ferber, qui est intéressante à introduire dans les autres définitions.

Nous pensons donc que nous pouvons faire une synthèse des différentes définitions que nous venons de présenter, afin de donner une définition "adéquate" de la notion d'agent, mais toujours en fonction du domaine d'application visé.

Notre définition pour un agent

Dans cette thèse, nous nous intéressons particulièrement à la modélisation et la simulation de l'interaction entre l'agent et son environnement, sans trop nous soucier de la modélisation de l'agent lui même, que nous voulons laisser générique. Nous reprenons donc la définition de Russell et Norvig, mais, nous ajoutons à cette définition certaines précisions concernant les propriétés que doit vérifier l'agent. Ces propriétés sont : l'autonomie, la réactivité au sens de Wooldridge et Jennings, la mise en situation, et la possession d'un ou de plusieurs objectifs. Nous changeons la définition introduite par Wooldridge et Jennings pour la notion de la mise en situation, en précisant que la perception et l'action doivent être locales pour les raisons que nous avons citées précédemment.

Pour résumer, la définition que nous adoptons dans cette thèse pour un agent est la suivante :

l'agent est une entité qui perçoit son environnement à travers ses capteurs et sur lequel il agit via ses effecteurs. Il doit aussi vérifier *les propriétés* suivantes :
- l'*autonomie*, i.e. la capacité à agir sans intervention humaine,
- la *réactivité*, i.e. la capacité à répondre aux événements extérieurs,
- la possession d'un ou de plusieurs *objectifs*,
- la *situation*, i.e. l'agent perçoit partiellement son environnement via ses capteurs, et peut en changer la configuration en agissant dessus localement via ses effecteurs.

L'*environnement* est *un ensemble de ressources externes* dont dispose l'agent, et de *règles* guidant son évolution. L'environnement subit aussi les *conséquences des actions* de l'agent et permet *leurs représentations.*

Les exemples de l'être humain et du robot cités précédemment restent toujours des exemples valides d'agents situés selon notre définition. Une entité pour le contrôle et la commande d'une chaîne de production peut être vue comme un agent aussi. Un logiciel qui s'exécute sur une machine connectée à un réseau, et qui peut répondre ou ignorer (i.e. une sorte d'autonomie) des requêtes externes communiquées via le réseau, peut être considéré comme un agent.

1.2.2 Agent : principaux travaux

Au début de l'apparition des agents, les chercheurs se sont concentrés principalement sur la conception de modèles de raisonnement, afin de les doter de capacité cognitive. Ces modèles permettent de définir comment l'agent décide de ses actions en fonction de ses percepts et de ses objectifs. En particulier, la capacité de planifier ses propres actions est donc apparue comme fondamentale [Jennings *et al.*, 1998]. Les premiers travaux dans ce domaine étaient donc très liés à ceux de la planification en IA. Le système STRIPS [Fikes et Nilsson, 1971] figure parmi les premiers systèmes de planification proposés en IA. Un système STRIPS de planification comporte au moins un modèle symbolique de l'environnement de l'agent (utilisant la logique du premier ordre), une spécification symbolique des actions de l'agent, sous la forme "pré-condition, action, effet", et un algorithme de planification en fonction de ces symboles et du but recherché. D'autres systèmes de planification ont été proposés dans le même sens, basés sur la planification hiérarchique, décomposant des tâches de haut niveau en sous tâches de niveaux moins abstraits. Nous citons comme exemples, NOAH [Sacerdoti, 1977], SIPE [Wilkins, 1984] et O-Plan [Currie et Tate, 1985]. Klein et Backstrom ont proposé un algorithme pour la planification d'une sous-classe de problèmes de contrôle séquentiel dans l'industrie [Klein et Backstrom, 1991]. Ils utilisent un formalisme qui ressemble à celui de STRIPS, basé sur la notion de vecteur d'états pour le système, de pré-condition, post-condition

et *prevail-condition*[1] d'une action.

La majorité de ces solutions était basée sur la logique du premier ordre, et consiste à chercher le meilleur plan dans l'espace de tous les plans envisageables. Malheureusement, les chercheurs utilisant ces méthodes se sont trouvés confrontés aux limites de ces algorithmes, qui ne peuvent être mis en œuvre que sur des applications simples. Ces problèmes se résument par l'explosion rapide de la taille de l'espace de recherche d'un plan, vue la complexité des tâches du monde réel. De plus les erreurs et les incertitudes des capteurs et des effecteurs des agents ne sont pas prises en compte lors de la génération d'un plan par ce type d'algorithmes, ce qui rend les plans générés parfois invalides. En effet, lorsque un agent perçoit son environnement, il n'est pas toujours sûr d'avoir perçu la bonne information, puisque des bruits ou des dysfonctionnements peuvent intervenir au niveau de ses capteurs. Concernant les effecteurs d'un agent, ils peuvent également présenter des imperfections ou des dysfonctionnements, ce qui implique que les actions d'un agent n'auront pas toujours les effets escomptés.

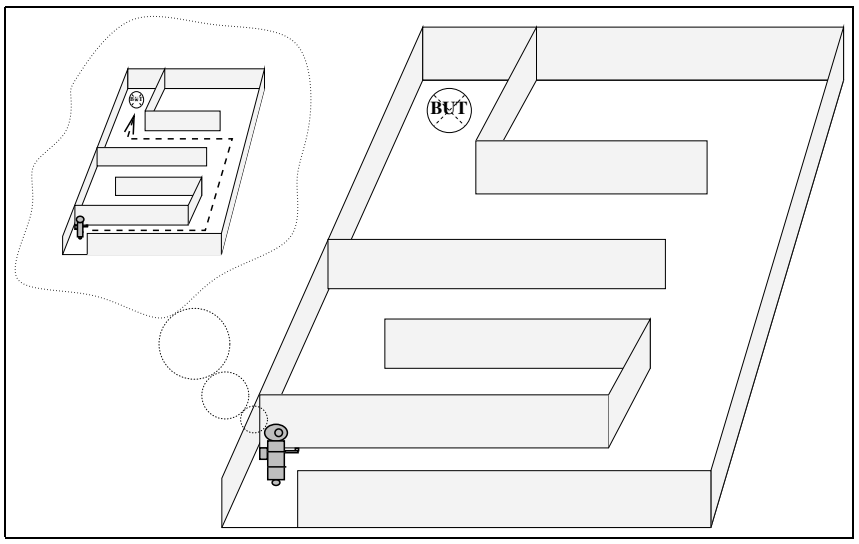

FIG. 1.2 – *Un robot qui planifie son chemin pour arriver au but*

Par exemple, dans le cas d'un robot représenté par un agent situé, ses actions ne sont pas toujours fiables, ainsi que ses percepts. En effet, le robot peut parfois glisser pendant qu'il avance ou échouer à attraper un objet, ... , comme il peut confondre une porte avec un début de couloir, ... Ceci a pour effet de rendre le plan généré au préalable invalide et de susciter un besoin fréquent de replanification au cours de l'exécution du plan. La figure 1.2 montre un robot et le plan qu'il a généré pour arriver au but spécifié. Dans la figure 1.3, nous voyons que le robot a échoué pour arriver à son but, à cause d'une mauvaise estimation de la distance parcourue, de patinage au niveau des roues et/ou de mauvaise reconnaissance de l'environnement. Dans ce cas, il doit se repérer à nouveau dans ce dernier et regénérer un nouveau plan pour atteindre le but.

Nous mettons l'accent sur la modélisation des capteurs et des effecteurs des agents, vue leur importance dans l'interaction entre l'agent et l'environnement, qui a été signalée

1. exprime ce qui doit rester vrai lors de l'exécution de l'action

par plusieurs chercheurs comme par exemple Russel et Norvig dans [Russell et Norvig, 1995], A. Drogoul et D. Fresneau dans [Drogoul et Fresneau, 1998] ou M. Ghallab[2].

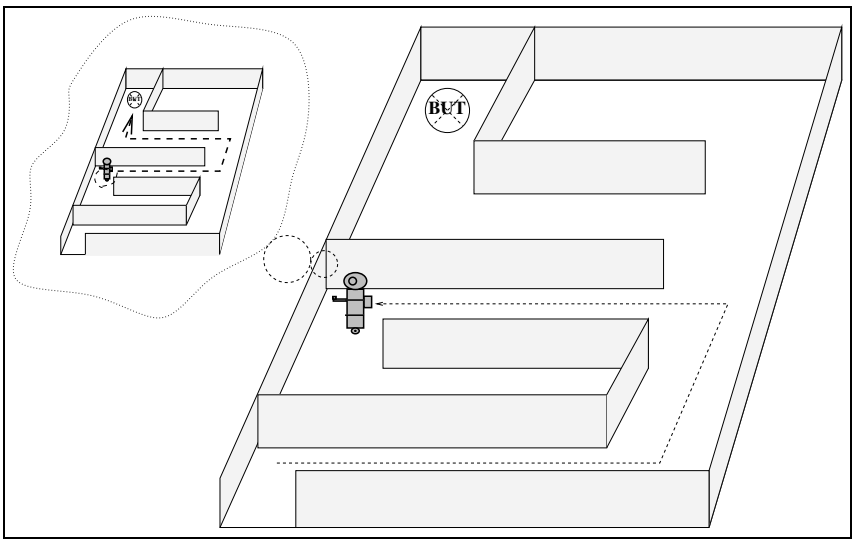

FIG. 1.3 – *Un robot qui replanifie son chemin, après un échec pour arriver au but*

Un autre inconvénient venait encore de la focalisation des recherches sur les agents isolés, dont les capacités sont aussi limitées. En effet, jusqu'aux années 80, les chercheurs ont essayé de doter des agents isolés de comportements intelligents, tels que l'apprentissage, le raisonnement, la résolution de problèmes, etc ... , [Jennings *et al.*, 1998]. Ces travaux ont abouti à des agents isolés assez complexes, et dont les comportements demandent beaucoup de temps de calcul. Ceci a eu pour conséquence la restriction de ces recherches à des exemples de petites tailles, à cause de la complexité des comportements conçus. En effet, le temps de calcul de ces comportements explose rapidement lorsque la taille du problème devient grande, ce qui est le cas des applications réelles. Ajoutons à ceci le manque en structuration et en organisation des connaissances d'un agent [Ferber, 1997]. Il fallait donc trouver des solutions à ces problèmes, afin de mieux exploiter le nouveau paradigme d'agent.

1.2.3 Conclusion

La notion d'agent situé est une nouvelle approche pour aborder la modélisation et la résolution de problèmes dans le domaine de l'intelligence artificielle. Son avantage est qu'elle offre un bon cadre pour la structuration des données, représentées dans l'environnement et dans l'agent, et d'intégrer le raisonnement au sein de l'agent. Les données contenues dans l'agent représentent les objectifs visés par exemple, et celles intégrées dans l'environnement sont les ressources externes dont dispose l'agent. La notion clé de cette structuration est l'interaction entre l'agent situé et l'environnement, qui n'est autre que la perception et l'action locales de l'agent sur son environnement.

2. Interview du 3 juin 2000 dans le journal Libération. http://www.liberation.fr/chantiers/robots2.html

Malheureusement, cette notion d'interaction était peu exploitée dans les premiers travaux d'agent. Il en découle que cette approche s'est limitée à des applications simples, à cause de la complexité des comportements intégrés au sein d'un agent pour résoudre un problème. Ceci est d'autant plus accentué par la focalisation des recherches sur des agents isolés (aspect mono-agent), et à la non prise en compte des erreurs et incertitudes des capteurs et des effecteurs de l'agent.

1.3 Les systèmes multi-agents situés

Nous présentons dans cette section les systèmes multi-agents qui permettent de faire face aux limites des capacités des agents isolés. Nous commençons par expliciter les raisons d'apparition des SMA dans ce paragraphe. Nous présentons dans le paragraphe suivant quelques définitions pour la notion de SMA, et nous précisons la définition retenue pour cette thèse. Suite à la naissance des systèmes multi-agents, différents modèles de comportement ont été proposés pour les agents, indiquant comment l'agent doit générer sa décision en fonction des buts spécifiés et de ses percepts. Nous présentons ces modèles de comportement dans la partie intitulée les architectures d'agents. Nous décrivons après quelques domaines d'application des systèmes multi-agents, afin de montrer la diversité de ces applications et le besoin en simulation multi-agent en conséquence.

Les systèmes multi-agents permettent une meilleure distribution et structuration des connaissances et du contrôle, par rapport à un système mono-agent, et de simplifier ainsi le comportement d'un agent. En effet, dans un SMA un ensemble d'agents interagissent selon des modes plus ou moins prédéfinis, afin de répondre au problème pour lequel le SMA a été conçu. Selon J. Ferber, le développement des SMA est due à quatre raisons [Ferber, 1997], que nous détaillons dans ce qui suit.

La première raison est que l'intelligence artificielle classique, qui consiste à centraliser les connaissances et le contrôle en une seule entité, ne nous permet malheureusement pas de modéliser des problèmes complexes. Cela était à l'origine de l'apparition de l'intelligence artificielle distribuée (IAD), qui distribue les connaissances et le contrôle sur un ensemble d'entités. En effet, il y a eu tout d'abord les systèmes multi-experts faisant intervenir plusieurs bases de connaissances plus ou moins coordonnées. Mais, un besoin en coopération entre les systèmes s'est fait sentir afin qu'ils réussissent bien leur(s) mission(s), et qui ne pouvait pas être facilement exprimé à l'aide du formalisme des systèmes experts.

La seconde raison provient du besoin en nouvelles techniques performantes pour la modélisation et la simulation dans le domaine des sciences du vivant. En effet, il s'est avéré qu'un "simple" ensemble d'équations différentielles ne permet pas toujours de reproduire l'évolution des écosystèmes habités, et une modélisation représentant les individus par des entités informatiques paraissait intéressante.

La robotique est la troisième raison d'apparition des SMA. Des expériences ont montré qu'un ensemble de petits robots autonomes, tant au niveau énergie que décisionnel, dotés de comportements plus ou moins primitifs sont capables de produire un comportement collectif performant, qui peut être qualifié d'"intelligent" [Ferber, 1995]. Ce comportement, connu sous le nom de comportement émergent, est aussi performant ou

meilleur que celui d'un seul robot complexe doté de toutes les capacités pour le rendre "intelligent". Mais, le problème est de faire coopérer ces robots pour arriver au comportement collectif "intelligent".

Enfin, l'évolution de l'informatique et plus particulièrement celle des systèmes distribués forment la quatrième raison. La collaboration entre des composants logiciels dans des environnements hétérogènes et distribués (réseaux d'ordinateurs et/ou machines parallèles) pour la réalisation d'un objectif commun, dépasse les fonctionnalités standard d'un programme et nécessite des capacités de coopération avec les autres programmes pour atteindre l'objectif commun.

1.3.1 SMA : définitions

Comme pour les agents situés, plusieurs définitions de système multi-agent ont été proposées et nous allons commencer par présenter celle introduite par Ferber dans [Ferber, 1995]. Un SMA est défini par la donnée :

- d'un environnement E, c'est-à-dire un espace disposant généralement d'une métrique,
- d'un ensemble d'objets O situés dans E ; c'est-à-dire qu'à chaque instant on peut déterminer les positions exactes de chaque objet dans E. Deux types d'objets sont distingués : actifs et passifs ou inertes,
- d'un ensemble d'agents A, qui sont des objets actifs ($A \subseteq O$), (c.f section 1.2.1 pour la définition de Ferber d'un agent)
- d'un ensemble de relations R définissant les liens entre les objets (agents et objets passifs),
- d'un ensemble d'opérations Op pour la manipulation des objets par les agents (perception, création, consommation, transformation, ...),
- et enfin un ensemble d'opérateurs chargés de représenter les résultats de l'application de ces opérations, y compris la réaction du monde, appelés les lois de l'univers.

Une autre définition est celle donnée par Wooldridge dans [Weiss, 1999], qui présente un SMA comme étant un ensemble d'agents en interaction afin de réaliser leurs buts ou d'accomplir leurs tâches (c.f section 1.2.1 pour la définition d'un agent de Wooldridge). Cette définition impose généralement certaines conditions [Jennings *et al.*, 1998], telles que :

- chaque agent perçoit partiellement son environnement,
- le contrôle doit être distribué,
- les données sont aussi distribuées,
- les calculs sont asynchrones.

Elle introduit aussi la notion d'interaction, qui est l'influence mutuelle entre les agents et l'environnement et/ou les agents eux mêmes, sachant qu'un agent peut aussi désigner un utilisateur externe du système. Les interactions entre agents peuvent être directes par l'intermédiaire des communications, comme elles peuvent être indirectes via l'action et la perception de l'environnement. Les interactions peuvent être mises en œuvre dans un but de :

- coopération entre les agents, lorsqu'ils ont des buts communs,

- coordination, c'est-à-dire d'organisation pour éviter les conflits et tirer le maximum de profit de leurs interactions afin de réaliser leurs buts,
- compétition, lorsque les agents ont des buts antagonistes.

Les SMA consistent donc, généralement, en un ensemble d'agents en interaction selon des modes de coopération, de concurrence ou de coexistence [Moulin et Chaib-draa, 1996; Chaib-draa, 1999], afin de répondre aux problèmes pour lesquels ils ont été conçus.

Nous remarquons alors que la définition formelle d'un système multi-agent n'existe encore pas. Les définitions proposées dans la littérature sont d'accord sur certaines caractéristiques d'un SMA, et principalement la notion d'interaction entre les agents et l'environnement, et/ou les agents eux mêmes. Bien que cette notion (i.e. l'interaction) n'était pas explicite dans la définition proposée par Ferber pour la notion de SMA, il l'a introduite plus tard dans le même document [Ferber, 1995].

Quelques formalismes génériques ont été proposés pour la spécification des systèmes multi-agents, mais malheureusement la plupart d'entre eux sont abstraits et ne peuvent pas aboutir à des implantations informatique concrètes [d'Inverno *et al.*, 1996]. En effet, ces formalismes ne proposent pas comment concevoir, implanter et vérifier le SMA qui sera obtenu à partir de la spécification établie [Hilaire *et al.*, 2000]. Néanmoins, quelques travaux sont en cours afin de proposer des formalismes complets, allant de la phase de la spécification multi-agent à celle de l'implantation, comme l'approche de Hilaire et al. qui, pour le moment, ne permet que de faire le prototypage du SMA spécifié [Hilaire *et al.*, 2000]. Cependant, ces approches sont encore limitées à des applications assez simples.

Finalement, nous pensons que la définition adéquate d'un SMA dépend encore du problème traité, et nous présentons dans le paragraphe suivant la définition que nous retenons dans nos travaux pour un système multi-agent.

Notre définition pour un SMA

Comme nous l'avons signalé lors de la présentation de la définition retenue pour un agent (section 1.2.1), nous voulons garder une certaine généricité dans nos définitions. En ce sens, la définition de Wooldridge nous paraît la meilleure [Weiss, 1999] : un *SMA* est *un ensemble d'agents ayant des buts ou des tâches, et qui interagissent pour les accomplir.* Bien entendu, la notion d'interaction entre agents, inclut les notions de *coopération*, de *coordination* et/ou de *compétition* entre ces derniers.

Cependant, nous gardons la définition de Ferber, qui nous paraît beaucoup plus proche d'un formalisme de description de SMA, afin de dériver un modèle formel pour décrire notre modèle d'interaction agent/environnement.

Le problème principal qui reste évoqué avec la définition de Wooldridge est "comment faire interagir les agents afin qu'ils remplissent leurs tâches, et atteindre les buts pour lesquels ils ont été conçus".

1.3.2 Les différentes architectures d'agents

Après avoir défini les agents et les systèmes multi-agents, nous présentons dans cette partie les différents types d'architecture proposés pour les agents, afin de comprendre

leurs modes de fonctionnement. En effet, l'architecture d'un agent permet de définir son comportement vis à vis de son environnement, ainsi que des autres agents dans le cas d'un SMA.

Agents délibératifs

C'est la première architecture d'agents qui a été proposée. Elle est basée sur l'IA symbolique, et elle permet de planifier les actions d'un agent au sein de son environnement (voir section 1.2.2).

Mais, avec l'apparition des SMA, de nouvelles idées sont apparues concernant l'architecture d'un agent. En effet, avec les limites de l'IA symbolique et plus spécialement celle basée sur la logique, mentionnées dans la section 1.2.2, les chercheurs ont essayé de définir de nouvelles architectures d'agents pour pallier les limites des architectures délibératives.

Agents réactifs

Brooks était parmi les premiers à faire et à défendre des études pour proposer une nouvelle architecture d'agents. Il est à l'origine de ce qu'on appelle l'IA réactive. Il a postulé que l'intelligence est le produit de l'interaction entre l'agent et l'environnement et qu'elle peut aussi émerger de l'interaction entre plusieurs agents avec de simples comportements [Brooks, 1986; Brooks, 1991b; Brooks, 1991a]. Il a proposé une architecture d'agent consistant en un ensemble de comportements pour l'accomplissement d'une tâche, où chaque comportement est une machine à états finis qui fait correspondre les entrées des capteurs aux actions des effecteurs. Ces comportements sont généralement organisés en couche, allant du plus simple (éviter les obstacles dans le cas d'un robot, ...) au plus sophistiqué. Si à un instant donné plusieurs actions sont générées à partir de différents comportements activés, la résolution de conflit sera effectuée par interaction entre les différentes actions (par exemple, une action inhibe l'autre). Ces architectures réactives ont montré leur efficacité dans beaucoup d'applications au travers des résultats obtenus, mais malheureusement elles présentent des limites dues aux points suivants :

- l'agent n'a pas de représentation mentale de l'environnement et il doit choisir les bonnes actions à partir des données locales uniquement, ce qui n'est pas toujours évident,
- le comportement global de l'agent ne peut pas être facilement prévu, puisqu'il émerge de l'interaction entre les comportements de ses différentes couches. Par conséquent, il n'est pas toujours possible de concevoir un comportement d'agent en fonction du but spécifié,
- ce dernier point est d'autant plus accentué par la présence d'un grand nombre de comportements au sein d'un agent.

Agents hybrides

Les architectures hybrides sont donc apparues pour faire face à ces inconvénients. En effet, des chercheurs ont combiné les deux architectures, délibérative et réactive, pour

tirer profit des deux. Il ont adopté une architecture hiérarchique comme celle des agents réactifs, généralement en trois couches ; la couche la plus basse implante les comportements réactifs, celle du milieu le comportement délibératif et la plus haute intègre le niveau social (communication, coopération, ... , avec les autres agents). Plusieurs méthodes pour la circulation des informations des capteurs et le passage du contrôle entre les couches ont été proposées. On peut citer par exemple le système InteRRap (figure 1.4) de J.P. Müller [Müller, 1996b], dont chaque couche comporte une base de connaissance fonction de son degré d'abstraction : modèle de l'environnement au niveau de la couche réactive, modèle mental au niveau de la couche de planification locale et modèle social pour la couche de coopération et de planification globale (ou encore coopérative). Les informations sensorielles alimentent le modèle de l'environnement (la couche la plus basse), qui alimente à son tour la couche qui est juste au dessus et ainsi de suite. Le passage de contrôle entre les couches s'effectue de la façon suivante : une couche de niveau supérieur ne prend le contrôle que si celle de niveau plus bas s'avère "incompétente", et elle doit en plus se servir des primitives de cette dernière pour réaliser son but.

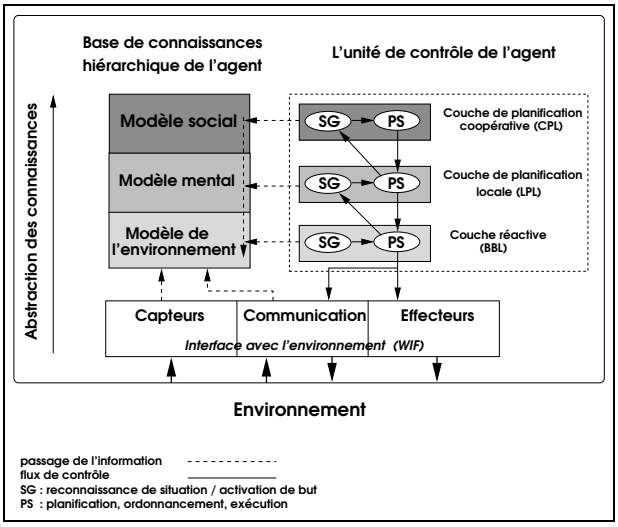

FIG. 1.4 – *L'architecture d'un agent InteRRap [Müller, 1996b]*

Agents BDI

Une autre architecture a été proposée dans le cadre du raisonnement pratique (i.e. le raisonnement pratiqué par l'être humain), connue sous le nom de BDI (*Belief, Desire, Intention*) [Bratman, 1987; Bratman *et al.*, 1988]. Elle est fondée sur des extensions de la logique, et est basée sur les états mentaux suivants :

- les croyances : les connaissances de l'agent sur l'environnement,
- les désirs : les "options" de l'agent, c'est-à-dire les différents états vers lesquels l'agent peut vouloir transiter,
- les intentions : les états vers lesquels l'agent a choisi de transiter.

Le fonctionnement d'un agent BDI consiste donc à mettre à jour ses croyances à partir des informations qu'il reçoit de son environnement. Ensuite, à déterminer les différents choix qu'il a en face, les analyser et fixer ses états buts en fonction d'eux, c'est-à-dire ses intentions. Enfin, à définir ses actions en tenant compte de ses intentions.

Malheureusement, cette architecture n'est pas adaptée pour intégrer des mécanismes d'apprentissage au sein d'un agent. Elle ne permet pas non plus de modéliser la composante sociale d'un système multi-agent, c'est-à-dire les interactions avec les autres agents [Georgeff *et al.*, 1999].

Discussion

Les agents réactifs sont les plus simples à implanter, et leur efficacité a été vérifiée dans plusieurs applications. Dans ce genre d'architecture, les résultats émergent de l'interaction entre les différents comportements, simples, des agents. Mais, bien que ces comportements soient simples, ils ne sont pas toujours évidents à concevoir. Contrairement aux agents réactifs, les agents délibératifs sont beaucoup plus complexes et plus difficiles à mettre en œuvre. Chaque agent se base sur ses propres compétences de façon isolée, et c'est ce qui a fait les limites de cette architecture. Les agents hybrides ont essayé de combiner le meilleur des deux architectures précédentes, afin de remédier à leurs inconvénients. Les agents BDI essayent de reproduire le raisonnement humain, mais ils ne sont pas convenables pour implanter des comportements d'apprentissage, et ne modélisent pas la composante sociale des SMA.

Nous pensons donc que dans un système multi-agent, les architectures réactives et hybrides représentent le meilleur compromis entre complexité d'implantation et efficacité du système global.

1.3.3 SMA : Les applications

Les systèmes multi-agents sont à l'intersection de plusieurs domaines scientifiques, tels que : l'informatique répartie, le génie logiciel, l'intelligence artificielle et la vie artificielle [Chaib-draa, 1999]. Par conséquent, ils font appel à plusieurs autres disciplines, telles que la sociologie, la psychologie sociale, les sciences cognitives, la biologie, etc ... Il en découle que les SMA sont appliqués dans divers domaines, qu'on peut classer en quatre grandes catégories, à savoir l'industrie, le commerce, le divertissement et la médecine [Jennings *et al.*, 1998].

Dans les applications industrielles, nous trouvons celles qui se sont intéressées à la fabrication, comme le système YAMS [Parunak, 1987], basé sur le protocole "Contract Net" [Smith, 1980], pour la gestion du processus de production d'une ou plusieurs usines dont les unités de production sont représentées chacune par un agent. D'autres applications se sont intéressées au contrôle de processus, puisque les contrôleurs de processus sont eux mêmes réactifs et autonomes (par exemple ARCHON [Jennings *et al.*, 1996]). Il y a des applications qui concernent la télécommunication, comme le contrôle de réseau [Schoonderwoerd *et al.*, 1997]. D'autres applications ont été développées pour le contrôle du trafic aérien, comme le système de Zeghal pour la coordination sans communication,

ni négociation entre agents représentant les engins aériens, basé sur les champs de forces [Zeghal, 1993]. Enfin, il existe des applications pour la gestion du trafic et du transport (par exemple, celle de covoiturage [Burmeister *et al.*, 1997]).

Parmi les applications commerciales, il y a celles qui se sont intéressées à la gestion de l'information (ex. [Chen et Sycara, 1998]), celles qui concernent le commerce électronique (ex. [Chavez et Maes, 1996]) et enfin celles pour la gestion des processus d'entreprise (ex. [Merz *et al.*, 1997]).

Un autre grand domaine d'application des SMA est celui du divertissement. En effet, dans le cadre des jeux (ex. [Grand et Cliff, 1998]), du théâtre interactif et de la réalité virtuelle, l'agent étant semi-autonome peut jouer un rôle important pour l'implantation de telles applications (ex. [Bates, 1994]).

Enfin, dans le domaine de la médecine, des applications à base d'agents ont été développées et d'autres en cours de développement, pour la surveillance des patients (ex. [Hayes-Roth *et al.*, 1989; Durand *et al.*, 2000]), ainsi que pour la gestion des soins aux patients (ex. [Huang *et al.*, 1995]).

1.3.4 Conclusion

Dans un SMA, un ensemble d'agents, qui sont des entités autonomes, sont en interaction entre eux (coopération, coordination, etc ...) et/ou avec un environnement partagé, afin de résoudre le problème pour lequel ils ont été conçus. Les données sont donc distribuées, ainsi que le contrôle. La notion clé des SMA est l'interaction qui peut faire émerger la solution au problème posé, à partir de comportements simples implantés au sein des agents. Ceci a été surtout constaté dans les SMA utilisant des agents réactifs ou hybrides. Malheureusement, il n'est pas toujours évident de trouver les comportements simples et adéquats pour faire émerger la solution recherchée.

Nous avons essayé de montrer, à travers le survol des différents domaines d'application des systèmes multi-agents et des exemples d'applications réalisées, l'utilité de l'approche multi-agent et la diversité de ses applications, surtout pour aborder des problèmes complexes et/ou de nature distribuée.

Malheureusement, dans certaines applications délicates, où les utilisateurs ne peuvent pas se permettre de mener des expériences sur le système réel, pour des raisons de sécurité, matérielles ou autres, les simulateurs multi-agents ont une grande importance pour tester et valider les comportements des agents, avant de les engager dans le système réel.

D'où l'intérêt de concevoir des simulateurs multi-agents qui soient aussi fidèles que possible au système réel, de telle sorte que les résultats obtenus au niveau de la simulation restent assez valides au moment du passage à l'expérimentation du système réel. Nous avons dit "assez valide", puisque la simulation fait appel à la modélisation, qui fait toujours une abstraction ou une simplification de la réalité, et par conséquent nous ne pourrons pas vraiment avoir des résultats de simulation qui coïncident exactement avec celles du système réel.

1.4 Les modèles stochastiques

Les modèles de décision stochastiques proposent un formalisme qui permet de modéliser la non-fiabilité des capteurs et des effecteurs des agents, au moment de leurs prises de décisions. Les modèles de décision de Markov, qui font partie de ces modèles stochastiques, offrent un formalisme qui permet de représenter et de quantifier les résultats des actions des agents et de leurs fonctions de perception. Nous commençons par définir ce qu'est un processus stochastique, puis nous introduisons les modèles de décision Markoviens et décrivons quelques exemples, et enfin nous discutons les apports et les limites de ces modèles pour les agents.

1.4.1 Définitions

Processus stochastique

Un processus stochastique se définit comme une famille de variables aléatoires $X(t)$, tel que $t \in T$ et $T \subset \mathbb{R}$. Cet ensemble T représente généralement le temps et est dénombrable, dans ce cas le processus est dit discret.

Suite stochastique

Soit un ensemble $S = \{s_1, s_2, \ldots, s_n\}$ dénombrable d'états et un système qui passe aléatoirement entre l'instant t et l'instant $t+1$ d'un état s_i à un état s_j de S. Soit q_t la variable aléatoire représentant l'état occupé par le système à l'instant t. On appelle suite stochastique à ensemble discret d'états, l'ensemble $\{q_1, q_2, \ldots, q_t, \ldots\}$.

Chaîne de Markov

Une chaîne de Markov du premier ordre est une suite stochastique qui vérifie à tout instant t la propriété suivante (dite propriété de Markov) :

$$Prob(q_t = s_i / q_{t-1} = s_j, q_{t-2} = s_k, \ldots, q_1 = s_l) = Prob(q_t = s_i / q_{t-1} = s_j)$$

Cela signifie que l'état du système à l'instant t dépend uniquement de son état à l'instant $t-1$.

1.4.2 Les modèles décisionnels de Markov

Les modèles décisionnels de Markov sont utilisés pour représenter et résoudre des problèmes de planification d'un agent évoluant dans un environnement incertain. Il existe plusieurs modèles dans la littérature, dépendants du problème traité et de complexités différentes. Nous nous intéressons à deux modèles dans cette partie uniquement, qui seront utiles pour la compréhension de notre modèle d'interaction stochastique.

Nous commençons par définir formellement les Processus de Décision de Markov (MDP), puis les Processus de Décision de Markov Partiellement Observables (POMDP).

Les MDP

Dans un processus de décision de Markov (MDP) [Bellman, 1957], l'agent est supposé connaître exactement son état courant, mais les résultats de ses actions sont incertains.

Un MDP est donc défini comme étant un quadruplet (S,A,T,R), tels que :

- S : ensemble fini d'états de l'environnement,
- A : ensemble fini d'actions dont dispose l'agent,
- T : fonction de transition entre états, chargée de représenter les incertitudes des effets des actions des agents, de la façon suivante :

$$T : S \times A \times S \rightarrow [0,1]$$

  $T(s,a,s')$ représente la probabilité de passer de l'état s à l'état s', suite à l'exécution de l'action a,
- R : la fonction de gain ou de pénalité permettant de guider l'agent jusqu'au but. Elle lui permet d'éviter les dangers et d'optimiser son chemin dans l'environnement. Cette fonction dépend du problème, et elle peut, par exemple, être de la forme : $R : S \rightarrow \mathbb{R}$ et signifier que l'agent sera récompensé ou sanctionné s'il occupe un état donné de l'environnement. Cette fonction peut aussi tenir compte de l'ancien état et de l'action qui a amené l'agent dans cette nouvelle situation, et prendre la forme suivante : $R : S \times A \times S \rightarrow \mathbb{R}$.

La génération d'un plan pour un agent, revient à la résolution du MDP correspondant, en fonction de l'état initial occupé par l'agent et de son état but. C'est-à-dire qu'il faut trouver l'action optimale pour chaque état qui sera occupé par l'agent en partant de l'état initial et jusqu'à arriver à l'état but, afin de maximiser l'espérance des récompenses. Le problème général des MDP appartient à la classe des problèmes P-complets [Papadimitriou et Tsitsiklis, 1987; Littman *et al.*, 1995b]. Il existe plusieurs algorithmes pour résoudre des problèmes de type MDP, mais les plus connus sont les deux suivants :

- le *Value Iteration* de Bellman même [Bellman, 1957], dont le choix de l'action optimale se base sur celle qui maximise la somme du gain immédiat, et du gain futur pondéré par les probabilités de transitions correspondantes,
- le *Policy Iteration* [Howard, 1960], qui part d'un plan initial quelconque et l'améliore de façon itérative ; c'est un algorithme de type *anytime*.

Les POMDP

Les Processus de Décision Markovien Partiellement Observables (POMDP) [Smallwood et Sondik, 1973; Laroche, 1997] sont issus du domaine de la recherche opérationnelle [Cassandra *et al.*, 1994]. Ils sont utilisés dans le cas où l'agent observe partiellement son environnement, en plus de l'incertitude des résultats de ses actions.

Un POMDP est alors défini formellement comme étant un tuple $(S,A,T,R,\mathcal{O},O)$, tels que S, A, T et R sont définis comme dans le cas d'un MDP, et $\mathcal{O}$ et O sont définis comme suit :

- $\mathcal{O}$: ensemble fini de symboles observables,

- O : fonction d'observation qui spécifie la probabilité d'observer un symbole de $\mathcal{O}$ connaissant l'état dans lequel se trouve l'agent. Elle peut être définie de deux façons différentes en fonction du problème.

  La première est : $O : S \times \mathcal{O} \rightarrow [0,1]$, qui veut dire que l'observation est relative uniquement à l'état courant de l'agent.

  La seconde est : $O : S \times A \times \mathcal{O} \rightarrow [0,1]$, qui indique que l'observation dépend à la fois de l'état et de l'action exécutée par l'agent dans cet état.

Les POMDP sont donc mieux adaptés pour modéliser une application de planification d'agent situé plus fidèlement. En effet, dans ce type d'applications les agents ont généralement des capteurs et des effecteurs non-fiables, c'est-à-dire que des incertitudes et des erreurs peuvent se produire à leurs niveaux, et les POMDP offrent un formalisme intéressant pour les modéliser.

L'inconvénient de ce modèle est que les algorithmes permettant de résoudre un tel problème de planification ont des complexités en temps de calcul importantes et ne peuvent être appliqués à des problèmes de tailles réelles. En effet, pour résoudre un tel problème, l'agent doit calculer le plan optimal qui le mène de l'état initial, qu'il observe partiellement, à l'état but. L'agent doit donc tester les actions applicables dans l'état courant, recenser tous les états vers lesquels il peut transiter et choisir la meilleure action pour arriver au but, en fonction des gains ou des punitions immédiats et futurs. De plus l'agent observe partiellement son environnement, c'est-à-dire qu'il n'a pas de certitude concernant les états qu'il occupe dans ce dernier, ce qui complique davantage sa tâche. Cette observation partielle de l'environnement se traduit par un maintien d'une distribution de probabilité sur l'"ensemble des états probables" de S (connu sous le nom de *belief state*), que l'agent pense occuper à un moment donné. A partir de cette petite idée concernant la résolution d'un POMDP, nous voyons bien que la complexité du problème en temps de calcul augmente considérablement en fonction du nombre d'états défini. Cette complexité dépend aussi de la structure choisie pour représenter le POMDP [Littman, 1994], et pour résumer, ces problèmes sont NP-difficiles dans le meilleur des cas [Laroche, 2000]. Par conséquent leur application reste restreinte à des problèmes-jouets [Littman *et al.*, 1995a; Laroche, 2000].

Nous citons ci-dessous quelques algorithmes pour résoudre des problèmes de type POMDP :

- l'algorithme *Witness* [Littman, 1994], proposé par Littman en 1994 pour le partitionnement et la recherche dans l'espace des états probables. Le calcul des distributions de probabilité sur l'ensemble des états probables s'effectue en utilisant une adaptation de l'algorithme *Value Iteration* [Bellman, 1957] proposé par Bellman en 1957,
- une variante de *Witness*, qui est un algorithme de type *anytime*, qui interrompt l'algorithme *Witness* après un certain temps ou nombre d'itérations, en supposant que la solution trouvée est satisfaisante [Littman *et al.*, 1995a],
- l'algorithme *Incremental Pruning* [Zhang et Liu, 1996], suppose que bien que l'agent n'observe que partiellement son environnement, il a toujours une idée sur la partie de ce dernier qui contient l'état courant. Ceci forme une autre classe d'algorithmes, qui permet de réduire l'espace de recherche et par conséquent la complexité du problème,

- l'algorithme proposé par R. Washington [Washington, 1996], part d'une solution initiale supposant que l'environnement est complètement observable, et essaie de construire la solution finale au fur et à mesure, à partir des observations effectuées. Cet algorithme forme une nouvelle classe qui fournit des solutions sous-optimales, mais plus rapidement que les précédentes,
- des algorithmes d'apprentissage de la solution optimale ou sous-optimale, comme ceux proposés par McCallum [McCallum, 1995] et Dutech [Dutech, 1999] se basant sur une mémoire du passé, permettant de résoudre des problèmes avec un grand nombre d'états.

1.4.3 Conclusion

Les modèles décisionnels de Markov offrent un cadre formel pour modéliser l'incertitude dans laquelle est plongé un agent situé. L'algorithmique associée permet d'aborder la planification sous incertitude de manière très efficace. Par exemple, les MDP modélisent les erreurs et incertitudes qui peuvent se produire au niveau des effecteurs d'un agent, alors que les POMDP modélisent celles qui se produisent, à la fois, au niveau des capteurs et des effecteurs de l'agent.

Malheureusement, les algorithmes proposés dans la littérature ont des complexités en temps et/ou en espace généralement élevées. De plus, la plupart de ces modèles de décision ont été proposés pour la planification d'agents isolés, et ne peuvent pas être facilement étendus à un ensemble d'agents (i.e. planification multi-agent), sans augmenter la complexité en temps de calcul de leurs algorithmes de résolution.

1.5 Bilan

Nous avons présenté dans ce chapitre quelques définitions pour le terme "agent", ainsi que la définition que nous adoptons dans cette thèse. Nous avons aussi décrit quelques travaux qui ont été effectués dans le domaine agent. Cela étant dans le but de montrer les limites des agents isolés, ainsi que de présenter les problèmes qui sont apparus à cause de la non prise en compte de la non-fiabilité des capteurs et des effecteurs des agents.

Concernant les limites des agents isolés, elles nous ont permis d'introduire les systèmes multi-agents, qui simplifient les comportements des agents, en effectuant une meilleure distribution et structuration des connaissances et du contrôle entre un ensemble d'agents.

Nous avons donc présenté quelques définitions de ce qu'est un SMA, ainsi que la définition que nous avons choisie pour nos travaux. Ensuite, nous avons présenté les différentes architectures d'agents, afin de comprendre le fonctionnement des agents. Enfin, nous avons montré la diversité des applications des systèmes multi-agents, et signalé l'intérêt de construire des simulateurs multi-agents qui soient fidèles aux systèmes réels.

Quant aux problèmes de la modélisation des erreurs et incertitudes des capteurs et des effecteurs des agents, ils nous ont permis d'introduire les modèles stochastiques, et plus particulièrement les modèles de décision de Markov.

Nous détaillons dans le chapitre suivant les aspects et les problèmes relatifs à la simulation multi-agent.

Chapitre 2

Simulation multi-agent et parallélisme

2.1 Introduction

Nous présentons dans ce chapitre la simulation de façon générale, puis la simulation multi-agent en particulier, afin de montrer son principe et ses intérêts. Nous décrivons aussi les éléments nécessaires pour effectuer une simulation multi-agent. Nous détaillons par la suite ces éléments tout en présentant et discutant au fur et à mesure différents travaux qui ont été menés à leurs niveaux. Cela va nous permettre d'introduire la problématique à laquelle nous nous intéressons.

Etant donné que les agents agissent simultanément dans un SMA, le parallélisme s'avère intéressant pour implanter leurs activités (i.e. leurs perceptions, génération de décisions et actions). Nous présentons donc dans la seconde partie de ce chapitre les environnements parallèles, de façon générale (définitions, objectifs, programmation et langages parallèles), afin de familiariser les lecteurs externes au domaine du parallélisme avec ces notions. Nous présentons après les différentes implantations parallèles de systèmes multi-agents proposées dans la littérature, tout en signalant les problèmes que présentent ces modèles parallèles, et que nous abordons au cours de cette thèse.

Nous signalons que nous utilisons le terme "simultanéité" pour exprimer le fait que deux agents ou plus agissent au même instant. Nous réservons par contre le terme "parallèle" pour décrire une exécution informatique concurrente des activités des agents.

2.2 Généralités

Dans certains cas où l'étude d'un système s'avère impossible ou inadmissible en portant des expériences sur le système même, pour différentes raisons (coût, danger, problèmes techniques, environnement virtuel, ...), nous avons besoin de faire appel à la simulation pour mener ces études.

La simulation d'un système passe tout d'abord par la création d'un modèle. Ce dernier est généralement une abstraction ou une simplification du système réel [Banks et Carson, 1984]. Il existe plusieurs modèles selon le système à étudier :

- modèle physique : une maquette qui doit être évidemment moins coûteuse et plus facilement réalisable que le système réel, comme par exemple la maquette d'un avion

dans le domaine de l'aéronautique. La maquette peut avoir la taille du système réel ou une taille réduite selon une échelle. La simulation consiste à mener certaines expériences sur la maquette, avec des risques destructifs souvent importants, et à étudier les résultats,

- modèle mathématique : qui consiste en des équations mathématiques, dont la résolution simule l'exécution du système réel. Ces équations peuvent être, par exemple, des équations différentielles et leur évaluation revient à leur résolution. Nous pouvons citer comme exemple l'équation de la chaleur, dont la résolution simule la diffusion de la chaleur dans un milieu en fonction de ses caractéristiques,
- modèle informatique : c'est un ensemble de variables évoluant en fonction du temps, pour lesquelles un ensemble d'équations mathématiques décrivant l'évolution est trop complexe à résoudre, ou à définir. C'est alors le rôle d'un programme informatique de décrire cette évolution au pas de temps spécifié, qui peut être fixe ou variable. La simulation consiste donc à lancer le programme sur ordinateur, en spécifiant les paramètres nécessaires, y compris la durée de la simulation, et à étudier les résultats qui seront fournis.

Quelque soit le modèle utilisé, il doit être validé avant de mener les expériences visées dessus et prendre en considération les résultats qui seront obtenus. Cette étape de validation s'effectue en comparant les résultats trouvés par simulation sur des exemples représentatifs, avec ceux obtenus sur le système réel, et en examinant les disparités. Un modèle sera valide si les disparités ne dépassent pas un certain seuil fixé en fonction du problème. Dans le cas échéant, il faut ajuster le modèle en changeant certains paramètres afin qu'il coïncide le plus avec la réalité.

2.3 La simulation d'agents situés

Nous avons vu qu'un modèle informatique peut être utilisé pour la simulation d'un système qui ne peut pas être représenté par une maquette, ou qui ne peut pas être décrit par de simples équations mathématiques, ou encore si ces dernières ne peuvent pas être "solutionnées". Lorsque le système à décrire devient complexe et/ou de nature distribuée, les systèmes multi-agents peuvent être alors les modèles informatiques adéquats pour le décrire (comme par exemple la coopération d'un ensemble de prédateurs pour la capture d'une proie). Il "suffit donc de dégager"[3] les entités qui seront représentées par des agents, l'environnement dans lequel elles vont évoluer et leurs modes d'interactions, à savoir l'interaction entre les agents et l'environnement et entre les agents eux mêmes (cf. définition dans la section 1.2.1). La simulation multi-agent consiste alors à faire évoluer les agents (i.e. émuler leurs comportements) en fonction du temps, et à étudier et analyser l'évolution du système global, suite aux actions produites par les agents et à leurs interactions. C. Reynolds distingue la notion de simulation basée agent, comme étant une sous classe de la simulation multi-agent limitée aux interactions locales entre les membres d'une population [Reynolds, 1999].

3. Vu l'absence de description formelle pour les SMA, ainsi que de méthodologie, cette étape n'est pas toujours évidente et nécessite parfois une certaine expérience pour la réaliser.

Dans le cadre de la simulation de systèmes complexes utilisant une modélisation multi-agent, il existe différents travaux montrant l'intérêt d'utiliser une approche multi-agent pour la simulation de tels systèmes. Parunak et al. [Parunak *et al.*, 1998] ont mentionné qu'une modélisation basée agent est convenable pour les domaines ayant un haut degré de localisation, de distribution et dominés par des décisions discrètes, alors qu'une modélisation basée sur les équations mathématiques est convenable lorsque le système est centralisé, et sa dynamique est dominée par les lois physique, plutôt que le traitement d'informations.

Les travaux sur la simulation multi-agent s'appliquent à de nombreux domaines, comme la modélisation de phénomènes sociaux, biologiques, économiques, etc ... Nous pouvons citer par exemple les travaux parus dans [Gilbert et Conte, 1995] et ceux de Goldspink [Goldspink, 2000] pour montrer les avantages des approches basées agent pour la simulation de sociétés artificielles, ainsi que les travaux de Epstein et Axtell [Epstein et Axtell, 1996] pour des exemples de simulation dans ce domaine. Le simulateur BioLand de Werner et Dyer est un exemple de simulateurs d'agents représentant des entités biologiques [Werner et Dyer, 1994]. Enfin, concernant la simulation multi-agent dans le domaine de l'économie, nous pouvons citer par exemple les simulations réalisées par Rouchier, dans lesquelles un monde artificiel a été créé où des agents s'échangent des dons [Rouchier, 1998]. Un autre système est celui réalisé par Steiglitz et Shapiro pour simuler un marché multi-agent avec la production, la consommation et l'échange négocié par enchère, dans lequel les opérations des commerçants peuvent être observées, et étudier ainsi les interactions qui provoquent des hausses importantes et soudaines des prix, qui peuvent causer le crash du marché [Steiglitz et Shapiro, 1998].

Une simulation multi-agent passe par la définition de plusieurs modèles nécessaires à son déroulement, à savoir :

- le modèle de l'espace pour la description de l'environnement,
- le modèle de temps pour décrire l'évolution de la simulation au cours du temps,
- le modèle de l'interaction agent/environnement, qui décrit la dynamique du système,
- et éventuellement le modèle de l'interaction agent/agent, décrivant aussi la dynamique du système d'un point de vue multi-agent. Ce modèle n'est pas toujours obligatoire à définir, puisque les interactions entre les agents peuvent s'effectuer implicitement via l'environnement. Dans ce cas, il suffit de décrire comment gérer les actions simultanées des agents au niveau du modèle de l'interaction agent/environnement, et surtout comment gérer les conflits entre les différentes actions.

Nous détaillons ces aspects dans ce qui suit, mais en nous focalisant sur la gestion des actions simultanées des agents, plutôt que sur la modélisation de l'interaction agent/agent.

2.3.1 Représentation de l'espace

L'espace représente l'environnement dans lequel les agents sont plongés. Il comporte un ensemble de lois, appelées lois du monde, qui régissent l'évolution des agents. Elles indiquent les conséquences des actions des agents sur l'environnement. L'environnement a aussi une certaine topologie, qui selon Drogoul et Ferber a une grande importance pour

l'organisation de l'ensemble des agents [Gilbert et Doran, 1994] [4], vu que les interactions entre les agents passent entre autre par l'environnement. Cette topologie peut être définie de plusieurs façons, en fonction des caractéristiques du problème. Par exemple, dans le simulateur de robots footballeurs SIEME, proposé par L. Magnin, le modèle d'espace est continu [Magnin, 1996]. Il consiste en un espace cartésien à deux dimensions. La plate-forme de simulation multi-agent MICE, développée par Durfee et Montgomery, utilise une représentation discrète de l'environnement, qui consiste en une grille d'emplacements, dont la taille peut être fixée par l'utilisateur [Durfee et Montgomery, 1989]. Dans le cas du simulateur MOBIDYC de V. Ginot et C. Le Page, conçu pour aider les biologistes à modéliser la dynamique des populations de poisson ou autre, le modèle d'espace est discret. Il consiste en un ensemble de cellules homogènes qui peuvent être carrées ou hexagonales selon le choix de l'utilisateur et qui seront occupées par les agents [Ginot et Le Page, 1998]. Dans l'environnement de simulation CORMAS de F. Bousquet et al., l'environnement consiste aussi en un ensemble de cellules carrées ou hexagonales de connexités différentes, où chaque cellule a parmi ses attributs un ensemble de voisins et l'ensemble des agents qui l'occupent [Bousquet *et al.*, 1998].

Nous constatons donc qu'il y a principalement deux modèles d'espace, un continu et l'autre discret. Dans le cas où les agents se déplacent dans l'environnement, et où la notion de distance est très significative pour eux, un modèle d'espace continu sera le meilleur pour représenter plus fidèlement le système réel. Ceci est le cas, par exemple, de la simulation d'un ensemble de robots mobiles.

Mais, si la notion de distance ne demande pas autant de précision pour un agent, comme dans le cas d'un ensemble d'agents mobiles dans le réseau Internet, qui se déplacent entre les sites, un modèle d'espace discret est suffisant pour réaliser les simulations.

2.3.2 Représentation du temps

Le modèle de temps définit le pas de déroulement de la simulation et par conséquent sa finesse. Selon Fianyo et al., il existe principalement deux grands types de modèles de temps [Fianyo *et al.*, 1998] :

- un modèle discret avec un pas de temps constant où la simulation se déroule par cycle. MICE [Durfee et Montgomery, 1989], MOBIDYC [Ginot et Le Page, 1998] et CORMAS [Bousquet *et al.*, 1998] sont des exemples de simulateurs à pas de temps constant fixé par l'utilisateur.
- un modèle événementiel discret où la simulation passe d'un évènement à l'autre. Il faut donc calculer à l'avance les instants où il y aura des évènements importants à simuler, et faire évoluer le système d'un instant à l'autre. Par exemple, dans le cas du simulateur SIEME [Magnin, 1996], un évènement peut correspondre à l'instant d'interception d'une balle par un robot footballeur, en fonction de leurs vitesses et de leurs trajectoires. Le simulateur SWARM est un autre exemple ; dans SWARM, un agent peut être composé d'un ensemble d'agents, de façon hiérarchique, gérés par une horloge et qu'on appelle *swarm* [Minar *et al.*, 1996]. L'interpréteur ABLE, proposé

4. Le chapitre 6 de [Gilbert et Doran, 1994].

par Wavish et Graham pour la simulation d'agents situés, a aussi un modèle de temps événementiel ; il détermine le premier évènement parmi ceux prédits à se déclencher et passe directement à la simulation de l'instant correspondant à cet évènement [Graham et Wavish, 1991].

En conclusion, dans le cas où nous pouvons prévoir tous les événements qui vont se produire, nous pouvons utiliser un modèle de temps événementiel pour la simulation, afin de réduire son temps d'exécution global. Mais, dans le cas contraire un modèle de temps discret avec un pas constant reste toujours applicable, et c'est celui qui prend le plus de temps d'exécution. Dans ce dernier cas, nous devons bien choisir le pas de temps afin de ne pas "oublier" certains événements intéressants à simuler.

Nous remarquons donc que le modèle de temps influe bien sur le nombre moyen d'événements simulés. En effet, une simulation avec un petit pas de temps, peut simuler plus d'événements qu'une ayant un grand pas de temps. Par conséquent, le modèle de temps influe en quelque sorte sur la finesse de la simulation, en termes de nombre moyen d'événements simulés. Ce dernier est parfois fonction du modèle d'espace choisi, si les agents sont mobiles dans l'environnement. En effet, dans ce cas les agents auront des vitesses de mouvement qui sont fonction de la topologie de l'environnement et du modèle de temps, puisque la variation de temps δt et le déplacement δx sont reliés par la vitesse v de l'agent, de la façon suivante : $\delta x = v.\delta t$. Dans le cas de la simulation d'un ensemble de robots mobiles, par exemple, les instants de conflits spatiaux entre les agents et leur nombre dépendent du modèle d'espace utilisé ; dans le cas où il est continu nous pouvons détecter plus de conflits entre les agents que dans le cas où il est discret, puisque l'avancement des agents est beaucoup plus fin dans le premier cas que dans le second.

2.3.3 Modélisation de la relation agent/environnement

Nous décrivons dans cette partie les travaux qui se sont intéressés à la modélisation de la relation entre l'agent et l'environnement dans le cadre de la simulation d'agents situés, ou qui peuvent y servir. Nous classons ces travaux en trois classes, en fonction des hypothèses faites, implicitement ou explicitement, par leurs concepteurs concernant la fiabilité de leurs capteurs et de leurs effecteurs :

- la première classe est celle où les capteurs et effecteurs des agents sont supposés fiables,
- la seconde est celle qui modélise partiellement la non-fiabilité des capteurs ou des effecteurs,
- la troisième modélise, à la fois, la non-fiabilité des capteurs et des effecteurs des agents.

Nous détaillons dans ce qui suit ces trois classes de travaux.

Capteurs et effecteurs des agents supposés fiables

Beaucoup de travaux se sont intéressés à la modélisation de l'interaction entre l'agent et l'environnement, et ont signalé l'importance que joue l'environnement dans l'évolution des agents. Par exemple le modèle "Influences and Reaction" de Ferber et Müller [Ferber et Müller, 1996], a mis l'accent sur l'importance de la séparation des rôles de

l'agent, de celui de l'environnement. En effet, dans ce modèle il y a une claire différence entre les influences des agents, qui sont les décisions d'action prises par ces derniers, et la réaction de l'environnement à ces influences, c'est-à-dire la mise en œuvre de ces actions. Cette séparation permet de gérer les interactions complexes entre les agents situés, ainsi que la simultanéité de leurs actions, qui sont modélisées au niveau de la réaction de l'environnement en combinant les influences des différents agents. Cette idée a été utilisée par O. Labbani, pour la formalisation des couplages entre les robots et l'environnement, dans la méthodologie Cirta qu'elle propose pour la conception de comportements collectifs émergents de robots miniatures [Labbani-Igbida, 1998]. Le modèle "Influences and Reaction" a été aussi utilisé par R. Canal pour dériver son modèle "Propositions/Actions", qui permet de modéliser la réaction en chaîne qui peut se produire suite à la simultanéité des actions des agents (i.e. leurs influences).

D'autres modèles ont également souligné l'importance de l'environnement dans un système multi-agent, comme le simulateur SIEME [Magnin, 1996], qui fait également une séparation entre l'action de l'agent et la réaction de l'environnement. Fianyo et al. ont aussi donné de l'importance à l'environnement, en proposant un modèle multi-agent pour la simulation de l'exploitation des périmètres irrigués, utilisant des agents ayant des comportements de type automate stochastique et communiquant via l'environnement [Fianyo *et al.*, 1997]. C'est aussi le cas du simulateur multi-agent, proposé par Montesello et al. pour des robots footballeurs, qui utilise une coordination implicite entre les agents via l'environnement [Montesello *et al.*, 1998]. Le modèle proposé au sein de notre équipe MAIA, par C. Bourjot et al., pour la coordination d'agents artificiels par le biais de l'environnement s'inspire des sociétés d'araignées. Il est basé sur la notion de stigmergie, qui est une suite de stimuli-réponses entre chaque agent et l'environnement [Bourjot *et al.*, 1999].

Des plates-formes ont été proposées pour l'étude des comportements des agents dans des environnements dynamiques, en séparant le rôle de l'agent de celui de l'environnement. Nous citons par exemple l'outil MICE [Durfee et Montgomery, 1989], et la plate-forme TileWorld permettant la simulation d'un ensemble d'agents évoluant dans un environnement (une grille à deux dimensions) dynamique, où des objets peuvent apparaître et disparaître. Elle a été introduite en premier lieu par Pollack et Ringuette [Pollack et Ringuette, 1990], puis reprise par Philips et Bresina sous le nom de "NASA TileWorld" [Philips et Bresina, 1991].

Malheureusement, la plupart des modèles et simulateurs multi-agents proposés ne peuvent pas exprimer explicitement les erreurs qui peuvent se produire au niveau des capteurs et des effecteurs des agents, ni leurs incertitudes. Cela explique pourquoi ils ont des difficultés à maintenir valides les résultats obtenus par simulation en passant à l'expérimentation sur le système réel. Cette disparité entre les résultats de simulation et ceux du système réel a été observée dans plusieurs cas, par exemple avec les robots Khepera et leur simulateur, qui sont les plus utilisés par les chercheurs menant des expériences nécessitant plusieurs tests. Ceci a poussé les créateurs de ces robots Khepera, non seulement à améliorer leur simulateur, mais aussi à simplifier leurs robots pour obtenir des résultats presque identiques avec les deux plates-formes (réelle et de simulation) [Bryson *et al.*, 2000].

Néanmoins, le modèle "Influences and Reaction" [Ferber et Müller, 1996] semble être

un bon modèle de départ pour représenter la non-fiabilité des capteurs des agents et de ses effecteurs, puisqu'il fait une claire séparation entre l'action de l'agent et la réaction de l'environnement.

Intégration partielle de la non-fiabilité des capteurs ou des effecteurs

Quelques travaux ont essayé de traiter partiellement les incertitudes. Par exemple, le simulateur multi-agent proposé par Stone et Veloso pour des robots footballeurs, gère les incertitudes des capteurs en raisonnant sur les observations passées [Stone et Veloso, 1997]. J.P. Müller a proposé un modèle markovien d'interaction entre des agents, dont l'architecture est de type InteRRap (voir section 1.3.2), pour prédire leurs performances pour la résolution d'un problème. Il utilise un modèle de prédiction des comportements des agents et un modèle Markovien pour représenter les transitions des agents entre les différents états du système, afin de prédire le comportement collectif des robots en interaction [Müller, 1996a; Müller, 1996b]. Le modèle MMDP, proposé par Boutilier pour la planification, l'apprentissage et la coordination multi-agent est basé sur des processus de décision de Markov (MDP) ; le système est constitué d'un ensemble d'états et de transitions entre les états est représentée par une chaîne de Markov qui dépend des actions jointes des agents [Boutilier, 1996].

Le premier modèle gère uniquement les incertitudes des capteurs des agents. Les deux derniers traitent seulement les incertitudes et les erreurs des effecteurs. En outre, ils n'ont été proposé que pour la prédiction des performances ou la planification, et non pas pour la simulation ; c'est-à-dire pas pour en tenir compte au moment de la simulation et obtenir ainsi des résultats proches de la réalité.

Modélisation de la non-fiabilité des capteurs et des effecteurs

La méthode proposée par Miglino et al. pour l'apprentissage d'un comportement de navigation d'un robot de type Khepera, en évitant les obstacles, passe par l'échantillonnage de l'environnement réel à travers les capteurs et les effecteurs du robot, afin d'obtenir un modèle précis de la dynamique robot/environnement. Cette méthode passe par la suite, par l'introduction de certains bruits au niveau du simulateur, représentant les incertitudes des capteurs et des effecteurs du robot, au moment de l'apprentissage du comportement représenté par un réseau de neurones reliant les entrées des capteurs aux sorties des effecteurs et évoluant au cours du temps par un algorithme génétique. Les conclusions qui ont été tirées des expériences menées sur ce simulateur sont que les disparités entre les résultats de la simulation et du système réel ne sont pas très importantes et qu'elles peuvent disparaître en continuant l'apprentissage pour quelques cycles sur le système réel [Miglino *et al.*, 1996].

Ce simulateur est dédié à l'apprentissage de comportements de robot par un réseau de neurones. L'introduction des bruits des capteurs et des effecteurs se résume par des perturbations au niveau des entrées des neurones au moment de l'apprentissage. Mais, ces bruits ne sont pas quantifiés par rapport à ceux du système réel, c'est-à-dire qu'ils ne reflètent pas avec précision les erreurs et incertitudes des capteurs et effecteurs de l'agent réel. En plus, le simulateur développé ne peut pas être utilisé pour tester et

valider la robustesse au bruit d'un comportement conçu pour un agent, si celui-ci n'est pas implanté par un réseau de neurones.

La plate-forme MASS proposée récemment (Juillet-Août 2000) par Vincent, Horling et Lesser pour la simulation multi-agent, a essayé aussi de rendre les conditions d'expérimentation plus réelles. Le but était de faciliter la modélisation et les tests rapides de différents comportements adaptatifs d'agents, au sens des mécanismes de coordination, de détection, de diagnostic et de réparation dans un environnement dynamique. En effet, le système proposé comporte une entité nommée simulateur et des agents qui sont découplés et qui communiquent via ce dernier. Les agents ont des méthodes qui ont chacune trois variables (coût, qualité et durée), décrites chacune par des distributions de probabilité discrètes. Le simulateur utilise ces distributions et la disponibilité des ressources nécessaires pour baisser (ou améliorer si besoin est) les résultats de ces méthodes. Il simule aussi les problèmes de routage des messages entre agents dans un réseau, en les retardant ou en les perdant. Le simulateur affecte aussi la perception qu'a un agent de son environnement, en définissant la notion de "vue subjective" pour ce dernier et de "vue objective" pour lui même. Il permet donc à un agent, en fonction des capteurs utilisés, de percevoir son environnement avec un certain degré de précision [Vincent *et al.*, 2000; Horling *et al.*, 2000].

Vincent et al. ont développé une application pour la gestion intelligente d'une maison, i.e. une application d'"habitat intelligent". MASS simule les tâches requises par les occupants de la maison, pendant une journée. Neuf agents ont été conçus pour remplir ces tâches ; un agent représentant le lave-vaisselle, un pour le chauffe-eau, un pour le climatiseur, ... Ils ont testé les performances de différents protocoles de coordination entre agents, afin d'accomplir les différentes tâches, en intégrant des imprévus, comme par exemple un manque d'eau chaude de la part de l'agent chauffe-eau, lorsque le lave-vaisselle en a besoin.

Cette plate-forme nous semble être la plus intéressante et la plus proche de nos travaux, puisqu'elle traite en quelque sorte le même problème que nous avons soulevé, qui est celui de la modélisation de la non-fiabilité des capteurs et effecteurs des agents pour la simulation. Le but de cette modélisation est d'avoir des résultats de simulation qui restent assez valides en passant au système réel. Mais, malheureusement cette plate-forme reste silencieuse sur un aspect important du modèle, qui concerne la méthode utilisée pour quantifier les incertitudes des capteurs et des effecteurs des agents du système réel, afin que les conditions d'expérimentation soient vraiment plus réelles. Elle reste aussi ambiguë sur certains points d'implantation et de reproduction de ces incertitudes au niveau du simulateur.

Conclusion

Pour résumer le problème de la modélisation de la relation entre l'agent et l'environnement, nous avons constaté que la plupart des travaux de simulation multi-agent n'ont pas essayé de modéliser la non-fiabilité des capteurs des agents, ni de celle de leurs effecteurs. Cela explique pourquoi les utilisateurs de ces simulateurs ont des difficultés pour garder valides les résultats obtenus par simulation lors du passage à l'expérimentation sur le système réel.

2.3.4 Gestion de la simultanéité des actions des agents

Un des problèmes qui peuvent se poser lors de la simulation d'un ensemble d'agents situés évoluant dans un environnement, est la simultanéité de leurs actions : comment simuler la génération et la mise en œuvre des actions, de deux agents ou plus, produites simultanément? Quelques travaux se sont intéressés à ces aspects, alors que d'autres les ont passés sous silence, bien qu'ils soient très importants pour la simulation d'agents situés agissant simultanément. Il y a donc eu quelques réponses, et des solutions se traduisant par des implantations séquentielles ou parallèles ont été proposées.

La solution séquentielle la plus courante consiste à faire un mécanisme de scrutation qui interroge les agents sur leurs actions (influences) à chaque instant de la simulation. Par la suite, une fois toutes les influences rassemblées, il détecte les conflits entre les actions et les résout selon une politique prédéfinie (ordre de priorité, rapports de force, lois physiques, ...), dépendant du domaine. Enfin, il met en œuvre les différentes actions des agents, suite à cette résolution de conflits et en tenant compte des lois du monde. Cette méthode se base en réalité sur la séparation entre l'action de l'agent et la réaction de l'environnement. Nous pouvons citer comme exemple le simulateur MICE [Durfee et Montgomery, 1989], qui utilise des prédicats spécifiés par l'utilisateur lui indiquant quoi faire en cas de conflit. Le modèle "Influences and Reaction" [Ferber et Müller, 1996] a été également proposé dans ce sens, c'est-à-dire pour gérer la simultanéité des actions des agents ; il intègre les lois du monde au niveau de la réaction de l'environnement aux différentes influences des agents.

Concernant les implantations parallèles des actions simultanées, il n'y a pas eu beaucoup de travaux pour profiter de ce parallélisme implicite au niveau des systèmes multi-agents. Nous allons présenter les modèles implantés ainsi que quelques exemples de travaux dans la section qui suit.

2.4 Les environnements parallèles

Comme déjà signalé, cette partie permet de présenter les notions de base du parallélisme, afin de familiariser les lecteurs externes au domaine du parallélisme avec ces différentes notions.

Nous décrivons par la suite quelques travaux concernant les implantations parallèles de systèmes multi-agents proposées dans la littérature, que nous classons selon certaines catégories, tout en montrant les problèmes qu'ils présentent et auxquels nous nous intéressons.

2.4.1 Définitions du parallélisme

Selon Almasi et Gottlieb, le parallélisme peut être vu comme une grande collection d'éléments de calcul, qui peuvent communiquer et coopérer pour résoudre rapidement de grands problèmes. Ils ajoutent aussi que cette définition fait apparaître plus de questions que n'apporte de réponses, puisque la facilité de programmation et la fiabilité du programme construit restent des facteurs importants sans réponse dans cette définition

[Almasi et Gottlieb, 1989]. Nous trouvons pratiquement cette même définition du parallélisme dans [Carriero et Gelernter, 1990], ainsi que dans [Wolper, 2000]. Mais Wolper a rajouté une petite précision indiquant que les programmes ou les sous-programmes, s'exécutant simultanément, peuvent interagir pour partager des ressources, en plus ou à la place de la réalisation d'un but commun.

Une définition plus générale a été introduite par Cosnard et Trystram, disant que "le traitement parallèle est une forme de traitement de l'information qui permet, en cours d'exécution, l'exploitation d'événements concurrents. Ces événements se situent à plusieurs niveaux : au niveau du programme, de la procédure, de l'instruction ou à l'intérieur d'une instruction" [Cosnard et Trystram, 1993]. Cette définition nous permet d'introduire certains objectifs du parallélisme, décrits dans le paragraphe suivant.

2.4.2 Objectifs du parallélisme

La programmation parallèle a plusieurs intérêts, en fonction du domaine d'application visé. Nous distinguons principalement deux grands domaines :

- le calcul parallèle, comme le calcul scientifique où les temps de calculs peuvent être très importants. Beaucoup de simulations, ainsi que de problèmes d'IA en sont des exemples.
- les systèmes distribués (et parallèles), comme les réseaux et les systèmes d'exploitation multi-utilisateurs.

Dans le domaine du calcul parallèle, le parallélisme permet principalement l'amélioration des performances des applications, en termes de temps d'exécution et de taille de problèmes résolus [Almasi et Gottlieb, 1989; Carriero et Gelernter, 1990; Wolper, 2000]. Ces derniers se traduisent respectivement par l'accélération des traitements (i.e. le *speed-up*), et la résolution de problèmes de plus grandes tailles (i.e. le *size-up*). Cela peut avoir lieu surtout dans le cas où les exécutions vont se dérouler sur plusieurs processeurs, que ce soit une machine parallèle et/ou un réseau de stations de travail.

Dans le domaine des systèmes distribués, le parallélisme a d'autres objectifs, qui sont aussi importants que ceux que nous venons de citer, mais pas souvent évoqués. Dans le cas des systèmes multi-utilisateurs, par exemple, le parallélisme permet le partage dans le temps d'une machine entre plusieurs utilisateurs, afin d'augmenter le confort et le taux d'utilisation des machines. Ceci passe par l'intégration de mécanismes favorisant le parallélisme au niveau du système d'exploitation, comme l'installation de files d'attentes de processus, de différentes priorités, qui se partagent un processeur dans le temps. Le parallélisme offre aussi des méthodes d'organisation des programmes, ainsi que des mécanismes de communication et de synchronisation entre ces derniers. Ceci peut être exploité pour permettre à des machines distantes inter-connectées de coopérer ou de coordonner leurs actions pour l'élaboration de certaines tâches, comme le partage de périphériques [Cosnard et Trystram, 1993; Vialle et Cornu, 1997].

En ce qui concerne les systèmes multi-agents, le parallélisme peut avoir un autre apport en plus de celui de performances parallèles. En effet, puisque les agents agissent simultanément au sein de l'environnement, il serait intéressant d'avoir un modèle de programmation parallèle permettant d'exprimer les comportements des agents de façon

concurrente. Ce modèle de programmation va permettre aux utilisateurs des simulateurs multi-agents d'avoir un formalisme assez implicite, qui leur épargne de chercher des implantations purement séquentielles, avec toutes les contraintes qu'elles imposent, pour des comportements qui sont implicitement parallèles, et de simplifier ainsi le développement des SMA.

2.4.3 La programmation parallèle

Nous partageons l'avis de Gamboa Dos Santos [Gamboa Dos Santos, 1995] concernant la distinction de trois niveaux d'abstraction dans le domaine du parallélisme : le niveau architecture, le niveau modèle d'exécution et celui du modèle de programmation. Une application parallèle doit assurer une composition cohérente de ces trois niveaux d'abstraction.

Architecture

Le niveau architecture signifie la structure de la machine physique (le *hardware*). Celle-ci englobe [Gamboa Dos Santos, 1995] :

- les techniques d'interconnexion des processeurs,
- l'organisation de la mémoire, au sens distribuée ou partagée,
- les mécanismes de synchronisation offerts par la machine,
- la topologie reliant les unités de calcul et celles de contrôle (anneau, matrice, graphe quelconque, etc ...).

Modèle d'exécution

Le niveau modèle d'exécution représente la façon dont les programmes s'exécutent du point de vue du programmeur. Nous présentons ci-dessous la liste des modèles d'exécution, basée sur la classification de Flynn, destinée à l'origine à classer les types des ordinateurs [Flynn, 1972] :

- SISD (*Single Instruction stream Single Data stream*) : dans ce modèle, l'exécution se fait avec un seul flot d'instructions et un seul flot de données. C'est le cas des machines classiques mono-processeur exécutant des programmes séquentiels isolés (i.e. qui ne communiquent pas),
- MISD (*Multiple Instruction stream Single Data stream*) : dans ce cas, l'exécution se déroule avec plusieurs flots d'instructions, mais un seul flot de données. C'est-à-dire que plusieurs opérations successives manipulent la même donnée simultanément. Ce modèle d'exécution n'a pas d'existence réelle, vu qu'une même donnée doit être traitée par plusieurs instructions simultanément, mais certains l'assimilent à une exécution en mode *pipeline* [Almasi et Gottlieb, 1989; Germain-Renaud et Sansonnet, 1991],
- SIMD (*Single Instruction stream Multiple Data stream*) : ce modèle décrit une exécution ayant un seul flot d'instructions et plusieurs flots de données. C'est-à-dire que la même instruction est appliquée à plusieurs données simultanément. C'est une modélisation de l'exécution sur certaines machines massivement parallèles, dans lesquelles

beaucoup d'unités de calcul exécutent en même temps la même instruction sur des données différentes,

- MIMD (*Multiple Instruction stream Multiple Data stream*) : ce modèle décrit une exécution avec plusieurs flots d'instructions et plusieurs flots de données. C'est la modélisation de plusieurs programmes, traitant différentes données et qui collaborent ensemble. Cette notion de collaboration est primordiale, car sans communication entre les programmes, le modèle se simplifie en des SISD complètement disjoints. Ce modèle d'exécution est également adopté par les nouvelles machines massivement parallèles.

Un nouveau modèle d'exécution est apparu, appelé SPMD (*Single Program stream Multiple Data stream*) [Karp, 1987], qui est une extension du modèle SIMD. Le modèle SPMD représente le cas où un même programme ou bloc d'instructions est appliqué à des données différentes. La différence essentielle entre le SPMD et le SIMD réside dans les synchronisations. Dans le cas du SIMD, la synchronisation se fait au niveau des instructions, alors que pour le modèle SPMD la synchronisation ne s'effectue que lorsque des données doivent être échangées entre différentes entités de calcul. Notons également, que la différence entre les modèles SPMD et MIMD est faible et dépend uniquement des programmes qui s'exécutent en parallèle. Dans le modèle SPMD, nous avons plusieurs exécutions du même code, alors que dans le MIMD plusieurs codes différents peuvent s'exécuter en parallèle.

Modèle de programmation

C'est le niveau le plus abstrait, permettant au concepteur de spécifier la source de parallélisme de son application. On distingue principalement trois sources de parallélisme [Germain-Renaud et Sansonnet, 1991] :

- le parallélisme de données,
- le parallélisme de contrôle,
- le parallélisme de flux.

A chaque source de parallélisme, nous associons un modèle de programmation portant le même nom, que nous détaillons dans ce qui suit.

Dans le premier modèle, les données guident le parallélisme grâce aux structures de données répétitives de l'application. Par exemple, les vecteurs et les matrices dans certaines applications scientifiques représentent des structures de données répétitives. Ce modèle a l'avantage de garder des structures de contrôle simples et proches de celles du modèle de programmation séquentiel. Généralement, ce modèle de parallélisme de données s'exécute efficacement sur une machine ayant un modèle d'exécution du type SIMD. Mais, des travaux ont été effectués afin de pouvoir l'exécuter sur des machines du type MIMD [Blelloch *et al.*, 1994].

Le second modèle, qui est celui du parallélisme de contrôle, utilise des processus ou tâches communicants pour assurer le parallélisme. La communication entre processus peut s'effectuer soit par envois de messages, soit à travers une mémoire partagée où les données sont accessibles à tous les processus. A ce jour, c'est encore le programmeur qui s'occupe de cet échange de données entre les processus, et de leurs synchronisations

au sein de l'application parallèle. C'est le principe du modèle "Fork and Join" [Conway, 1963; Almasi et Gottlieb, 1989].

Le troisième modèle, celui du parallélisme de flux, ou encore le *pipeline*, concerne les applications comportant une suite d'opérations traitant un flux de données, en cascade (une sorte de travail à la chaîne). Les applications de ce type peuvent s'exécuter sur des machines basées sur le modèle MISD, si elles existent, ou encore sur des machines du type MIMD [Germain-Renaud et Sansonnet, 1991].

2.4.4 Les langages parallèles

Le problème de la programmation parallèle est qu'elle fait appel à des techniques relativement difficiles à mettre en œuvre et nécessitant une certaine expérience pour les maîtriser. Ceci rend le temps de développement d'une application parallèle plus important que celui de l'application séquentielle, ce qui impose au programme parallèle d'avoir une durée de vie beaucoup plus grande en guise de compensation [Vialle *et al.*, 2000]. Afin de faciliter et d'accélérer le développement de programmes parallèles, en choisissant le langage adapté au problème, nous classons les langages parallèles en quatre grandes classes :

- les langages séquentiels à parallélisation automatique, comme certaines extensions parallèles de FORTRAN [Toomey *et al.*, 1988; Schneider, 1990],
- les langages séquentiels, enrichis de directives de compilation comme OpenMP [Dagum et Menon, 1998],
- les langages séquentiels enrichis par des bibliothèques de fonctions parallèles, comme le langage C avec la bibliothèque de passage de messages MPI [Pacheco, 1997], ou enrichis avec des processus légers, tels que les *P-threads* [Nichols *et al.*, 1996],
- les langages avec des modèles de programmation parallèles, menant à des parallélisations plus naturelles des applications, comme les langages à acteurs [Agha, 1986].

La première classe est la plus attirante pour les utilisateurs, vu que c'est le compilateur qui se charge de la parallélisation du code écrit en séquentiel. Malheureusement, cette technique est restée limitée aux applications de calcul scientifique régulier (comme celles de calcul sur les vecteurs et les matrices) puisqu'elle ne peut paralléliser efficacement quasiment que des boucles imbriquées régulières [Feautrier, 1995]. Des travaux se poursuivent dans ce domaine, mais il faut encore fournir beaucoup d'efforts pour obtenir des compilations parallèles efficaces [Feautrier, 1996].

La seconde classe de langage consiste à faire ajouter par le programmeur des directives de compilation dans le code source séquentiel, au niveau des parties que le compilateur doit paralléliser. Son avantage est de permettre une parallélisation incrémentale des applications. Mais, malheureusement, les outils utilisés ne sont simples d'emploi et efficaces que sur des algorithmes réguliers, sinon ils ne dispensent pas d'une recherche en algorithmique parallèle [Vialle et Dedu, 2000].

La troisième classe est plus ancienne et plus connue que la seconde, pour la programmation des applications ayant un modèle d'exécution de type MIMD. Elle est aussi la plus utilisée aujourd'hui, puisque les utilisateurs connaissant généralement le langage séquentiel n'ont qu'à apprendre les nouvelles fonctions de la bibliothèque parallèle qu'ils

vont utiliser. Le principe de programmation parallèle des langages de cette classe est donc facile à comprendre, mais l'utilisateur doit fournir beaucoup d'efforts pour effectuer son implantation parallèle, comme la création des processus, leurs synchronisations, leurs communications, ...

La quatrième classe de langages parallèles a essayé de trouver un compromis, en minimisant les efforts que le programmeur doit fournir, et en augmentant le degré de parallélisme qu'un compilateur peut détecter. Une des solutions possibles est celle de concevoir un langage ayant un modèle de calcul[5] parallèle qui mène le programmeur à concevoir son code de façon parallèle. Par exemple, les langages à Acteurs [Agha, 1986; Houck et Agha, 1992; Kim et Agha, 1993] font partie de ce type de langages. Ces langages étaient destinés à des applications d'intelligence artificielle symboliques [Hewitt *et al.*, 1973], mais ils n'ont pas eu beaucoup de succès, vu qu'ils avaient des modèles de calcul assez compliqués au niveau de l'utilisation. D'autres langages similaires basés sur les objets concurrents ont été proposés pour différentes applications [Yonezawa *et al.*, 1986; Nierstrasz, 1987; Yasugi *et al.*, 1992; Konstantas, 1993]. Mais, ces langages n'ont pas eu beaucoup de succès non plus, à cause de leurs modèles de calculs qui ne sont pas évidents à utiliser aussi et/ou des difficultés de les implanter efficacement sur des machines parallèles.

Etant donnés ces problèmes, un autre type de langages faisant partie de la quatrième classe citée, a été introduit par S. Vialle et al. [Vialle *et al.*, 1998; Vialle *et al.*, 2000]. Cette famille de langages est appelée ParCeL (*Parallel Cellular Languages*). Elle a été conçue dans le but de fournir des modèles de calcul qui peuvent être efficacement implantés sur les machines parallèles modernes, ainsi que pour offrir des langages parallèles ayant des syntaxes et des sémantiques proches de celles des langages usuels. Les langages ParCeL sont basés sur des cellules qui sont créées dynamiquement et qui s'exécutent concurremment. Ces langages sont destinés à certaines applications d'IA, telles que les systèmes multi-agents et les réseaux de neurones, qui sont difficiles à paralléliser efficacement et qui peuvent nécessiter beaucoup de temps de calcul. Cette restriction des applications cibles a été effectuée car la conception d'un langage parallèle efficace et facile à utiliser n'est pas évidente. Cette conception est d'autant plus difficile que l'on veut construire des langages qui couvrent une multitude d'applications.

La première version ParCeL-1 avait son propre modèle de calcul, syntaxe, sémantique et compilateur. Mais, malheureusement, ce langage n'a pas eu la chance d'être exploité, puisque il était optimisé pour les architectures à base de *transputer*, et le temps d'avoir une version complète de ParCeL-1 (comprenant le compilateur), ces architectures ont été dépassées et de nouvelles architectures plus attractives ont vu le jour. Par conséquent, le développement de ParCeL-3 s'est restreint à la machine virtuelle du modèle de calcul et à une bibliothèque de fonctions. Un programme ParCeL-3 est donc un programme C, dont la fonction principale (*main*) et les fonctions pour la gestion des cellules sont fournies par la bibliothèque. ParCeL-3 a un modèle d'exécution de type SPMD, et est dédié aux nouvelles machines parallèles à mémoire partagée (physiquement ou virtuellement). Nous l'avons utilisé pour développer notre simulateur multi-agent sur des machines

5. Un modèle de calcul décrit juste comment le calcul sera effectué [Almasi et Gottlieb, 1989]

parallèles qui ont toujours du succès[6].

2.5 Les implantations parallèles de SMA

Quelques travaux ont été menés dans le but de profiter du parallélisme inhérent des systèmes multi-agents, c'est-à-dire de la simultanéité de leurs activités, dans le but d'offrir à l'utilisateur un cadre pour l'expression de l'évolution des agents plus intuitif et plus facile à définir. Un autre but de l'implantation parallèle, qui est primordial et très bien connu dans la communauté des chercheurs en parallélisme, mais rarement souligné par les chercheurs en multi-agent utilisant le parallélisme est l'efficacité des implantations. Cette efficacité se traduit par de bonnes accélérations du programme, i.e. le *speed-up*, et/ou des traitements de problèmes de plus grandes tailles, i.e. le *size-up*. En effet, un système multi-agent peut avoir plusieurs agents, qui sont plus ou moins complexes, avec leurs mécanismes de perceptions, de génération de décisions, de planification, ... , nécessitant des ressources de calcul importantes. Une implantation parallèle des simulateurs multi-agents sur un ensemble de processeurs, permet donc de réduire le temps de calcul et/ou d'effectuer des simulations à plus grandes échelles. Ces deux buts ne sont pas exclusifs, mais il n'est pas toujours évident de trouver la bonne implantation parallèle pour le SMA, qui permet d'assurer un bon compromis entre la simplicité d'expression de la simultanéité des comportements des agents et l'efficacité de l'implantation.

Nous classons les travaux d'implantations parallèles de systèmes multi-agents en trois catégories, qui sont les suivantes :

- les parallélisations basées agent,
- les parallélisations de certains composants d'un agent,
- la répartition spatiale de l'environnement du SMA sur un ensemble de processeurs.

Nous présentons dans ce qui suit ces différentes catégories.

2.5.1 Parallélisation basée agent

La solution la plus évidente et la plus utilisée est celle d'attribuer la gestion de chaque agent à un processus, étant donné le parallélisme inhérent des systèmes multi-agents. Les agents évoluent donc en parallèle et se synchronisent à chaque fois que c'est nécessaire (pour communiquer, coordonner leurs actions, résoudre un conflit, ...), puis se relancent à nouveau en parallèle.

Nous citons comme exemple de simulateurs multi-agents parallèles utilisant ce principe, BioLand proposé par Werner et Dyer, qui est massivement parallèle, et qui simule l'évolution simultanée d'agents biologiques dans un environnement. Le comportement de chaque agent est contrôlé par un réseau de neurones, qui est capable d'apprendre et d'évoluer dans le temps [Werner et Dyer, 1994]. Un autre exemple est l'environnement de modélisation et d'exploration de systèmes distribués StarLogo, qui simule des milliers de tortues dont les actions s'exécutent en parallèle [Resnick, 1994]. Dans le simulateur SWARM, les entités *swarm* s'exécutent aussi en parallèle [Minar *et al.*, 1996].

6. Voir la justification de ce choix dans la section 5.3.

D'autres systèmes multi-agents ont aussi des implantations parallèles, comme le système à règles de production multi-agent proposé par Ishida. Dans ce dernier, chaque système à règles de production est représenté par un agent, qui peut se diviser en un ensemble d'agents agissant en parallèle lorsqu'il devient surchargé, pour des contraintes de temps réel, et qui à leur tour peuvent être fusionnés en un seul agent quand la charge redevient faible [Ishida, 1995].

Une autre implantation parallèle de simulateurs multi-agents a été proposée par Vincent et al. à travers leur système de simulation multi-agent [Vincent *et al.*, 1998]. Ce système simule les effets des attaques directes dans un réseau d'ordinateurs sur les résultats des actions des agents, ainsi que leurs recouvrements. Ce système comporte une entité nommée simulateur et des agents qui s'exécutent en parallèle comme étant des processus indépendants communicant via le simulateur. Ce dernier synchronise l'exécution des agents, en leur envoyant des tops d'horloge à chaque pas de simulation pour commencer les calculs, et en attendant leurs messages de terminaison pour envoyer le prochain top.

Un exemple de travaux basé sur la logique des processus est la plate-forme de simulation événementielle proposée par Theodoropoulos et Logan dans le but de permettre une meilleure distribution des agents et de leur environnement sur un ensemble de processeurs. En effet, ils ont modélisé chaque agent par un processus logique. L'environnement est partitionné en fonction de l'application (il n'y a pas de règles spécifiées), et chaque partition est aussi modélisée par un processus logique. Les agents sont supposés ne communiquer entre eux et avec l'environnement que par envoi de messages. Les canaux de communications sont alors gérés par des processus logiques, dont le nombre varie dynamiquement au cours de la simulation. Cette variation est fonction des régions de l'environnement qui sont concernées par les évènements produits par les agents (appelées sphères d'influence) et du trafic des canaux, afin d'alléger les charges des processus de communications [Theodoropoulos et Logan, 1999]. Les auteurs ont mentionné qu'ils ont suivi cette voie, basée sur la logique des processus, puisque le peu de travaux qui ont été effectués pour des implantations parallèles de système multi-agent ont révélé des performances relativement pauvres. Mais, malheureusement, nous ne pouvons pas juger l'efficacité de leur plate-forme, puisque leurs travaux restent théoriques sans implantation.

La plupart de ces travaux, affectant un processus à un agent, ne font pas mention de leurs performances du point de vue du parallélisme. Nous redoutons que leurs performances soient médiocres, à cause de la simultanéité même des actions des agents. En effet, des actions simultanées et conflictuelles ne peuvent pas être traitées en parallèles ; les processus doivent se synchroniser pour résoudre le conflit et mettre en œuvre les différentes actions. Ces synchronisations impliquent forcément des pertes de temps. De plus, un problème de granularité apparaît, puisqu'il existe des applications où un agent ne fait pas beaucoup de calculs entre deux actions successives. Les processeurs doivent donc supporter beaucoup de petits processus et perdent leur temps dans leurs gestions et leurs synchronisations. Finalement, nous ne pouvons pas juger de l'efficacité et de l'adéquation de ces travaux pour la simulation parallèle de systèmes multi-agents.

2.5.2 Parallélisation de certains composants d'un agent

Une autre approche pour la parallélisation de systèmes multi-agents a été présentée par Gagné et al. à travers leur simulateur d'un avion militaire AURORA, utilisant les Acteurs[7] qui s'exécutent en parallèle. En effet, ils ont attribué la simulation de chaque agent à un acteur, où les agents représentent l'équipage de l'avion (le pilote, le navigateur tactique, ...) qui sont en communication. Chaque agent a sa propre base de connaissance et son système expert. Mais, la particularité de ce système est que les agents ont des capteurs communs, d'où l'attribution de la simulation de chaque capteur à un acteur aussi. Ils ont introduit d'autres acteurs pour la simulation de l'environnement, comme les engins ennemis, l'horloge système, ... Tous les acteurs peuvent s'exécuter en parallèle sur un réseau de processeurs hétérogènes et communiquent dynamiquement à travers le passage de message [Gagné *et al.*, 1993]. Nous aurions pu classer cette approche parmi celles basées agent, mais puisque nous nous intéressons aux agents situés qui perçoivent leur environnement via leurs capteurs (i.e. les capteurs font partie de l'agent), nous l'avons classée dans cette nouvelle catégorie qui essaye de paralléliser différents composants d'un même agent.

Une autre implantation parallèle a été proposée par Balch et al. pour leur simulateur de navigation de robots mobiles. Ils se sont intéressés à l'implantation de l'ensemble des comportements de la couche réactive d'un robot (comme l'évitement d'un obstacle, la poursuite d'un but, l'avancement d'un robot, ...), où chaque comportement génère un vecteur de mouvement, et le déplacement final du robot est la somme de ces vecteurs. La gestion de chaque comportement est attribuée à un *thread* (processus léger). Tous les *threads* s'exécutent en parallèle et communiquent via la mémoire partagée. L'ordonnancement de ces *threads* est effectué dynamiquement, en tenant compte de la durée estimée de façon heuristique pour l'exécution de chaque comportement. De bonnes performances, en termes de *speed-up*, ont été trouvées sur deux machines multiprocesseurs à mémoire partagée, mais sur un exemple ne comprenant qu'un seul robot [Balch *et al.*, 1994]. Nous ne savons pas si ces performances se maintiennent lorsque le nombre de robots augmente.

Les travaux de Veit et Richter vont un peu dans ce sens. En effet, ils ont proposé une plate-forme, nommée FTA, pour le développement de systèmes distribués, comme ceux faisant appel à des agents de granularité moyenne en interaction. Cette plate-forme consiste en un ensemble d'agents, appelés acteurs, qui s'exécutent en parallèle sur une grappe (*cluster*) de machines inter-connectées fonctionnant sous Linux. Chaque acteur représente une entité du système visé, comme les capteurs d'un robot, son système de décision (appelé le "cerveau"), ses effecteurs, ... , qui communiquent en utilisant les primitives de passage de messages de Linux. Le concepteur du système distribué peut après un certain nombre d'exécutions encore diviser un acteur en plusieurs et les répartir sur d'autres machines pour équilibrer les charges [Veit et Richter, 1998]. Une application de contrôle d'un robot autonome a été développée sur cette plate-forme par Richter et al. pour valider cette approche, où le robot a pour tâche de reconstituer le plan d'une chambre partitionnée avec des murs [Richter *et al.*, 2000]. Malheureusement, aucune performance du point de vue parallélisme n'a été communiquée, et il est donc encore

7. Voir section 2.4.4.

une fois impossible de juger l'efficacité de la parallélisation.

Cette méthode de parallélisation est encore plus fine que l'approche basée agent, d'un point de vue granularité, et par conséquent tous les problèmes que nous avons cités pour l'approche basée agent restent vrais pour cette approche, et ils peuvent même être plus importants.

Cela dit, il n'y a pas eu beaucoup d'implantations parallèles de simulateurs multi-agents allant dans ce sens, et les seuls qui ont rapporté de bonnes performances, Balch et al., ont utilisé un seul agent (robot) dans leurs expériences [Balch *et al.*, 1994].

2.5.3 La répartition spatiale

Une autre approche d'implantation parallèle des systèmes multi-agents a été proposée, notamment pour la simulation, partitionnant l'environnement en un ensemble de mailles de tailles égales et attribuant la gestion de chaque maille, y compris les agents qui l'occupent, à un processus. Des expériences on été menées dans le cadre du DEA de L. Kipp, effectué en coopération entre le LORIA et SUPELEC, permettant la communication des processus soit à travers l'échange de messages, soit par partage de mémoire, et les résultats qui ont été obtenus ont montré que la mémoire partagée est plus efficace, du point de vue du parallélisme, que l'approche par échange de messages, et est également plus facile à implanter [Kipp, 1997]. Un simulateur parallèle et optimisé a été développé par L. Bouton en se basant sur ces derniers résultats, dans le cadre d'un autre DEA effectué en coopération entre le LORIA et SUPELEC, et des expériences ont pu être menées avec succès sur des environnements de très grandes tailles et comprenant un grand nombre d'agents [Bouton, 1998].

Les expériences ont porté sur des systèmes de larges populations, afin de vérifier plus facilement que les agents sont équitablement répartis entre les différentes mailles, et d'assurer ainsi l'équilibre de charge entre les processeurs. Les opérations de mise à jour de l'environnement se sont effectuées efficacement en parallèle, et un petit surcoût dû au parallélisme est apparu suite aux résolutions de conflits dans les régions se situant au niveau des frontières des partitions.

Un nouveau simulateur optimisé, avec plus de fonctionnalités, est en cours de conception et d'implantation à SUPELEC dans le cadre de la thèse de E. Dedu [Dedu, 2000]. Il est encore trop tôt pour disposer de mesures de performances précises.

Nous pensons que, malheureusement, cette solution n'est efficace que lorsque les conflits ne se produisent que loin des frontières des partitions, et que les agents sont équitablement répartis sur l'environnement, ce qui n'est pas le cas de toutes les applications multi-agents que nous voulons simuler. En effet, cette méthode peut provoquer un déséquilibre de charge entre les processeurs, si par exemple un grand nombre d'agents se concentre dans une seule partition et les autres restent presque inoccupées. Elle peut encore, dans le cas où il y a beaucoup de changement de partitions de la part des agents, engendrer un surcoût de parallélisme très important dû aux multiples synchronisations et communications non-prévues entre les processus, pour la migration des agents. Vu que nous nous intéressons aux agents situés, le surcoût dû aux synchronisations et communications entre processus peut être encore plus élevé, si les agents ont des rayons de perception importants et se placent près des frontières, puisqu'ils auront besoin de

percevoir fréquemment les partitions voisines.

2.5.4 Conclusion

Nous avons présenté quelques travaux d'implantations parallèles de systèmes multi-agents, que nous avons classés en trois catégories, qui sont :
- les implantations parallèles basées agent,
- les implantations parallèles de certains composants d'un agent,
- la répartition spatiale de l'environnement.

Comme nous l'avons déjà fait remarquer, peu de mesures de performances, en termes de *speed-up* et/ou de *size-up* ont été communiquées, et par conséquent nous ne pouvons pas juger l'adéquation de ces implantations parallèles par rapport au parallélisme inhérent des SMA.

Dagaeff et Chantemargue ont discuté aussi dans [Dagaeff et Chantemargue, 1997], l'adéquation de l'implantation parallèle, affectant un agent à un processus ou utilisant une répartition spatiale, par rapport au parallélisme inhérent des systèmes multi-agents, mais dans un autre sens un peu plus philosophique. En effet, ils se sont demandés si l'implantation parallèle permet de reproduire fidèlement la simultanéité réelle des actions des agents, et si les résultats seront valides du point de vue multi-agent. Ce problème a été soulevé puisque chaque agent peut avoir sa propre horloge qui cadence son évolution dans l'environnement, et le fait de forcer les processus qui les implantent à se synchroniser à des instants imposés par le partage des données de l'environnement, peut fausser les résultats qui seront obtenus. Cela reste valable même si chaque agent est représenté par un processus. Les auteurs n'ont pas proposé de solution à ce problème. Nous sommes en partie d'accord sur ce point, toutefois en se situant dans le cadre d'une simulation, nous devons toujours faire une certaine abstraction de la réalité. De plus, nous pensons qu'avec une implantation séquentielle, nous ne pouvons pas résoudre ce problème non plus, et même que nous pouvons avoir des résultats encore plus biaisés. Il est encore plus difficile d'implanter correctement en séquentiel la simultanéité et l'asynchronisme des actions des agents.

2.6 Bilan

Nous avons présenté dans ce chapitre la simulation d'agents situés, les environnements parallèles de façon générale, ainsi que quelques travaux concernant les implantations parallèles des systèmes multi-agents.

Nous avons commencé par décrire les différents éléments nécessaires pour une simulation d'agents situés. Nous avons alors décrits les différents modèles d'espace et de temps existants, ainsi que quelques travaux concernant la modélisation de l'interaction entre l'agent et l'environnement. Nous avons signalé que la plupart de ces travaux ne modélisent pas la non-fiabilité des capteurs et des effecteurs des agents lors de la modélisation de l'interaction entre l'agent et l'environnement. Cela explique pourquoi les résultats des simulations multi-agents ne coïncident pas souvent avec celles des expérimentations du système réel.

Comme les agents évoluent simultanément dans un SMA, nous avons étudié les implantations parallèles existantes des systèmes multi-agents, afin de profiter des avantages du parallélisme pour la simulation multi-agent. Ces avantages se résument par la réduction du temps de simulation ou par le traitement de problèmes de plus grandes tailles (appelés performances parallèles). Un autre avantage est celui de fournir à l'utilisateur un formalisme intuitif pour l'expression des comportements des agents.

Nous avons noté que peu de performances parallèles ont été communiquées pour les implantations parallèles de SMA proposées dans la littérature. Nous avons mentionné que nous redoutons que leurs performances soient médiocres, et que par conséquent nous ne pouvons pas juger l'adéquation de ces modèles parallèles pour la simulation multi-agent.

Nous présentons dans le chapitre suivant les solutions que nous proposons à tous les problèmes que nous venons de soulever.

Chapitre 3

Proposition de modèles d'interaction stochastique et de simulation parallèle

3.1 Problématique

Nous nous plaçons dans le cadre de la simulation d'un ensemble d'agents situés dans un environnement. Rappelons qu'un agent situé a une activité sensori-motrice qui lui permet d'évoluer dans l'environnement dans lequel il est immergé. Ainsi, il perçoit cet environnement via ses capteurs et agit sur celui-ci via ses effecteurs.

Une difficulté importante dans la mise au point de tels agents est liée à l'incertitude et l'incomplétude des informations délivrées par leurs capteurs, mais aussi à la difficulté de modéliser les effets des actions qu'ils entreprennent. Un simulateur se doit de reproduire ces difficultés si l'on veut pouvoir immerger dans un monde réel des agents qui auraient été mis au point dans ce simulateur. Ceci est le premier problème auquel nous nous sommes intéressé durant cette thèse.

Nous décrivons donc dans la section suivante le modèle stochastique d'interaction entre l'agent et l'environnement, que nous proposons dans le but cité ci-dessus. Nous présentons le fondement théorique de notre modèle, une description formelle et textuelle des différents composants du modèle, ainsi que le principe de la simulation multi-agent utilisant notre modèle d'interaction.

Le second problème que nous avons étudié concerne l'implantation parallèle efficace, en termes de performances parallèles, de simulateurs multi-agents exploitant le parallélisme inhérent des SMA, c'est-à-dire la simultanéité des activités des agents (i.e. la perception, la décision et l'action). Nous présentons donc dans la seconde partie de ce chapitre, notre modèle de simulation parallèle à équilibrage dynamique de charge entre les processeurs. Nous reprenons les différents modèles de simulation parallèle proposés dans la littérature (cf. section 2.5), et nous montrons leur inadéquation, en général, pour la simulation d'agents situés. Nous décrivons par la suite les détails de notre modèle de simulation parallèle, aussi bien formellement que textuellement.

3.2 Un modèle stochastique d'interaction agent / environnement

Comme nous venons de le signaler, nous nous intéressons à la modélisation des fonctions sensori-motrices des agents. Celle-ci doit tenir compte de la non-fiabilité des capteurs et des effecteurs, des incertitudes et des erreurs qui peuvent se produire au niveau matériel, logiciel (la phase d'interprétation des signaux pour les capteurs et le manque de précision numérique pour la commande des effecteurs), ainsi que les perturbations provenant du bruit de l'environnement.

Les erreurs qui se produisent au niveau des capteurs peuvent engendrer des confusions entre les percepts. Par exemple, dans le cas d'un robot mobile, il se peut qu'il confonde une porte avec un début de couloir, dans le cas d'une mauvaise reconnaissance de l'environnement. Un autre exemple est celui d'un ensemble d'agents mobiles et communicants sur le réseau Internet. L'environnement représente donc le réseau et la perception de l'agent revient à la réception d'un message acheminé via l'environnement. Dans ce cas, des messages à destination d'un ou plusieurs agents peuvent avoir des délais de transmission variables, comme ils peuvent être altérés, voire perdus au moment de leurs réceptions.

Concernant les effecteurs d'un agent, l'action entreprise n'a pas toujours le résultat escompté, à cause des erreurs et des incertitudes qui peuvent se produire au niveau de ses effecteurs. En reprenant l'exemple des agents mobiles communicants sur Internet, l'altération d'un message ou sa perte peut se produire au moment de son émission par l'agent, c'est-à-dire au moment de l'action. Pour le cas du robot mobile, il peut par exemple glisser de temps à autre en avançant dans l'environnement ou légèrement se désorienter.

Ces erreurs et incertitudes se produisent donc au moment de l'interaction de l'agent avec son environnement, tant au niveau de la perception que de l'action. Par conséquent, nous devons en tenir compte au moment de la modélisation de l'interaction entre l'agent et l'environnement, afin de développer des comportements robustes à de telles erreurs de perception ou d'action, inhérentes au monde réel, au niveau du simulateur même.

3.2.1 Approche

Nous avons développé un modèle formel d'agent inspiré de la théorie du contrôle [Dean et Wellman, 1991]. Ce modèle assimile un agent à un contrôleur couplé à un environnement (i.e. en interaction), dont la dynamique est régie par un système d'équations : $S(t+1) = f(S(t), Infl_a(t))$, où $S(t)$ représente l'état de l'environnement à l'instant t, et $Infl_a(t)$ est la sortie ou plutôt l'influence de l'agent a sur l'environnement. La perception de l'agent pour son environnement à l'instant t est régie par l'équation : $Perc_a(t) = P_a(S(t))$ (figure 3.1).

L'objectif du simulateur que nous proposons est d'évaluer le comportement d'un agent sans pour autant l'immerger dans un environnement réel. Cela peut être réalisé en modélisant le fait que les actions des agents n'ont pas toujours les effets escomptés, au niveau de la dynamique du système. Quant à la fonction de perception, elle doit aussi

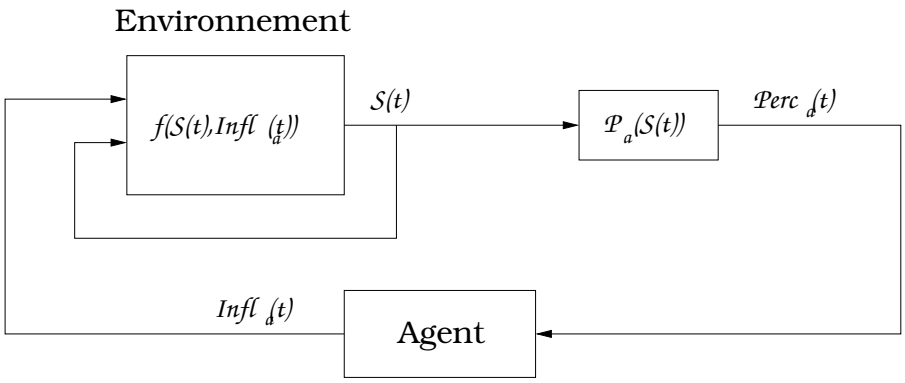

FIG. 3.1 – *Description de la dynamique agent/environnement selon la théorie du contrôle*

reproduire l'imprécision, l'incomplétude et les erreurs du système réel.

Notre simulateur exploite une modélisation stochastique de la **dynamique** de l'environnement :

$$f(S(t),Infl_a(t)) = P(S(t),S(t+1)|Infl_a(t)), \tag{3.1}$$

et une modélisation probabiliste de la perception de l'agent :

$$P_a(S(t)) = P(Perc_a(t)|S(t)). \tag{3.2}$$

L'équation 3.1 indique que l'environnement passe d'un état à l'instant t à un autre à l'instant $t+1$, en fonction de l'influence de l'agent, selon une certaine probabilité (i.e. ce passage n'est pas déterministe). De même, l'équation 3.2 mentionne qu'à l'instant t, l'agent perçoit l'observation $Perc_a(t)$ à partir de son environnement avec une certaine probabilité.

Cette approche va nous permettre de construire des simulateurs qui peuvent servir de plates-formes d'expérimentation et/ou d'apprentissage de divers comportements pour les agents.

Afin de tenir compte de l'aspect multi-agent, nous avons prévu un opérateur de combinaison des influences des agents $\prod_{a \in Agents}(Infl_a(t))$, qui détecte et résout les conflits entre les agents. Cet opérateur est introduit au niveau des équations de la dynamique de l'environnement de la façon suivante : $S(t+1) = f(S(t), \prod_{a \in Agents}(Infl_a(t)))$.

3.2.2 Fondement théorique

D'après ce qui précède, les erreurs et les incertitudes des capteurs et des effecteurs d'un agent doivent être intégrées au niveau du modèle de l'interaction entre l'agent et l'environnement.

Nous avons vu dans la partie bibliographie, que les processus de décision de Markov partiellement observables (POMDP) [Smallwood et Sondik, 1973], proposés pour la planification d'un agent (voir section 1.4.2), offrent un bon formalisme pour modéliser

et quantifier la non-fiabilité des capteurs et des effecteurs d'un agent. Ces modèles sont déjà utilisés au sein de notre équipe pour différentes applications.

Etant donné que les erreurs et incertitudes des effecteurs d'un agent se produisent au moment de la mise en œuvre de son action, il serait intéressant de les intégrer au niveau de la réaction de l'environnement. De plus, il faut que notre modèle d'interaction puisse gérer les actions simultanées des différents agents du système, dès lors qu'on se situe dans un cadre multi-agent.

Nous avons trouvé que le modèle "Influences and Reaction" [Ferber et Müller, 1996] (voir section 2.3.3) permet d'intégrer tous les paramètres que nous venons de citer ci-dessus. En effet, nous avons vu que ce modèle établit une claire distinction entre les influences qui sont produites par les comportements d'un agent et la réaction de l'environnement, représentant la mise en œuvre de ces influences.

Nous nous sommes donc inspirés de ces deux modèles (i.e. POMDP et "Influences and Reaction") pour créer un modèle stochastique d'interaction entre l'agent et l'environnement pour la simulation, permettant de rendre compte de la non-fiabilité des capteurs et des effecteurs des agents [Bouzid *et al.*, 2001].

Nous avons adapté le formalisme de description des systèmes multi-agents, introduit par Ferber dans [Ferber, 1997], pour présenter formellement notre modèle d'interaction et la simulation multi-agent qui l'utilise. Ce formalisme est le suivant (nous introduisons les domaines de définition des fonctions plus tard) :

- Un $SMA =< Ags, W, \prod >$, tels que (voir figure 3.2) :
  - Ags est l'ensemble des agents : $Ags = \{Ag;\ Ag\ est\ un\ agent\}$,
  - W est l'environnement, défini par $W =< E, A, S, T_a >$,
  - $\prod$ est l'opérateur de combinaison des influences simultanées des agents.
- Un agent Ag est défini par : $Ag =< P_a, Capts_a, Percept_a, F_a, Infl_a, \xi_a >$, avec :
  - P_a est la fonction de perception de l'agent a,
  - $Capts_a$ est l'ensemble des types de capteurs utilisés par l'agent a,
  - $Percept_a$ est l'ensemble des stimuli et sensations de l'agent a,
  - F_a est la fonction de comportement de l'agent a, intégrant implicitement son ou ses but(s),
  - $Infl_a$ est la fonction de génération d'influence de l'agent a, qui intègre aussi son ou ses but(s),
  - ξ_a est l'ensemble des états internes de l'agent a.
- Les différentes caractéristiques de l'environnement W sont :
  - E est l'espace, comprenant des objets, dans lequel évolue les agents,
  - $A = \bigcup_{a \in Ags} A_a$ est l'ensemble des influences qui peuvent être générées par tous les agents,
  - S est l'ensemble des états de l'environnement,
  - T_a est la loi d'évolution du monde suite à l'exécution d'une influence de l'agent a.

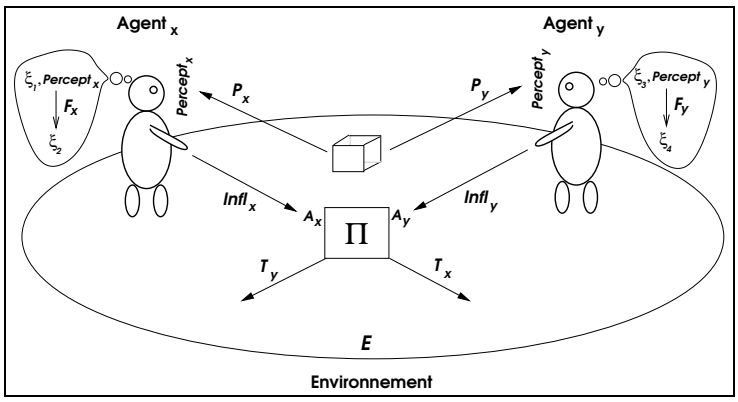

FIG. 3.2 – *Le formalisme utilisé pour de description d'un SMA*

Nous commençons par décrire le modèle de l'environnement W que nous avons choisi pour tout système multi-agent à simuler, ainsi que le modèle de l'agent Ag. Par la suite, nous détaillons notre modèle stochastique de l'interaction entre chaque agent et l'environnement que nous proposons, et enfin le principe de la simulation multi-agent utilisant ce modèle.

3.2.3 Le modèle de l'environnement

L'environnement représente l'espace physique ou virtuel du système à modéliser. Afin d'offrir un modèle d'environnement assez générique, nous avons prévu un ensemble d'objets placés dans cet espace. Nous distinguons deux types d'objets :

- les inactifs, notés Obj_i : ce sont les objets qui ne génèrent aucune action et qui ne communiquent pas. Ils subissent au plus les conséquences des actions des autres. Si on prend le cas d'un robot mobile naviguant dans un laboratoire, les objets inactifs peuvent représenter les murs, les armoires, etc ... ,
- les actifs, notés Obj_a : ce sont les objets qui peuvent générer des influences. En prenant l'exemple du robot mobile, un objet actif peut correspondre à une porte qui s'ouvre ou se referme aléatoirement. Ces objets actifs, sont juste prévus pour introduire une certaine dynamique dans le système et une sorte d'indéterminisme dans son évolution. Nous notons leur ensemble d'influences par A_{o_a}.

Un objet est caractérisé par un ensemble de traits ou d'observable accessibles par l'intermédiaire des capteurs des agents, permettant ainsi de les identifier.

Afin de simplifier les notations formelles dans la suite, nous étendons l'ensemble des objets aux agents, et nous notons le nouvel ensemble d'objets contenant les agents, les objets actifs et inactifs du système par $OBJ = \{Ags, Obj_a, Obj_i\}$.

Etant donné qu'un système multi-agent est dynamique, un ensemble de lois, appelées "lois du monde", est prévu pour régir l'évolution de l'environnement. Elles indiquent les conséquences des influences des agents ou des objets actifs sur l'environnement. Nous distinguons deux types de loi :

- les lois qui permettent de maintenir la cohérence du système : elles imposent l'exécution de certaines actions ou leur restriction lorsque certaines conditions deviennent vérifiées,

et quel que soient les influences des agents ou des objets actifs. En reprenant l'exemple des robots mobiles, un robot ne doit jamais traverser un mur ou un obstacle,

- des lois pour la gestion des influences simultanées : elles indiquent comment exécuter des influences générées simultanément. Cet aspect est relatif aux systèmes multi-agents, et est surtout utilisé quand un ensemble d'agents ou d'objets actifs produisent au même instant des influences conflictuelles. Ces lois indiquent donc comment résoudre les conflits et mettre en œuvre les différentes influences produites. C'est le cas par exemple de deux robots qui veulent avancer au même endroit, en même temps.

Ces lois vont nous permettre de décrire la loi d'évolution du monde T_a, et T_{o_a} (relative aux influences des objets actifs), que nous introduisons plus tard dans le paragraphe 3.2.6 décrivant la simulation basée sur notre modèle d'interaction.

L'espace E dans lequel évoluent les agents Ags, peut donc être décrit comme suit :

$$E = \{OBJ,M,R,WL\}$$

avec : M : une métrique, R : l'ensemble des relations entre les différents objets, et WL : l'ensemble des lois du monde.

Le modèle d'environnement, qui est une représentation abstraite de ce dernier, consiste en un ensemble d'états. Ce choix de modèle discret à base d'états, s'inspire des POMDP et nous sera utile pour la définition de notre modèle d'interaction agent/environnement par la suite.

Cet ensemble d'états de l'environnement, que nous notons S, aura donc la forme suivante : $S = \{s_1,s_2,\ldots,s_n\}$. Les s_i représentent des états élémentaires de S. Les agents et les objets que nous avons prévus occupent alors chacun un état élémentaire s_i. L'état S représente la configuration du système à un instant donné.

Un état élémentaire peut correspondre, par exemple, à une région de l'environnement dans le cas d'un robot mobile dans un environnement physique, comme il peut correspondre, dans le cas d'un agent mobile sur Internet, à un site du réseau.

L'ensemble S permet de définir la configuration de l'espace E à un instant donné, c'est-à-dire la position des différents objets dans E.

3.2.4 Modèle de l'agent

En proposant un modèle d'interaction agent/environnement intégrant la non-fiabilité des capteurs et des effecteurs, l'ambition est de pouvoir réaliser des simulations multi-agents dont les résultats seront "valides" en passant à l'expérimentation sur le système réel. Ceci veut dire que notre modèle doit nous permettre de construire des outils et des plates-formes de simulation, qui offrent la possibilité de tester différents comportements d'agents au sein d'un environnement et de les valider, avant de les intégrer dans le système réel. D'ailleurs, Bryson et al. ont discuté l'adéquation de la simulation, par rapport à d'autres plates-formes, comme les robots, pour la validation des comportements des agents. La conclusion qui a été tirée est que quelque soit la plate-forme utilisée, il faut qu'elle offre la possibilité de tester plusieurs hypothèses concurrentes, ainsi que de les évaluer (qualitativement ou quantitativement) [Bryson *et al.*, 2000]. Par conséquent, nous n'imposons pas de contraintes sur les modèles de comportements et de génération

d'influences des agents que nous laissons ouverts, même si un formalisme stochastique est implicite dans notre proposition. L'utilisateur est donc libre de spécifier n'importe quelle architecture ou modèle de comportement pour ses agents, afin qu'il puisse bénéficier d'un outil générique d'expérimentation et de test multi-agent. Ces tests seront menés dans le but de valider les comportements des agents et de voir leur robustesse aux bruits et aux erreurs des systèmes de perception et d'action.

Les agents doivent être situés et vérifier les propriétés précisées dans la section 1.2.1, à savoir l'autonomie, la perception et l'action locales sur l'environnement, etc ...

Un robot mobile représente donc un agent, puisqu'il vérifie les propriétés indiquées.

La définition de la fonction de comportement F_a et de génération d'influences $Infl_a$ reste donc à la charge de l'utilisateur :

$$F_a : \xi_a \times Percept_a \rightarrow \xi_a \quad et \quad Infl_a : \xi_a \rightarrow A_a$$

F_a permet de déterminer le nouvel état interne de l'agent en fonction de son état précédent et de ses perceptions. Par la suite, l'agent génère son influence en fonction de ce nouvel état interne. Les fonctions F_a et $Infl_a$ intègrent implicitement le(s) but(s) de l'agent.

3.2.5 Le modèle d'interaction agent / environnement

Après avoir défini le modèle de l'environnement, nous décrivons maintenant comment nous modélisons l'interaction entre l'agent et l'environnement, tout en tenant compte de la non-fiabilité des capteurs et des effecteurs des agents. En effet, ceci passe par l'intégration des erreurs et incertitudes au niveau des processus de perception de l'agent de son environnement, et de l'action de l'agent sur ce dernier.

Modélisation de la perception

Les agents situés perçoivent leur environnement via leurs capteurs. Ces derniers leur fournissent des mesures numériques, à partir desquelles les agents extraient des informations symboliques, que nous appelons des *observations*, et les distances qui séparent l'agent de chaque observation.

Chaque objet est alors désigné par une observation, qui est fonction du type de l'agent et des capteurs utilisés (figure 3.3), de la façon suivante :

$$\begin{aligned} f_o \quad : \quad & OBJ \times Capts_a \longrightarrow OBSV \\ & obj,capt \longmapsto obsv_i \end{aligned}$$

avec $OBSV = \{obsv_i, tq : i = 1..L\}$.

Une observation est alors une caractéristique de l'objet vue au travers d'un capteur.

Les mesures numériques initiales sont parfois soit incertaines, soit non discriminantes. Pour cette raison, deux objets différents ou plus, peuvent être désignés par la même observation, afin d'exprimer les incertitudes des capteurs, i.e. : $\exists\ obj_1,obj_2 \in OBJ$, tq : $f_o(obj_1,capt) = f_o(obj_2,capt)$.

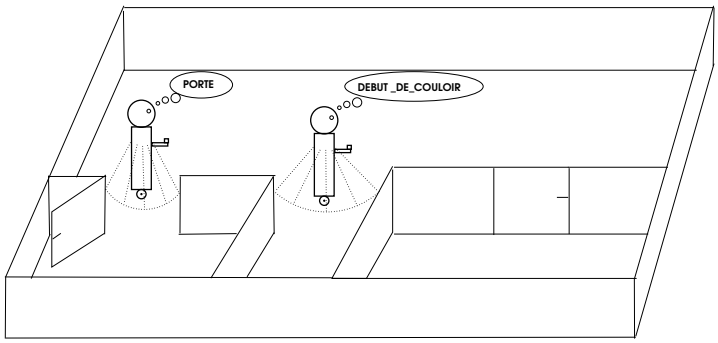

FIG. 3.3 – *Exemple de désignation d'objets en fonctions des capteurs utilisés par l'agent*

Les mesures initiales peuvent encore être plus erronées, en raison du bruit et de la déformation des signaux des capteurs. En outre, quelques erreurs peuvent se produire pendant le processus d'identification des observations. C'est pour cette raison qu'une *matrice de confusion entre les observations* est définie, afin d'exprimer toutes les erreurs qui peuvent se produire au niveau des capteurs des agents.

En d'autres termes, pour chaque agent et chaque type de capteur nous définissons un ensemble d'observations permettant d'identifier les différents objets du système. Ensuite, nous déterminons pour chaque observation une distribution de probabilité sur l'ensemble des observations qui peuvent être confondues avec elle.

Par exemple, dans la figure 3.4 l'$Agent_a$ attribue l'étiquette $Obsv_x$ à l'objet *Objet* pour l'identifier. Mais, parfois il peut confondre cet objet avec d'autres présents dans l'environnement, au moment de sa reconnaissance. Il peut par exemple l'identifier comme étant un autre objet, dont l'observation est $Obsv_1$ avec une probabilité P_{x_1}, ou $Obsv_2$ avec P_{x_2} et ainsi de suite. Nous avons donc $\sum_{i=1}^{L} P_{x_i} = 1$. De cette façon nous définissons la matrice de confusion entre les différentes observations du système, qui peut être établie par apprentissage sur le système réel, par exemple. Nous disposons donc d'une matrice de cette forme :

	$obsv_1$	$obsv_2$	...	$obsv_L$
$obsv_1$	p_{11}	p_{12}	...	p_{1L}
$obsv_2$	p_{21}	p_{22}	...	p_{2L}
...			...	
$obsv_L$	p_{L1}	p_{L2}	...	p_{LL}

TAB. 3.1 – *La matrice de confusion entre les observations*

De même, afin de tenir compte des incertitudes des distances séparant les entités observées et les agents, nous avons *divisé le rayon de perception de l'agent en classes* et attribué à chaque distance réelle mesurée (dans la métrique M de E) une classe. Nous définissons une fonction $f_{\mathcal{D}}$ de la façon suivante :

$$\begin{aligned} f_{\mathcal{D}} \quad : \quad & M \times Capts_a \longrightarrow \mathcal{D} \\ & d,capt \longmapsto D_i \end{aligned}$$

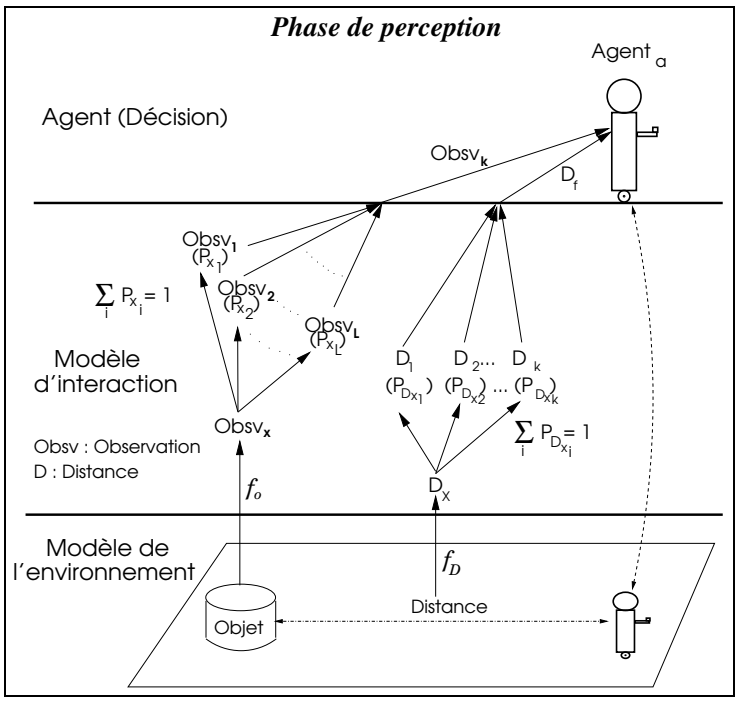

FIG. 3.4 – *Le modèle d'interaction agent/environnement : phase de perception*

avec $\mathcal{D} = \{D_i,\ tq : i = 1..k\}$, l'ensemble des classes de distances. La fonction $f_{\mathcal{D}}$ associe une classe de distance D_i à chaque distance réelle perçue par l'agent, afin de la rendre approximative, ou encore incertaine. De plus, une *matrice de confusion entre les classes de distance* est définie pour exprimer les erreurs qui peuvent se produire au niveau des capteurs et/ou au cours de la phase de calcul de la distance.

Par exemple, dans la même figure 3.4 la distance qui sépare l'objet de l'agent est attribuée à la classe D_x qui peut être confondue avec la classe D_1 selon la probabilité $P_{D_{x_1}}$, D_2 selon $P_{D_{x_2}}$, ... , et nous avons toujours $\sum_{i=1}^{k} P_{D_{x_i}} = 1$.

La matrice de confusion entre les classes de distances peut être aussi à son tour déterminée par apprentissage sur le système réel, et elle aura la même forme que celle des observations.

L'ensemble des stimuli et sensations d'un agent sera donc le suivant :

$$Percept_a = \{(obsv_i, D_i),\ tq : obsv_i \in OBSV \text{ et } D_i \in \mathcal{D}\}$$

Modélisation de l'influence

Chaque agent est équipé d'effecteurs pour agir au sein de l'environnement. Nous appelons *transition* la réaction de l'environnement à l'influence générée par la fonction $Infl_a$ de l'agent. La transition indique quel sera le nouvel état de l'environnement suite à la mise en œuvre de l'influence de l'agent. Cette transition dépend du type de l'agent, de son influence, de son voisinage et des lois du monde. C'est-à-dire que la transition peut changer en fonction des erreurs ou de l'imprécision des effecteurs et qu'elle dépend aussi du voisinage de l'agent. Ainsi, pour *chaque type d'agent et chaque influence* qu'il

peut générer, *un ensemble de transitions, ainsi qu'une distribution de probabilité* sur cet ensemble, exprimant leurs probabilités d'occurrence, sont définis.

Par exemple, dans la figure 3.5, à l'influence générée par la fonction $Infl_a$ de l'agent correspondent N transitions possibles $\{Trans_1, \ldots, Trans_N\}$. Chaque transition $Trans_i$ a une probabilité d'occurrence égale à P_i, tel que : $\sum_{i=1}^{N} P_i = 1$. Nous définissons donc une fonction par agent a :

$$\begin{aligned} f_{infl_a} \quad : \quad & A_a \times E \longrightarrow TRANS \\ & influence, E \longmapsto \{(Trans_1, P_1), \ldots, (Trans_N, P_N)\} \end{aligned}$$

Les ensembles de transitions correspondant aux différentes influences des agents, ainsi que leurs distributions de probabilités peuvent également être déterminés par apprentissage sur le système réel.

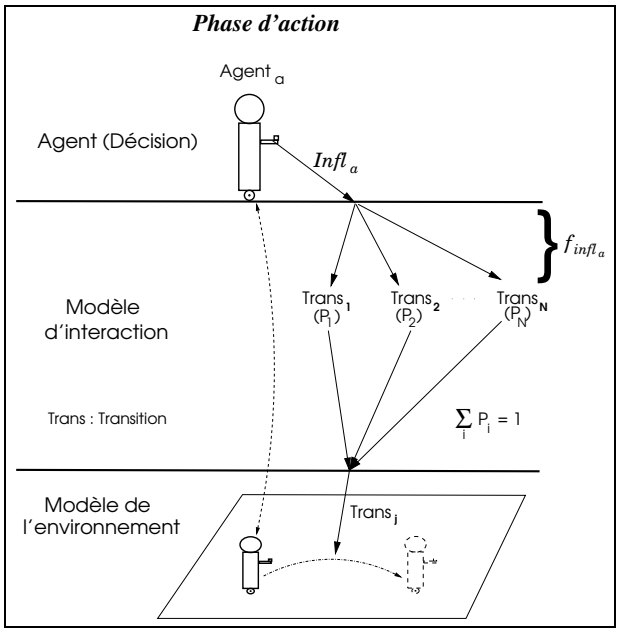

FIG. 3.5 – *Le modèle d'interaction agent/environnement : phase d'action*

Combinaison des influences des agents

Une politique de résolution de conflits doit être prévue par le concepteur du modèle du système à simuler, pour gérer le cas où deux agents ou plus émettent des influences simultanées conflictuelles. Le concepteur doit donc prévoir tous les types de conflits qui peuvent se produire entre les agents (ex : occupation d'un même emplacement, acquisition d'un même objet, ...). Il doit ensuite indiquer comment résoudre ces différents conflits, et l'ensemble des solutions forme ce que nous appelons la politique de résolution de conflits. Nous disposons donc d'un opérateur de combinaison d'influences $\prod$, qui permet de détecter et de résoudre les conflits entre les différentes influences des agents (voir la seconde étape du déroulement de la simulation du paragraphe suivant). Cette

politique va nous permettre de déterminer l'ensemble des transitions possibles relatives à une influence, en cas de conflit(s). Nous aurons donc pour chaque influence un ensemble de transitions formé de transitions en absence de conflits, et de transitions en cas de conflits, ayant chacune une certaine probabilité d'occurrence. Si nous prenons le cas où un agent représente un engin mobile, l'agent va suivre une certaine trajectoire indiquée par les transitions possibles, s'il ne rencontre pas d'obstacles sur son chemin. Mais, dans le cas où un autre agent croise sa trajectoire sans qu'ils n'essaient de s'éviter, les deux agents seront arrêtés ou déviés de leurs trajectoires en fonction des lois physiques qui gouvernent la collision entre deux corps mobiles. D'où l'apparition de nouvelles transitions pour chaque agent suite à l'occurrence de la collision.

Evidemment, les transitions qui apparaissent suite à un conflit uniquement, auront des probabilités nulles dans le cas général, et elles verront leurs probabilités changer en cas de conflit. Remarquons que lorsque les probabilités de transition changent, leur somme totale reste toujours égale à 1, puisque c'est une distribution de probabilités qui régit l'ensemble des transitions d'une influence donnée. Par exemple, dans la figure 3.6, les agents $Agent_1$ et $Agent_2$ ont deux influences conflictuelles et par conséquent les distributions de probabilité sur les ensembles de transitions correspondants seront modifiées suite au résultat de la politique de résolution de conflits (pour un exemple plus détaillé, reportez-vous au chapitre 4, "Application").

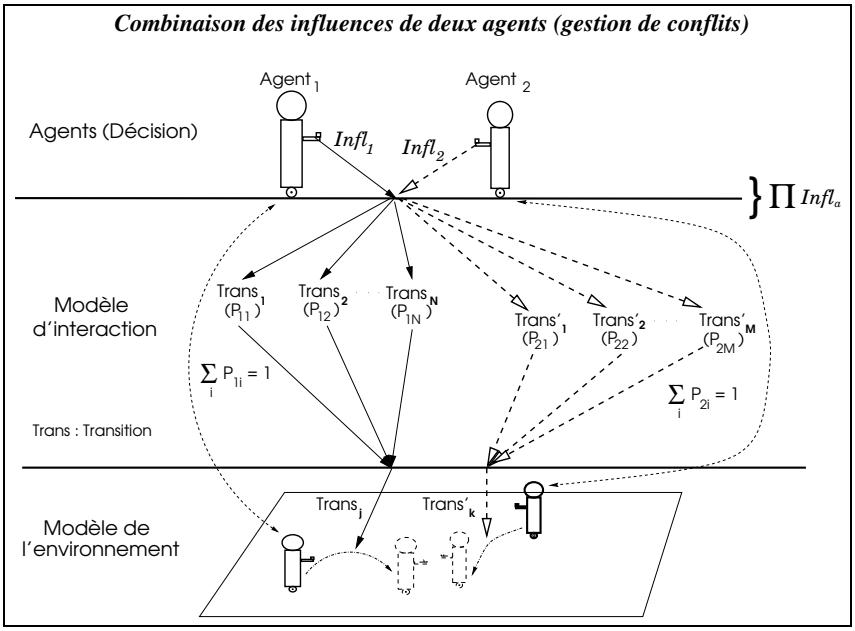

FIG. 3.6 – *Le modèle d'interaction agent/environnement : combinaison des influences*

3.2.6 Le principe de la simulation

Les distributions de probabilité sur les observations, les classes de distances et les transitions peuvent être apprises sur le système réel que l'on veut modéliser. Une fois ces distributions obtenues, nous dérivons le modèle d'interaction agent/environnement

spécifique à l'application en question, en créant une instance des modèles de perception et d'influence cités ci-dessus.

Modélisation de l'espace

Comme nous l'avons déjà mentionné, notre modèle d'interaction agent/environnement est inspiré d'une certaine classe de modèles de décision Markoviens, qui est celle des POMDP. Chaque agent occupe un état élémentaire et passe d'un état à l'autre selon une certaine probabilité. Par conséquent, bien qu'un modèle d'espace continu dans certains cas soit plus adapté pour simuler certains types d'application (comme les robots mobiles), notre modèle stochastique nous amène à utiliser un modèle discret pour l'environnement, qui consiste en un ensemble d'états (voir section 2.3.1).

Modélisation du temps

Nous avons vu dans la section 2.3.2, qu'un modèle de temps événementiel est meilleur qu'un modèle à pas de temps discret (plus rapide au niveau du temps d'exécution). Dans le cas de notre modèle d'interaction stochastique, l'évolution du système n'est pas déterministe, puisque à partir d'un état donné, le système peut passer à plusieurs autres états, en fonction d'une certaine distribution de probabilité. Ainsi, nous ne pouvons pas toujours prévoir l'évolution du système, même en l'absence de conflits entre les agents. Pour cette raison, nous ne pouvons pas déterminer à l'avance tous les instants possibles de conflit entre les agents, afin de faire évoluer la simulation d'un conflit à l'autre (i.e. modèle événementiel). Cela est d'autant plus accentué par la présence d'un grand nombre d'agents dans l'environnement. Nous avons donc décidé d'utiliser un modèle de temps discret pour notre simulateur.

Cependant, nous pensons qu'il y a encore un troisième type de modèle de temps, qui est hybride des deux modèles que nous avons trouvés dans la littérature, qui est le type discret à pas de temps variable. En effet, dans certains cas, nous avons à simuler des systèmes où plusieurs événements se produisent dans un petit laps de temps, d'un moment à l'autre, et il est préférable de mener la simulation à petits pas. A d'autres moments, ces systèmes n'ont que peu d'événements qui se produisent sur une grande période de temps, et il est plus avantageux de dérouler la simulation à grands pas, afin de réduire son temps d'exécution global. Ce basculement entre différents pas de temps, peut être prédit à l'avance en estimant approximativement les périodes où il y aura beaucoup d'événements et celles où ce ne sera pas le cas. Cela peut aussi se faire dynamiquement au cours de l'exécution, en se basant toujours sur les estimations du nombre d'événements qui vont se produire.

Par conséquent, nous pouvons utiliser pour notre simulateur un modèle de temps discret à pas variable, dans le cas où nous pouvons estimer les périodes de production des événements. Dans le cas contraire, nous utiliserons un modèle discret à pas de temps constant.

Après avoir choisi ce modèle de temps, nous avons décidé de permettre à chaque agent d'avoir sa propre fréquence d'action. En effet, comme nous utilisons un modèle de temps à pas discret, qui impose aux agents de se synchroniser à chaque cycle, le

fait que chaque agent ait une fréquence d'action qui peut différer de celles des autres, introduit dans le simulateur une certaine flexibilité. Les agents les plus rapides au niveau de leur prise de décision, ne seront donc pas pénalisés à cause des plus lents, puisqu'ils n'ont pas à les attendre pour voir leurs influences se mettre en œuvre. Le simulateur va permettre ainsi de remédier un peu aux inconvénients des modèles de temps à pas discret, discutés dans la section 2.3.2. La fréquence d'action d'un agent peut varier au cours de la simulation même, en fonction de l'application en question.

Le simulateur peut alors avoir un pas de temps discret et variable, toujours en fonction de la fréquence de l'agent le plus rapide. Les simulations peuvent ainsi se dérouler sur plusieurs échelles de temps.

Déroulement de la simulation

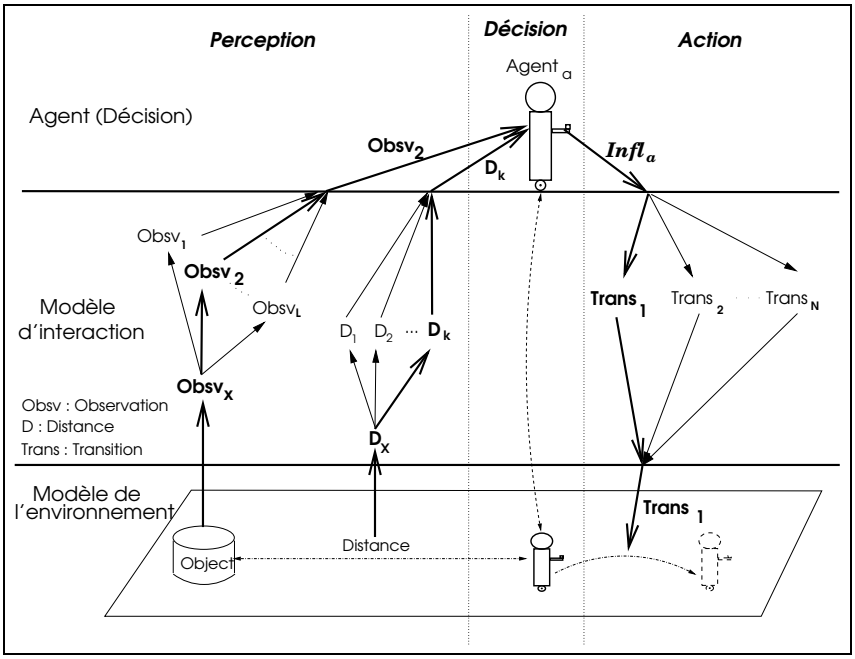

FIG. 3.7 – *Le principe de la simulation de l'interaction agent/environnement*

La simulation consiste à faire évoluer le système au cours du temps par cycles. Dans chaque cycle, tous les agents actifs (en fonction de leurs fréquences) perçoivent simultanément l'environnement. Par la suite, chaque agent met à jour son état interne en fonction de son état précédent et des percepts récupérés. Il génère ensuite son influence, en fonction de ce dernier (i.e. état interne). Simultanément, les autres objets actifs (en fonction de leurs fréquences aussi) produisent leurs influences selon des règles préétablies, comme dans le cas d'une porte qui s'ouvre ou se referme aléatoirement. Ensuite, le simulateur analyse les différentes influences afin de détecter les éventuels conflits ; il résout alors les conflits et exécute les différentes influences, tout en mettant à jour les différentes propriétés de l'environnement (l'occupation des états, les distributions de

probabilité, etc ...), après la mise en œuvre de chaque influence. Cela représente donc la transition de l'environnement.

Nous avons alors, au cours d'un cycle, une boucle principale composée de deux étapes :

- dans la première étape, les agents perçoivent l'environnement et produisent leurs influences (les autres objets actifs produisent juste leurs influences),
- dans la seconde étape, l'environnement réagit à ces décisions.

Nous décrivons dans ce qui suit le principe de la simulation stochastique que nous avons conçue.

Première étape : perception de l'environnement

Chaque agent perçoit son environnement. Pour chaque capteur ou ensemble de capteurs, une observation et une classe de distance sont choisies aléatoirement parmi celles qui peuvent être confondues avec les observations et classe de distance que l'agent devrait percevoir, tout en respectant les distributions de probabilité correspondantes.

Par exemple, dans la figure 3.7, partie "Perception" : $Obsv_x$ et D_x sont les bonnes observation et classe de distance, alors que $Obsv_2$ et D_k sont celles aléatoirement choisies.

La fonction de perception d'un agent a, a donc le profil suivant :

$$P_a : S \times \{[0,1] \times [0,1]\}^{nc} \rightarrow Percept_a.$$

Les intervalles [0,1] correspondent aux probabilités déterminant le choix d'une observation et d'une classe de distance, respectivement. Le paramètre nc représente le nombre de capteurs utilisés par l'agent a pour percevoir son environnement.

Une fois les observations et les distances obtenues, l'agent met à jour son état interne, selon la fonction : $F_a : \xi_a \times Percept_a \rightarrow \xi_a$. Ensuite, il génère son influence : $Infl_a : \xi_a \rightarrow A_a$. Comme déjà signalé, les objets actifs génèrent selon des règles préétablies ou aléatoirement leurs influences, afin d'introduire une certaine dynamique dans l'environnement : $Infl_{o_a} : \quad \rightarrow A_{o_a}$

Cette étape se résume formellement comme suit :

- calcul pour chaque agent a actif,
  - de son nouvel état interne : $\xi_a(t) = F_a(\xi_a(t-1), P_a(s(t), \{p_o, p_d\}^{nc}))$, avec $p_o, p_d \in [0,1]$. p_o est une probabilité, choisie au hasard, permettant de déterminer l'observation qui sera perçue à ce pas de temps de simulation via un capteur donné. De la même façon, p_d est la probabilité qui concerne le choix d'une classe de distance,
  - et de son influence : $A_a(t) = Infl_a(\xi_a(t))$.
- génération de l'influence de chaque objet actif o_a : $A_{o_a}(t) = Infl_{o_a}()$

Seconde étape : exécution des influences

Le simulateur rassemble toutes les propositions d'actions, c'est-à-dire les influences et détermine l'ensemble des agents et des objets actifs ayant des influences conflictuelles et les résout. Ceci s'effectue en respectant la politique fixée en fonction du problème, appliquée à l'ensemble des influences des agents et des objets actifs : $\prod Infl_a(\xi_a(t))$ et

$\prod Infl_{o_a}()$. Suite à cette étape, nous aurons pour chaque influence générée l'ensemble des transitions possibles après l'application des opérateurs de combinaison des influences.

Le simulateur exécutera alors les influences de la façon suivante : l'exécution d'une influence d'un agent se traduit par le choix probabiliste d'une transition parmi celles qui peuvent résulter de cette influence. Nous rappelons que les transitions correspondant à une influence comprennent, à la fois, celles qui se produisent en absence et en présence de conflit (activées ou désactivées à l'aide de leurs probabilités d'occurrence). Dans la figure 3.7, par exemple, partie "Action" : la transition $Trans_1$ a été choisie aléatoirement parmi les transitions $\{Trans_1, \ldots, Trans_N\}$ de l'influence générée par la fonction $Infl_a$ de l'agent. La loi décrivant l'évolution du monde suite à l'action d'un seul agent a le profil suivant :

$$T_a : S \times A_a \times [0,1] \rightarrow S.$$

L'intervalle [0,1] sert évidemment à indiquer que la transition du monde qui sera choisie dépend d'une certaine probabilité. L'influence d'un objet actif est tout simplement exécutée en tenant compte des lois du monde :

$$T_{o_a} : S \times A_{o_a} \rightarrow S.$$

Ces deux fonctions T_a et T_{o_a} vont donc exécuter les différentes influences, tout en respectant les lois du monde et mettre à jour les différentes occupations des états de l'environnement, les distributions de probabilité des transitions et des observations, etc ...

Nous remarquons que deux influences conflictuelles ou plus doivent être combinées en séquentiel, selon la politique de résolution de conflit, afin de garder la cohérence globale du système. Les résultats de cette combinaison d'influences seront exécutés en utilisant les fonctions adéquates T_a ou T_{o_a} d'évolution du monde.

Cette étape qui fait évoluer tout le système de son état S de l'instant t à un nouvel état à l'instant $t+1$ se résume formellement comme suit :

$$S(t+1) = \hat{\sum_{a \in Ags}} T_a(S(t), \prod Infl_a(\xi_a(t)), p_a) \ \hat{+} \ \hat{\sum_{o_a \in Obj_a}} T_{o_a}(S(t), \prod Infl_{o_a}()),$$

avec $p_a \in [0,1]$. Cette expression veut dire que l'évolution finale du système est la somme ($\hat{+}$), conformément à la politique de résolution de conflits, de la somme ($\hat{\sum}$) des transitions résultantes des influences des agents (de façon probabiliste), et de celles des objets actifs, après la détection et la résolution de conflits entre elles.

La figure 3.8 illustre le passage d'un système ayant deux agents d'un état $S(t)$ à l'instant t à l'état $S(t+1)$ à l'instant $t+1$ au cours d'un pas de simulation. Chaque agent a_i applique sa fonction de perception, met à jour son état interne et génère son influence. Par la suite, l'opérateur de combinaison des influences est appliqué afin de détecter et de résoudre les conflits entre les différentes influences. Enfin, le nouvel état du système est généré en fonction de la mise en œuvre de ces dernières et de son ancien état.

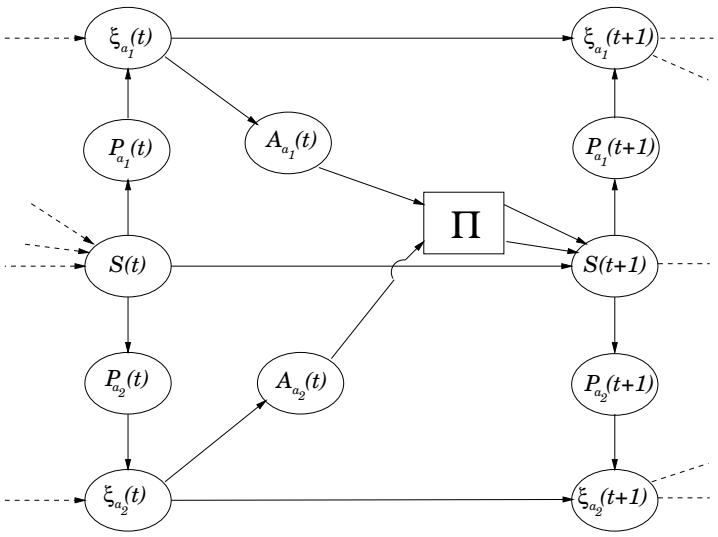

FIG. 3.8 – *Les étapes permettant le passage d'un système à deux agents a_1 et a_2, d'un état au suivant entre deux instants t et $t+1$*

3.2.7 Récapitulatif : Modèle d'interaction et simulation

En résumé, un système multi-agent utilisant notre modèle stochastique d'interaction entre l'agent et l'environnement, se présente comme suit : $SMA =< Ags, W, \prod >$ tels que,

- $Ags = \{Ag\}$ est l'ensemble des agents,
- $W =< E, A, S, T_a >$ est l'environnement,
- $\prod$ est l'opérateur de combinaison des influences des agents.

Un agent Ag est défini par : $Ag =< P_a, Capts_a, Percept_a, F_a, Infl_a, \xi_a >$, avec :

- $Capts_a$ est l'ensemble des types de capteurs utilisés par l'agent a,
- $Percept_a = \{(obsv_i, D_i),\ tq : obsv_i \in OBSV \text{ et } D_i \in \mathcal{D}\}$ est l'ensemble des stimuli et sensations de l'agent a,
- $P_a : S \times \{[0,1] \times [0,1]\}^{nc} \rightarrow Percept_a$ est la fonction de perception de l'agent a. Le paramètre nc est le nombre de capteurs utilisés par l'agent a pour percevoir son environnement,
- ξ_a est l'ensemble des états internes de l'agent a,
- $F_a : \xi_a \times Percept_a \rightarrow \xi_a$ est la fonction de comportement de l'agent a,
- $Infl_a : \xi_a \rightarrow A_a$ est la fonction de génération d'influence de l'agent a.

Les différentes caractéristiques de l'environnement sont :

- $E = \{OBJ, M, R, WL\}$ est l'espace dans lequel évolue les agents, avec :
  $OBJ = \{Ags, Obj_a, Obj_i\}$, Obj_a : les objets actifs, Obj_i : les objets inactifs, M : Métrique, R : Relations entre objets, et WL : l'ensemble des lois du monde,
- $A = \bigcup_{a \in Ags} A_a$ est l'ensemble des influences qui peuvent être générées par tous les agents,
- $S = \{s_1, s_2, \ldots, s_n\}$ est l'ensemble des états de l'environnement,

- $T_a : S \times A_a \times [0,1] \rightarrow S$ est la loi d'évolution du monde suite à l'exécution d'une influence d'un agent.

Les objets actifs Obj_a (prévus pour introduire une certaine dynamique dans l'environnement) :

- A_{o_a} est l'ensemble des influences de l'objet o_a,
- $Infl_{o_a} : \quad \rightarrow A_{o_a}$ est la fonction de génération d'influence de l'objet o_a,
- $T_{o_a} : S \times A_{o_a} \rightarrow S$ est la loi d'évolution du monde suite à l'exécution d'une influence d'un objet actif.

La simulation multi-agent utilisant cette description de SMA, s'effectue comme suit :

- calcul du nouvel état interne de chaque agent a, ainsi que de sa nouvelle influence :

$$\xi_a(t) = F_a(\xi_a(t-1), P_a(s(t), \{p_o, p_d\}^{nc})), \; avec \; p_o, p_d \in [0,1]$$

$$A_a(t) = Infl_a(\xi_a(t))$$

- détermination des influences des objets actifs :

$$A_{o_a}(t) = Infl_{o_a}()$$

- puis détermination de l'évolution totale du monde :

$$S(t+1) = \hat{\sum_{a \in Ags}} T_a(S(t), \prod Infl_a(\xi_a(t)), p_a) \; \hat{+} \; \hat{\sum_{o_a \in Obj_a}} T_{o_a}(S(t), \prod Infl_{o_a}()),$$

avec $p_a \in [0,1]$.

3.3 Un modèle de simulation parallèle

3.3.1 Motivations

Nous avons vu dans la partie décrivant les travaux dans le domaine de la simulation multi-agent parallèle (c.f. section 2.5), que le parallélisme a été principalement utilisé pour offrir un cadre simple et intuitif pour l'expression des comportements des agents. Ceci a été établi en exploitant le parallélisme inhérent des systèmes multi-agents. Nous avons aussi signalé que le parallélisme nous permet généralement d'accélérer l'exécution des simulations (*speed-up*) et/ou de traiter des problèmes de plus grandes tailles (*size-up*).

Suite à notre étude bibliographique, nous avons trouvé que les approches d'implantations parallèles de simulateurs multi-agents (répartition spatiale, parallélisation basée agent et parallélisation des différents composants d'un agent) ne sont pas vraiment adaptées pour obtenir de bonnes performances, d'un point de vue parallélisme (i.e. *speed-up* et *size-up*). Nous détaillons donc dans ce qui suit notre solution basée sur les conflits, qui est un peu plus complexe à implanter mais plus efficace que les autres méthodes [Bouzid *et al.*, 2001].

3.3.2 Parallélisation basée sur les conflits

C'est l'approche que nous proposons pour simuler des systèmes multi-agents situés de façon généralement efficace [Bouzid *et al.*, 2000a]. Notre idée est basée sur le principe suivant :
Un agent représente fondamentalement une tâche, mais s'il y a un conflit entre un ensemble d'agents, ils doivent être fusionnés en une et une seule plus grande tâche, afin qu'ils soient traités par un seul processeur, et assurer ainsi la cohérence du monde global (i.e. celles des données partagées). Par conséquent, cette stratégie nous épargne les surcoûts de communication entre les processeurs pour résoudre les conflits. Mais, le nombre de tâches et leurs tailles sont variables au cours de la simulation, ce qui peut provoquer un déséquilibre de charge entre les processeurs. De plus, beaucoup de tâches peuvent avoir des tailles trop petites pour en faire des processus à part entière.

Nous avons donc complété cette stratégie de parallélisation en utilisant un mécanisme d'équilibrage dynamique de charge, qui est le mécanisme de *work-pool* (réserve de tâches) [Lester, 1993] (figure 3.9). Toutes les tâches précédentes sont implantées sous la forme de simples structures de données, stockées dans une zone mémoire partagée, qui est le *work-pool*. Un ensemble de processus, appelés *workers*, est prévu pour la gestion de n'importe quel type de tâche figurant dans le *work-pool*. Les processus *workers*, sont répartis à raison d'un par processeur, afin d'éviter les pertes de temps dans le basculement dynamique des processus entre les processeurs. Chaque processus récupère une tâche du *work-pool*, la traite et génère éventuellement de nouvelles tâches dans le *work-pool*, et continue ainsi de suite jusqu'à ce qu'il ne reste plus de tâches dans le *work-pool*. Les accès concurrents au *work-pool*, qui est une structure de donnée partagée, reste une source possible de perte de temps, mais il existe certaines implantations optimisées pour limiter ce problème. Enfin, nous avons conçu un algorithme de *work-pool* optimisé pour notre cas particulier de simulation d'agents situés, utilisant un mécanisme de double *work-pool* et où le traitement d'une tâche ne génère pas de nouvelles tâches dans le même *work-pool*.

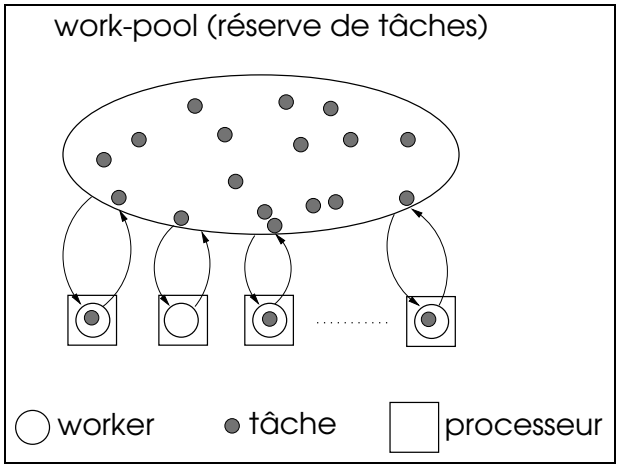

FIG. 3.9 – *Le mécanisme de work-pool*

3.3.3 Le mécanisme de double work-pool

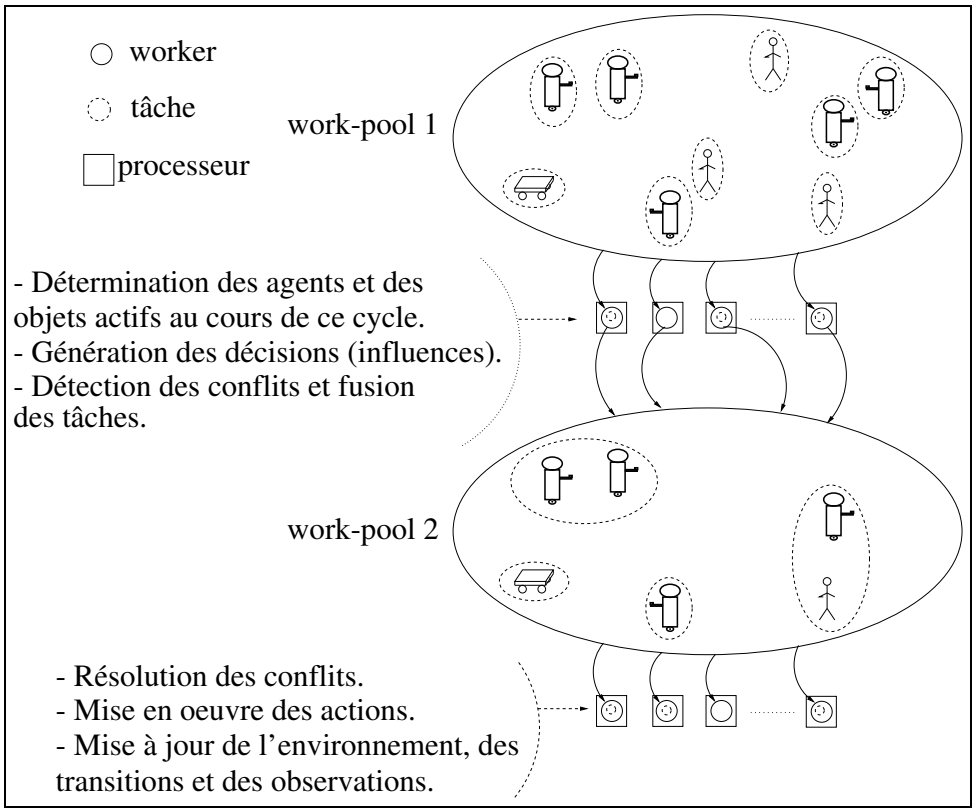

FIG. 3.10 – *Simulation parallèle basée sur les conflits, utilisant deux work-pools en cascade pour assurer un équilibrage dynamique de charge*

Nous avons créé deux *work-pools*, gérés par P processus *workers*, de la façon décrite dans la figure 3.10. Au début de chaque cycle, nous insérons tous les agents et les objets actifs dans le premier *work-pool*, sous forme de tâches basiques. Chaque *worker* récupère une tâche de ce *work-pool*, et teste si elle est active durant ce cycle ou non, c'est-à-dire si elle doit générer une influence. Ce test porte sur la fréquence de l'agent ou de l'objet en question, qui est l'inverse de sa période d'action. Si elle est active et si la tâche représente un agent, le processus *worker* exécute ses phases de perception et de génération d'influence de l'agent. Dans le cas où la tâche est un objet actif, le processus *worker* génère son influence aléatoirement, puisqu'il n'est prévu que pour introduire une certaine dynamique et indéterminisme dans le système. Nous avons donc formellement :

$$\underset{a \in Ags}{\|} [\xi_a(t) \;\; seq \;\; Infl_a(\xi_a(t))] \qquad \underset{o_a \in Obj_a}{\|} Infl_{o_a}()$$

avec, seq : exécution en séquentiel, $\|$: exécution en parallèle.

Ensuite, le *worker* insère l'agent ou l'objet, avec l'influence générée dans le second *work-pool* sous la forme d'une tâche. Mais, pendant l'insertion de cette tâche, le *worker* doit vérifier s'il y a un conflit entre elle et les autres tâches déjà présentes dans

le second *work-pool*. Si c'est le cas, les tâches conflictuelles doivent être fusionnées en une seule tâche de plus grande taille, afin qu'elle soit traitée par un seul processus *worker* et afin de garder un monde cohérent, sans surcoût excessif de communication entre processeurs. De façon formelle :

$$\underset{o \in (Ags \cup Obj_a)}{seq} \prod Infl_o$$

Durant cette phase de combinaison des influences des tâches, deux processus *workers* ou plus, insérant chacun une tâche dans le second *work-pool* simultanément, doivent éviter de tester les conflits avec une même tâche au même instant (puisqu'ils doivent la fusionner avec les siennes en cas de conflit). En outre, les tâches insérées simultanément doivent aussi être testées entre elles et fusionnées en cas de conflit. Ceci doit être fait par un seul processus *worker*, ce qui implique une synchronisation et une délégation de la combinaison de ces dernières tâches à un seul *worker* parmi ceux qui ont inséré des tâches simultanément dans le second *work-pool*. Nous pouvons remarquer que cette procédure est un peu complexe et peut provoquer un surcoût de parallélisme (synchronisation, communication). Nous avons donc choisi de traiter cette phase de combinaison des influences en séquentiel afin de ne pas compliquer davantage notre algorithme de simulation parallèle. Cela peut, malheureusement, constituer un goulot d'étranglement séquentiel pour notre simulation parallèle, mais nous ne pensons pas que le surcoût de ce choix séquentiel sera beaucoup plus important que le surcoût d'un traitement parallèle, qui nécessite des synchronisations et des communications entre les processus *workers*.

Une fois que toutes les tâches du premier *work-pool* sont traitées, les *workers* sont synchronisés en utilisant une barrière de synchronisation, avant de passer aux traitements des tâches du second *work-pool*. Par la suite, chaque *worker* cherche une tâche dans le second *work-pool*, résout les conflits entre les différents agents et objets de cette tâche, selon la politique fixée. Après, il met en œuvre leurs différentes actions, en exécutant les transitions correspondantes en séquentiel, tout en respectant les résultats de la politique de résolution de conflits. Enfin, le *worker* met à jour les différentes occupations des états de l'environnement, ainsi que les distributions de probabilité des transitions et des observations, après chaque transition effectuée. Formellement, nous avons :

$$\underset{Co \in Conflits}{\|} \; [\underset{o \in Co}{seq} T_o]$$

tels que :

- $Conflits$: l'ensemble des conflits entre les agents et/ou les objets actifs (i.e. l'ensemble des tâches du second *work-pool*),
- $\forall\, Co \in Conflits$, $\forall\, x, y \in Co$, $\exists$ un conflit entre x et y, ou $\exists\, z \in Co$, tel que x est en conflit avec z et y est en conflit avec z,
- $\forall\, Co_1, Co_2 \in Conflits$, $Co_1 \cap Co_2 = \varnothing$,
- $\forall\, x \in Co_1$, $\forall\, y \in Co_2$, $\nexists$ de conflit entre x et y.

Une fois que toutes les tâches du second *work-pool* sont traitées, les *workers* sont à nouveau synchronisés, en utilisant une autre barrière avant de passer au cycle de simulation suivant.

3.3.4 La programmation parallèle du simulateur

Après la présentation de notre modèle de simulation parallèle, nous décrivons maintenant les différents niveaux d'abstraction nécessaires pour la programmation d'un simulateur multi-agent basé sur ce modèle (c.f. section 2.4.3).

- Modèle de programmation : dans notre modèle de simulation, ce sont les tâches qui guident le parallélisme. Les tâches sont les agents, les objets actifs du système, et les conflits qui peuvent apparaître entre ces différentes composantes. Deux points de vue peuvent être adoptés pour la détermination du modèle de programmation. Si les tâches sont considérées comme de simples données traitées par les processus *workers*, nous pouvons dire que nous adoptons un modèle de programmation basé sur le parallélisme des données. Mais, si nous considérons que les processus *workers* prennent l'identité des tâches qu'ils traitent, c'est-à-dire qu'à chaque fois le *worker* se transforme en un agent, ou un objet actif ou un mécanisme de détection et de résolution de conflits, nous pouvons dire que nous effectuons un parallélisme de contrôle.

- Modèle d'exécution : notre modèle de simulation a un modèle d'exécution de type SPMD, puisque les processus *workers* appliquent le même algorithme (au niveau des deux *work-pools*) à des tâches différentes.

- Architecture : concernant le niveau architecture, notre modèle de simulation n'impose qu'une seule condition, qui est celle d'avoir une mémoire partagée, réelle ou virtuelle, pour contenir les *work-pools* qui doivent être accessibles par tous les processeurs.

3.4 Conclusion

Dans ce chapitre, nous avons présenté formellement et décrit textuellement les deux modèles que nous proposons pour la simulation multi-agent, qui sont les suivants :

- un modèle formel de systèmes multi-agents, comprenant un modèle d'environnement, un modèle d'agent, et principalement un modèle stochastique d'interaction entre l'agent et l'environnement. Notre modèle stochastique permet de modéliser les incertitudes et les erreurs des capteurs des agents, ainsi que de leurs effecteurs. L'ensemble des transitions résultantes des influences des agents et les distributions de probabilité qui les régissent, ainsi que les matrices de confusion entre les observations peuvent être apprises à partir du système réel. Ce modèle devrait permettre de construire des plates-formes de simulations multi-agents, dont les résultats des comportements des agents seront assez valides lors du passage à leurs expérimentations sur le système réel,
- un modèle de simulation multi-agent parallèle basé sur les conflits, utilisant un mécanisme original de double *work-pool*, exploitant le parallélisme inhérent des SMA. Ce modèle permet d'exprimer implicitement les comportements des agents, sans se soucier de leurs synchronisations, comme dans le cas des implantations séquentielles, et d'effectuer un équilibrage de charge dynamique entre les processeurs, afin d'obtenir de bonnes performances parallèles.

Nous vérifions dans les chapitres suivants la validité de nos idées, ainsi que de certains résultats attendus et cités précédemment.

Chapitre 4

Application : Vers une plate-forme de simulation de robots mobiles

4.1 Introduction

Afin de tester et de valider nos modèles d'interaction agent/environnement et de simulation parallèle, nous avons développé un outil de simulation pour des applications de robots mobiles qui s'appuie sur ces modèles (navigation, planification, coopération, coordination, etc ...) [Bouzid *et al.*, 1999]. Comme première application, nous avons choisi la navigation, qui est primordiale pour n'importe quelle application de robots mobiles.

Notre choix s'est porté sur de telles applications pour deux raisons. La première est que ces applications font partie des activités de recherche de notre équipe, et le développement d'une plate-forme de simulation capable de reproduire le plus fidèlement possible ce qui se passe dans la réalité s'avère fondamental. Ceci va nous éviter d'utiliser le robot réel dans des situations dangereuses, de pouvoir mener des expériences sur des mondes virtuels (comme pour utiliser un nombre de robots supérieur à celui que nous possédons), et enfin de réduire le temps de calcul de certaines applications demandant beaucoup de temps de traitement sur le robot réel. Par exemple, les applications d'apprentissage de comportements pour les robots, peuvent être dangereuses et nécessitent beaucoup de temps de traitement si elles sont effectuées directement sur le robot réel. La seconde raison est que selon Bryson et al., le développement de telles applications forme toujours un défi et un bon cadre pour tester et évaluer les différents mécanismes d'interaction (coordination, coopération, etc ...), théoriques conçus pour des systèmes multi-agents [Bryson *et al.*, 2000].

L'application que nous avons étudiée et mise en œuvre, appelée PIOMAS[8], consiste à simuler les déplacements d'un ensemble de robots dans un environnement structuré. La figure 4.1 montre un exemple d'environnement tel que nous l'envisageons.

Nous décrivons dans ce chapitre une instanciation du modèle que nous avons présenté dans le chapitre précédent. En particulier, nous insistons sur la présentation du modèle d'interaction robot/environnement [Bouzid *et al.*, 2000b]. Enfin, nous détaillons certains

8. *Parallel sImulator of stOchastic MultiAgent Systems*

points nécessaires pour la simulation multi-robot parallèle.

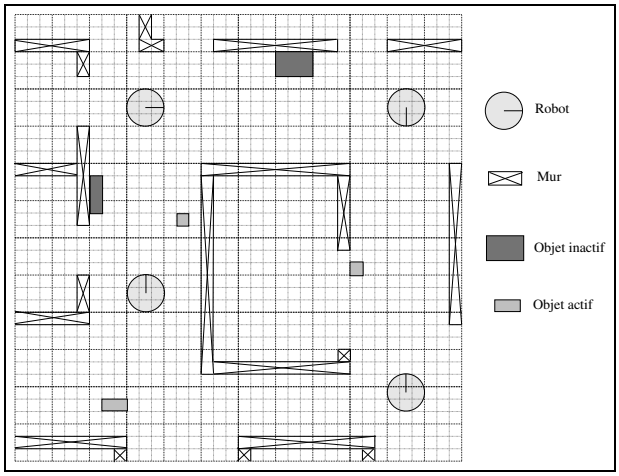

FIG. 4.1 – *Exemple de systèmes à simuler*

4.2 La description du système

Nous avons déjà vu dans la section 3.2.6, que l'environnement doit consister en un ensemble d'états, afin de pouvoir utiliser notre modèle stochastique d'interaction entre l'agent et l'environnement. Nous avons donc décidé de mailler l'environnement physique dans lequel évoluent les robots (comme dans la figure 4.1), qui va consister en un ensemble de cases. L'état global S de notre système sera une image de l'occupation des différentes cases de l'environnement, par les agents et les objets du système. Nous avons aussi choisi un modèle de temps discret à pas constant pour cette première application.

Les agents vont représenter les robots du système à simuler (un agent par robot). Les autres objets actifs Obj_a, qui sont prévus juste pour introduire une certaine dynamique dans le système (i.e. ils adoptent un comportement selon des règles préétablies ou aléatoires), vont représenter les êtres humains, les chariots ou tout autre objet mobile qui peuvent être déplacés par ces derniers, les portes qui s'ouvrent ou se referment périodiquement ou aléatoirement, etc ... Nous les appelons aussi objets mobiles. Les objets inactifs Obj_i, appelés objets inertes, seront les murs, les bureaux, les étagères, ... , qui structurent l'environnement.

Etant donnée la discrétisation du système (modèle d'espace et modèle de temps), nous avons choisi de définir la vitesse d'un agent ou d'un objet mobile en termes de période d'action. C'est-à-dire que chaque agent ou objet actif, génère une influence tout les n cycles, si sa période d'action est égale à n. Il en découle que la différence dans les vitesses d'action, entre les plus lents et les plus rapides robots et/ou objets actifs, est établie en retardant les actions des plus lents.

Chaque objet occupe une ou plusieurs cases de l'environnement, en fonction de sa taille et de celle des mailles. Nous avons choisi un maillage fin pour notre environnement,

de sorte que chaque agent occupe un ensemble de cases à la fois, afin d'être plus précis dans les mouvements des agents au cours de la simulation (voir paragraphe suivant) et dans la résolution des conflits (voir section 4.4.2).

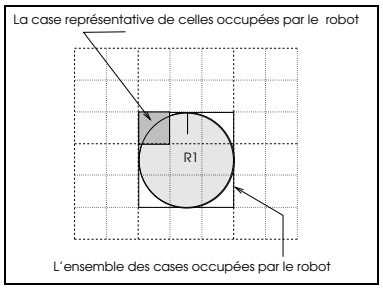

FIG. 4.2 – *L'ensemble des cases occupées par un robot*

Nous avons décidé que la case du coin face-gauche du robot soit la case représentative de celles occupées par l'agent (figure 4.2). De cette façon, nous pouvons varier le nombre de cases occupées par l'agent sans apporter beaucoup de modifications au niveau du simulateur, en prenant toujours comme origine des transitions et des observations la case du coin face-gauche du robot.

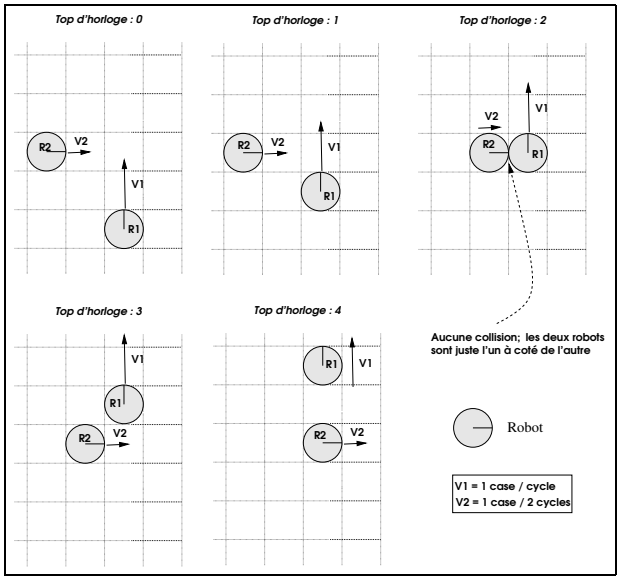

FIG. 4.3 – *Simulation utilisant un modèle d'espace discret : un robot = 1 case*

Afin de pallier un peu aux inconvénients (cités dans la section 2.3.1) de l'utilisation d'un modèle d'espace discret pour de telles applications de robots mobiles, nous avons aussi imposé pour chaque agent de ne se déplacer que d'une case au plus au cours d'un cycle.

En effet, en faisant avancer chaque robot d'un petit pas au plus au cours d'un cycle, nous nous rapprochons d'un modèle d'espace continu. Par conséquent, nous pouvons détecter plus de conflits qui peuvent se produire entre deux agents ou plus, avec ce

genre de modèle d'espace qu'avec un modèle d'espace discret classique où chaque agent occupe toute une case, ou pire encore, où un ensemble d'agents peuvent occuper la même case (comme dans MOBIDYC [Ginot et Le Page, 1998], CORMAS [Bousquet *et al.*, 1998], MICE [Durfee et Montgomery, 1989], etc ...). Par exemple, dans la figure 4.3, nous supposons que deux robots $R1$ et $R2$, occupant chacun une et une seule case, se déplacent selon deux trajectoires perpendiculaires, à des vitesses constantes. $R1$ a une vitesse $V1 = 1$ *case / cycle* et $R2$ a la vitesse $V2 = 1$ *case / 2 cycles*. Nous voyons en déroulant la simulation par cycle, que le robot $R1$ passe juste à coté de $R2$ au second top d'horloge, mais sans collision puisque il est le plus rapide. Alors que dans la figure 4.4, où chaque agent occupe 3x3 cases par exemple, au huitième top d'horloge une collision est détectée entre $R1$ et $R2$. L'inconvénient de ce maillage fin est que le temps de simulation devient plus long, mais la simulation devient plus fine et plus de conflits entre les robots seront détectés, ce qui rend ainsi les résultats plus proches de la réalité. Par conséquent, nous pouvons avoir plus confiance dans les résultats de ce type de simulation ayant des maillages fins, où chaque agent occupe plus d'une case à la fois.

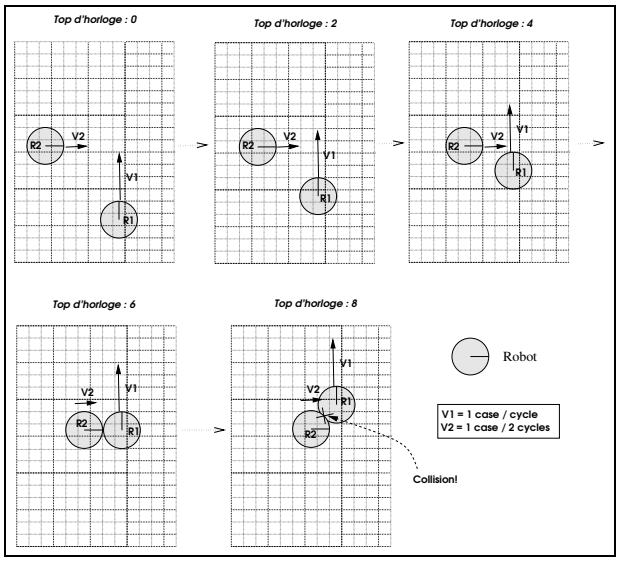

FIG. 4.4 – *Simulation utilisant un modèle d'espace discret fin : un robot* $= 3 \times 3$ *cases*

Nous avons pris comme exemple de robot à modéliser dans cette première version de l'outil de simulation, le NOMAD 200[9] pour lequel nous disposons d'un corpus d'apprentissage (pour ses capteurs et ses effecteurs) établi par notre équipe MAIA, dans les couloirs du laboratoire LORIA.

Le modèle de comportement qui gouverne l'évolution de l'agent est assez simple, consistant juste à naviguer dans les couloirs du laboratoire, tout en évitant les obstacles. Ceci étant dans le but d'offrir une plate-forme de simulation, permettant de tester et de valider différents comportements pour les agents.

9. commercialisé par Nomadics

4.3 Le modèle d'interaction robot/environnement

4.3.1 Au niveau de la perception

Nous avons commencé par modéliser les capteurs infrarouges et ultra-sons du robot NOMAD 200, tel que cela a été établi par O. Aycard au cours de sa thèse [Aycard, 1998], pour la désignation et la reconnaissance des différents objets de l'environnement. Nous avons associé une observation symbolique à chaque objet du système, comme par exemple :

- *une porte* $\longmapsto$ PORTE,
- *un humain* $\longmapsto$ OBSTACLE,
- *un mur* $\longmapsto$ OBSTACLE,
- *l'intersection de deux couloirs* $\longmapsto$ INTERSECTION, ...

L'ensemble des observations a donc le format suivant :
$OBSV$ = {PORTE,OBSTACLE,DEBUT_DE_COULOIR,INTERSECTION, ... }.
Nous avons utilisé trois classes de distance, désignant la distance euclidienne séparant le robot de chaque observation, qui est estimée à partir des mesures de ses capteurs. Ces classes qui sont fonction du rayon de perception du robot, sont les suivantes :

- VN : pour désigner une très courte distance (*Very Near*),
- N : pour indiquer une courte distance (*Near*),
- F : pour signaler une grande distance (*Far*).

L'ensemble des classes de distance est donc : $\mathcal{D}$ = {VN,N,F}.

Les matrices de confusion entre les observations et celle entre les classes de distance, que nous avons utilisées, ont été également déterminées expérimentalement au cours de la thèse de O. Aycard.

Nous avons regroupé les capteurs du robot par face, de sorte que chaque face fournit une observation et une classe de distance à l'agent. Pour commencer, nous avons prévu quatre faces, qui sont la face de devant, de derrière, de gauche et de droite. L'agent robot dispose donc de quatre observations par cycle, chacune accompagnée de sa classe de distance relative, pour générer sa décision.

Nous avons aussi modélisé les capteurs tactiles du NOMAD 200, qui lui permettent de détecter les chocs pendant sa navigation. Ces capteurs sont bien fiables et fournissent au robot des observations sûres. Comme pour les capteurs infrarouges et ultra-sons, nous avons regroupé les capteurs tactiles par face : devant, derrière, gauche et droite. L'agent possède donc quatre autres observations, qui lui indiquent de façon sûre si une collision s'est produite avec un objet de l'environnement ou non.

Dans une seconde version du simulateur, nous avons modélisé de nouveaux capteurs pour le robot, représentant la caméra de laquelle est muni notre NOMAD 200. Nous l'avons utilisée pour la reconnaissance des étiquettes comportant les numéros des bureaux de notre laboratoire. Nous avons donc pu intégré facilement dans notre simulateur ces nouvelles observations, correspondant aux numéros des bureaux, ainsi que leur matrice de confusion.

4.3.2 Au niveau de l'action

Nous avons supposé que chaque robot dispose d'un ensemble A_a d'influences de base, qui sont les suivantes :

- avancer d'un pas vers l'avant FWD,
- tourner de 90 degrés vers la droite TR,
- tourner de 90 degrés vers la gauche TL,
- rester sur place STOP.

Etant donné que nous nous intéressons à des robots homogènes, l'ensemble A des influences de tous les agents sera confondu avec A_a (i.e. $A = A_a = \{\text{FWD,TR,TL,STOP}\}$). Nous avons déterminé pour chaque influence l'ensemble des transitions qui peuvent en résulter, ainsi que les distributions de probabilité correspondantes, à partir d'un corpus d'apprentissages effectués sur le robot réel dans les couloirs de notre laboratoire en utilisant l'algorithme de Baum-Welch [Rabiner, 1989; Koenig et Simmons, 1996].

Transitions en absence d'obstacles

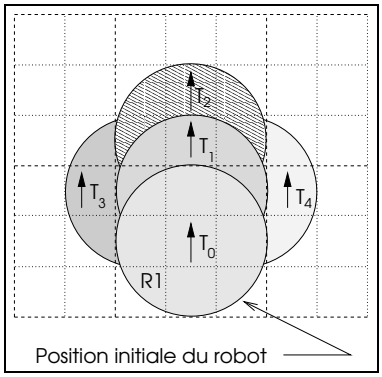

FIG. 4.5 – *Les transitions de l'influence "avancer d'un pas vers l'avant"*

Nous détaillons dans ce qui suit les résultats de l'influence "avancer d'un pas vers l'avant". Nous avons énuméré cinq transitions possibles en absence d'obstacles autour du robot (figure 4.5), qui sont les suivantes :

- T_0 échec de l'action ; le robot reste à la même place,
- T_1 le robot avance exactement d'une case en avant avec succès,
- T_2 le robot glisse et avance de deux cases en avant,
- T_3 le robot était mal-orienté, et par conséquent il a avancé d'un pas en avant tout en se décalant légèrement vers la gauche,
- T_4 le robot était mal-orienté, et par conséquent il a avancé d'un pas en avant tout en se décalant légèrement vers la droite.

Chaque transition T_i a une probabilité d'occurrence égale à P_i, tel que : $\sum_{i=0}^{4} P_i = 1$.

Concernant les autres influences qui peuvent être générées par le robot, qui sont "tourner de 90 degrés vers la droite", "tourner de 90 degrés vers la gauche" et "s'arrêter",

elles se déroulent, par soucis de simplification, avec succès en l'absence d'obstacles autour du robot. Ceci peut s'expliquer par le fait que les incertitudes qui peuvent se produire pendant la rotation du robot, sont négligeables par rapport à l'angle de 90 degrés. C'est aussi le cas pour l'arrêt du robot, d'autant plus qu'il se déplace avec de faibles vitesses.

Transitions en présence d'obstacles

L'ensemble des transitions que nous venons d'énumérer doit changer, lorsque des obstacles se présentent autour du robot, afin d'intégrer les lois du monde et de garder un modèle cohérent. Par exemple, dans la figure 4.6, la transition T_2 est inhibée à cause de l'obstacle frontal, qui se situe à une case du robot (3 positions possibles). Une nouvelle transition T_8 apparaît donc indiquant que le robot a avancé d'un pas vers l'avant, et a subi une collision frontale.

Tout choc subit par le robot lui sera communiqué sous la forme d'une observation sûre, via ses capteurs tactiles. L'agent doit bien sûr utiliser ces observations en priorité pour générer sa prochaine décision, puisqu'elles sont les plus sûres, et elles indiquent l'occurrence d'une collision qui doit être traitée en premier.

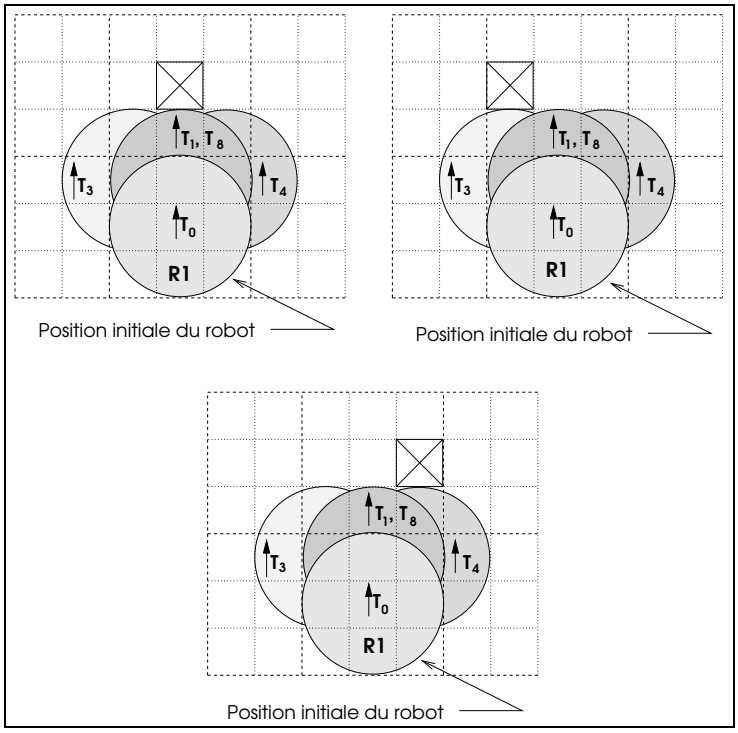

FIG. 4.6 – *Exemple de changement des transitions de l'influence "avancer d'un pas vers l'avant", en présence d'obstacles à une case en face du robot*

Dans la figure 4.7, toutes les transitions sont inhibées, sauf T_0 et T_3. Une nouvelle transition $T7$ apparaît signifiant que le robot a échoué dans son action d'avancement (i.e. il est resté à la même place), et qu'il a subit de plus un choc.

De la même façon dans la figure 4.8, toutes les transitions sont inhibées, sauf T_0 et

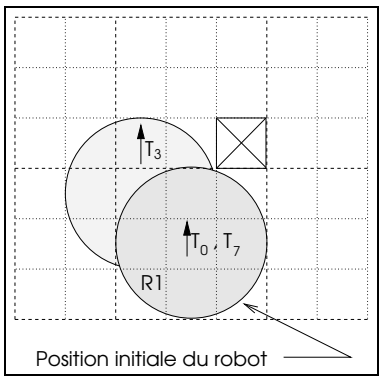

FIG. 4.7 – *Exemple de changement des transitions de l'influence "avancer d'un pas vers l'avant", en présence d'un obstacle juste devant le robot*

T_4. La nouvelle transition T_7, toujours signifiant que le robot a échoué dans son action d'avancement et qu'il a subit de plus un choc, apparaît encore une fois.

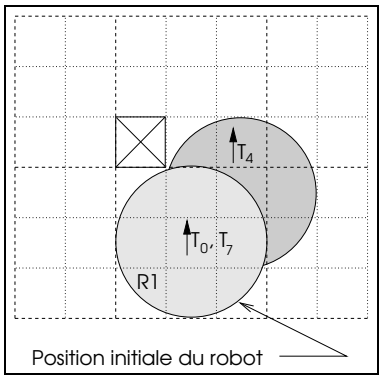

FIG. 4.8 – *Exemple de changement des transitions de l'influence "avancer d'un pas vers l'avant", en présence d'un obstacle juste devant le robot*

Nous avons donc énuméré 9 transitions possibles $\{T_0,T_1,\ldots,T_8\}$ pour l'influence "avancer d'un pas vers l'avant" du robot (i.e. FWD $\longmapsto$ $\{T_0,T_1,\ldots,T_8\}$). Nous avons classé les différentes positions d'obstacles qui peuvent gêner le robot, en fonction des modifications qu'elles entraînent au niveau de la distribution de probabilité de l'ensemble des transitions. Par exemple, dans la figure 4.6 un obstacle qui se présente sur n'importe quelle place des trois positions indiquées, aura le même effet sur la distribution de probabilité de l'ensemble $\{T_0,T_1,\ldots,T_8\}$ de transitions. Nous avons développé un algorithme qui permet de calculer les probabilités d'occurrence $\{P_0,P_1,\ldots,P_8\}$, de cet ensemble de transitions, en fonction des classes des obstacles qui sont présents autour du robot, quelque soit leur nombre, et en fonction des distributions de probabilité en absence d'obstacles apprises expérimentalement. Comme nous l'avons déjà signalé nous devons toujours avoir $\sum_{i=0}^{8} P_i = 1$.

En ce qui concerne l'influence "rester sur place", elle se déroule toujours avec succès même en présence d'obstacle, ce qui est logique. Mais, les influences "tourner de 90

degrés vers la droite" et "tourner de 90 degrés vers la gauche", peuvent parfois échouer à cause des obstacles qui peuvent se trouver autour du robot et gêner sa rotation en conséquence. En effet, le robot possède un bras manipulateur qui peut s'accrocher aux obstacles, qui se trouvent trop près du robot, lors de sa rotation. Les deux influences de rotation, ont donc chacune une probabilité de succès et une probabilité d'échec (i.e. rester dans la même orientation) lorsque des obstacles se présentent autour du robot. Cela veut dire que nous avons : TR $\longmapsto \{T_{R_0}, T_{R_1}\}$ et TL $\longmapsto \{T_{L_0}, T_{L_1}\}$.

4.4 Le simulateur multi-robot PIOMAS

Nous avons décrit dans la section 3.2.6 comment une simulation multi-agent utilisant notre modèle d'interaction se déroule. Nous détaillons dans ce qui suit comment détecter et résoudre les conflits entre les robots et/ou les objets actifs (i.e. la politique), et comment mettre en œuvre leurs influences en conséquence. Nous présentons après quelques détails concernant la simulation parallèle.

4.4.1 La détection de conflits

Dans une application de navigation de robots mobiles, les conflits entre les agents eux mêmes et/ou les objets actifs, reviennent à des conflits spatiaux. En effet, chaque agent ou objet actif génère une influence pour se déplacer, et un conflit peut apparaître lorsque deux entités ou plus décident d'occuper des places dans l'environnement qui se chevauchent.

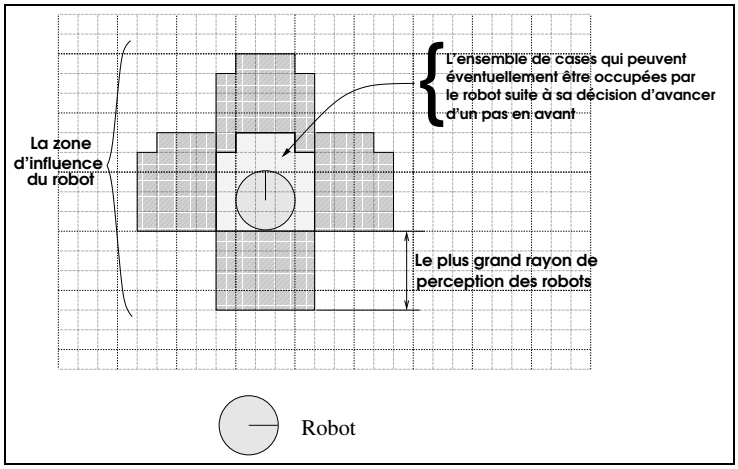

FIG. 4.9 – *Détermination de la zone d'influence d'une entité active ; cas d'un robot dont la décision est d'avancer d'un pas en avant*

Afin de détecter les conflits entre les différentes entités actives du système (agents et objets), nous avons défini la notion de zone d'influence d'une entité. Cette zone est définie en fonction de la position courante de l'entité, de l'ensemble des cases auxquelles elle peut éventuellement transiter (en tenant compte de sa décision), et du rayon de

perception des robots relativement aux quatre faces utilisées (la face de devant, de derrière, de gauche et de droite ; figure 4.9). Dans les exemples que nous avons simulés, les robots étaient homogènes et ils avaient donc les mêmes rayons de perception. Mais, si ce n'était pas le cas, il faut prendre en compte le plus grand rayon de perception des robots, puisque la zone d'influence est déterminée par rapport aux autres agents. En effet, cette zone détermine quelle est la partie de l'environnement dans laquelle si un autre agent se situe, il pourrait percevoir cette entité avant et/ou après l'exécution de son influence.

Après avoir déterminé la zone d'influence de chaque entité active, nous cherchons les conflits entre ces entités en utilisant le principe suivant : deux entités dont les zones d'influences se chevauchent sont supposées être conflictuelles.

4.4.2 La politique de résolution de conflits

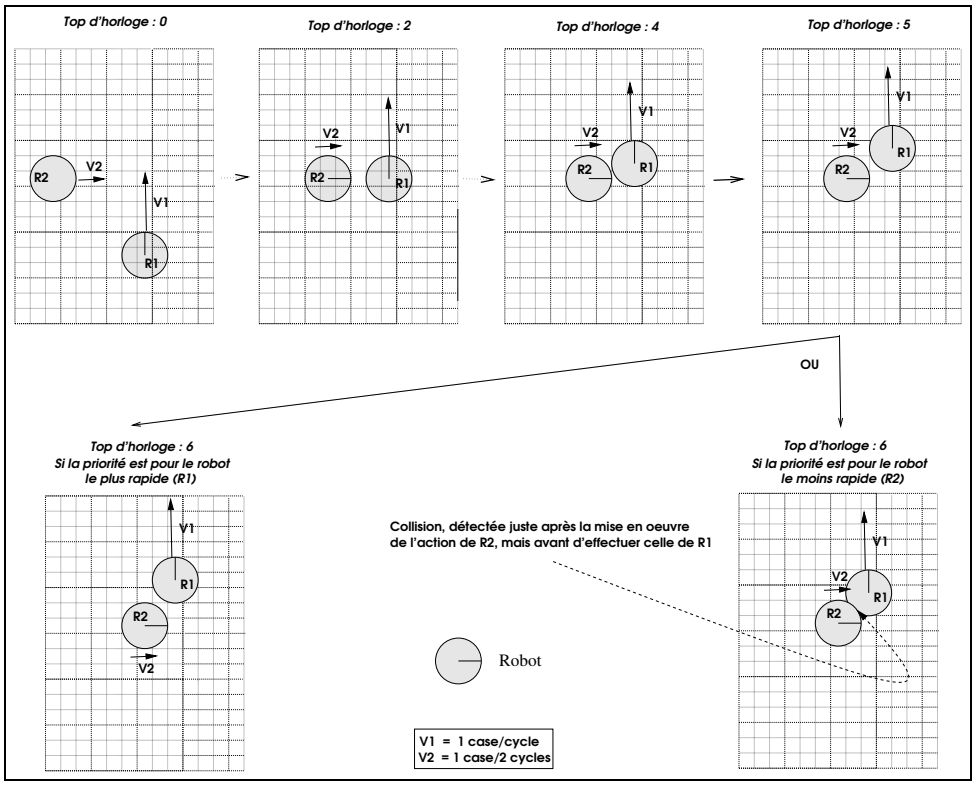

FIG. 4.10 – *Un exemple de cas où si la priorité est donnée au robot le moins rapide, un conflit sera détecté*

Dans la première version de notre simulateur PIOMAS, nous avons choisi une politique simple pour la résolution de conflits entre les différentes entités actives du système, basée sur les priorités. Cette politique est basée sur les points suivants :

Résolution de conflits entre les agents et les objets actifs

Tous les agents sont plus prioritaires que les objets actifs. En effet, nous supposons que les objets actifs sont plus intelligents que les robots, puisqu'ils représentent généralement les êtres humains ou des objets mobiles commandés par ces derniers. Par conséquent, c'est aux objets actifs de céder en cas de conflit afin d'éviter les collisions avec les robots.

Résolution de conflits entre les agents

La résolution de conflit entre agents, consiste à donner la priorité au robot le plus lent. Etant donné que la différence de vitesse entre les agents est établie en retardant les actions des agents les plus lents, ces derniers seront immobilisés pendant un cycle ou plus au cours de la simulation, ce qui n'est pas du tout réaliste. En effet, ces robots devraient agir à petits pas, qui soient plus petits qu'une case de l'environnement. C'est pour cette raison que nous retardons leurs mouvements, le temps que la somme de ces petits pas corresponde à la taille d'une case. Une façon à compenser, dans certains cas, ce retard au niveau de l'application de leurs actions, est de leur donner la priorité en cas de conflit. En effet, cette solution peut parfois nous permettre de détecter plus de collisions qui peuvent se produire en réalité entre les robots, que dans le cas où la priorité sera donnée au plus rapide. La figure 4.10 présente un exemple de situation où cette idée est vérifiée. Nous disposons de deux robots $R1$ et $R2$ qui se déplacent selon deux trajectoires perpendiculaires, à des vitesses constantes (vitesse de $R1 : V1 = 1$ *case / cycle* et vitesse de $R2 : V2 = 1$ *case / 2 cycles*). A partir du cinquième top d'horloge, deux cas de figure se présentent :

- priorité pour le robot le plus rapide : $R1$ avance avec succès, puis $R2$ avance aussi, et aucune collision n'est détectée,
- priorité pour le robot le plus lent : $R2$ avance le premier et heurte le robot $R1$. Une collision sera donc détectée au cours de ce cycle.

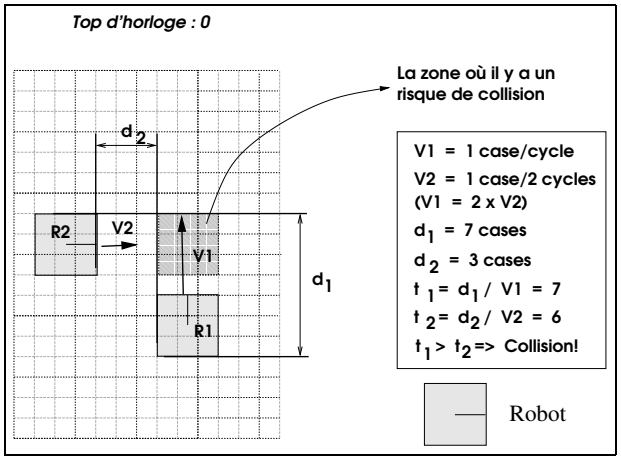

FIG. 4.11 – *Preuve de l'existence d'une collision entre les deux robots $R1$ et $R2$ dans la réalité*

Dans ce cas de figure, la deuxième solution qui donne la priorité au robot le plus lent est plus intéressante. En effet, dans le cas réel où les modèles d'espace et de temps sont continus, une collision se produit entre les deux robots $R1$ et $R2$. Selon la figure 4.11, le robot $R1$ nécessite un temps $t1 = d1/V1 = 7$ *cycles*, pour quitter la zone où il y a un risque de collision. Alors que le robot $R2$, ne nécessite qu'un temps $t2 = d2/V2 = 6$ *cycles* pour atteindre cette zone. $R2$ va donc arriver dans la zone de risque avant que $R1$ ne la quitte en réalité, puisque $t1 > t2$. D'où la collision qui va se produire entre les deux robots, qui peut être détectée en donnant la priorité au robot le plus lent en cas de conflit.

Mais, malheureusement, cette solution peut dans d'autres situations ne pas détecter certains conflits. La figure 4.12 montre le cas inverse du précédent où il y a un conflit à simuler, mais qui n'a pas été détecté puisque la priorité a été attribuée au robot le plus lent.

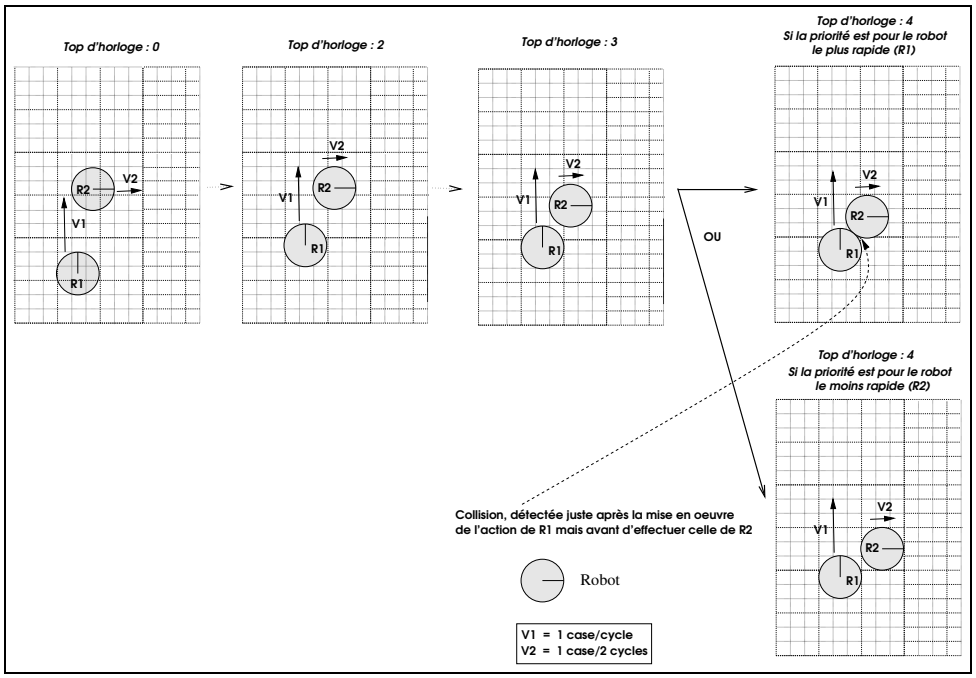

FIG. 4.12 – *Un exemple de cas où si la priorité est donnée au robot le moins rapide, le conflit ne sera pas détecté*

Finalement, nous avons choisi la solution qui consiste à donner la priorité au robot le plus lent, afin de compenser le retard accumulé dans la réalisation de ses actions. Si deux robots en conflit ont la même vitesse d'action, un tirage au sort sera effectué pour attribuer la priorité à l'un d'eux.

Résolution de conflits entre les objets actifs

La résolution de conflit entre les objets actifs peut se baser sur les vitesses d'action, comme pour les robots, comme elle peut se faire de façon aléatoire. En effet, étant donné

que les objets actifs sont juste prévus pour introduire une certaine dynamique et un certain indéterminisme dans le système, les agents doivent développer des comportements robustes indépendamment de ceux des objets actifs.

Enfin, nous remarquons que cette politique de résolution de conflits reste un choix, que nous avons essayé de justifier, et que l'utilisateur de l'outil de simulation peut toujours adopter et implanter d'autres politiques pour son application.

4.4.3 Mise en œuvre des influences conflictuelles

Dans le cas où un conflit est détecté entre plusieurs robots et/ou objets actifs, nous utilisons la politique basée sur les priorités, décrite ci-dessus, pour résoudre le conflit. Par la suite, nous mettons en œuvre l'influence de l'agent le plus prioritaire, en considérant toutes les autres entités comme étant des obstacles. L'ensemble des transitions possibles pour l'influence de cet agent sera donc déterminé exactement de la même façon que dans le paragraphe 4.3.2 précédent. De la même manière, nous exécutons les influences des autres agents un par un, selon l'ordre de priorité décroissant. Une fois tous les agents traités, nous mettons en œuvre les influences des objets actifs également par ordre de priorité décroissant, en fonction des lois du monde.

4.4.4 La simulation parallèle

Nous avons utilisé notre approche, décrite dans les paragraphes 3.3.2 et 3.3.3 pour établir une implantation parallèle de notre simulateur de robots mobiles. Nous rappelons que cette parallélisation est basée sur un mécanisme de double *work-pool*. Durant chaque cycle, nous aurons dans le premier *work-pool* tous les robots et les autres objets actifs du système, formant chacun une tâche. Dans le second *work-pool*, nous aurons tous les robots et les objets qui seront effectivement actifs courant ce cycle (en fonction de leurs périodes d'action), accompagnés de leurs influences. Une tâche de ce dernier *work-pool* sera formée d'un ensemble d'agents et/ou d'objets actifs en conflit. Comme nous l'avons déjà signalé dans le paragraphe 4.4.1, les conflits reviennent à des conflits spatiaux, et leur détection s'effectue selon la méthode indiquée dans le même paragraphe, afin de fusionner les tâches conflictuelles en une seule tâche.

4.5 Conclusion

Nous avons présenté dans ce chapitre le simulateur de navigation de robots mobiles que nous avons conçu, sur la base de nos modèles d'interaction stochastique entre un agent et son environnement, et de simulation parallèle par équilibrage dynamique de charge. Nous avons décrit nos choix concernant, principalement, le modèle d'espace et la politique de détection et de résolution de conflits entre les robots eux-mêmes et/ou les objets actifs du système. Nous avons indiqué comment ces derniers peuvent nous permettre de pallier, dans certain cas, les inconvénients des modèles discrets d'espace et de temps.

L'implantation de ce simulateur parallèle d'applications de robots mobiles, appelé PIOMAS, sera décrite dans le chapitre suivant.

Chapitre 5

Implantation du simulateur de robots mobiles PIOMAS

5.1 Introduction

Nous détaillons dans ce chapitre, l'implantation de notre outil de simulation parallèle sur des machines parallèles multiprocesseurs.

Nous commençons par donner une rapide description des machines parallèles modernes, et plus particulièrement de celles que nous avons utilisées au Centre Charles Hermite (CCH [10]) et à SUPELEC. Par la suite, nous décrivons le langage parallèle que nous avons utilisé et l'architecture logicielle ainsi développée pour notre simulateur, c'est-à-dire les détails techniques de l'implantation du simulateur PIOMAS. Enfin, nous présentons comment utiliser le simulateur PIOMAS ainsi obtenu.

5.2 Les machines parallèles utilisées

De nos jours, la plupart des machines parallèles ont une architecture du type MIMD (voir section 2.4.3), qui veut dire que plusieurs instructions peuvent s'exécuter simultanément sur différent processeurs, traitant des données différentes. Ces machines sont basées sur des processeurs du marché, tels que les processeurs Mips-R10000 ou R12000, mais plusieurs architectures de type MIMD existent. Au départ, il y avait deux grandes familles d'architectures de type MIMD : celles à mémoire partagée et celles à mémoire distribuée.

Au niveau des machines à mémoire partagée (multiprocesseurs), le nombre de processeurs est limitée à environ 20, mais les communications entre processeurs sont rapides et ces machines supportent plusieurs paradigmes de programmation parallèle. Il est possible d'implanter les communications entre les tâches, en utilisant la mémoire partagée ou le passage de messages simulé avec des copies de mémoires. Mais les communications entre les tâches sont généralement plus rapides sur les machines à mémoire partagée, en utilisant le partage de mémoire.

10. http://cch.loria.fr/

Les machines à mémoire distribuée (multi-ordinateurs) supportent un nombre illimité de processeurs. Il existe quelques machines qui en possède plusieurs milliers, communicant via un réseau inter-processeur. Toutes les tâches situées sur différents processeurs doivent communiquer en utilisant ce réseau, et le paradigme habituel de programmation parallèle est celui du passage de message. Il existe quelques réseaux haut débit pour la communication inter-processeur, mais les communications entre les tâches sur les machines à mémoire distribuée demeurent plus lentes que sur les machines à mémoire partagée.

Un nouveau type d'architectures est apparu ces dernières années : les machines à mémoire distribuée virtuellement partagée (DSM[11]) [Protić *et al.*, 1996]. Il supporte à la fois un grand nombre de processeurs (plusieurs centaines), et un espace mémoire global partagé (comme dans les architectures à mémoire partagée). Ces architectures sont basées sur une mémoire physiquement distribuée et sur un support matériel de connexion, qui cache cette distribution et assure un espace d'adressage global virtuel. Ces machines parallèles supportent efficacement les paradigmes de programmation à mémoire partagée.

Nous avons implanté notre outil de simulation de robots mobiles sur une machine parallèle, dont l'architecture est de type DSM : la machine SGI-Origin 2000 du CCH, ayant jusqu'à 64 processeurs[12] de type R10000 à 195 MHz et 24 G-Octets de mémoire centrale. Par la suite, nous avons porté notre simulateur sur un PC parallèle à mémoire partagée classique : une SGI-VWS 540 de SUPELEC, ayant 4 processeurs PIII-Xeon à 450 MHz et 640 M-Octets de mémoire centrale. La première machine est un gros ordinateur parallèle, très puissant et cher (environ 1 million de $). La seconde est une petite machine parallèle, moins puissante, mais beaucoup moins chère (environ 10 mille $). Cette dernière machine est aussi intéressante pour deux raisons. La première est qu'elle est plus facilement accessible pour de petites équipes de recherche. La seconde est qu'elle fonctionne sous *Windows-NT* et nous avons voulu tester la portabilité de notre simulateur multi-agent parallèle sur différents systèmes d'exploitation. Nous signalons que la version fonctionnant sous *Windows-NT* de notre simulateur, fonctionne aussi sur un ordinateur mono-processeur classique (sous *Windows-NT*) sans aucune restriction. D'où la possibilité de l'utiliser facilement sur un PC.

5.3 Le langage de programmation parallèle utilisé

Nous avons vu dans la section 2.4.4, qu'il y a principalement quatre familles de langages parallèles. Nous avons décidé d'utiliser le langage parallèle ParCeL-3 [Vialle *et al.*, 2000], développé à SUPELEC et faisant partie de la classe des langages ayant un modèle de programmation parallèle. Comme déjà signalé, la version actuelle de ParCeL-3 n'offre que la machine virtuelle du modèle de calcul et une bibliothèque de fonctions pour la gestion des cellules. ParCeL-3 a été choisi, puisque d'une part il a été conçu pour les applications d'IA, en particulier les systèmes multi-agents et les réseaux de neurones, et d'autre part parce qu'un programme ParCeL-3 a un fonctionnement cyclique, qui est

11. Distributed Shared Memory
12. Les machines SGI-Origin 2000 peuvent avoir jusqu'à 1024 processeurs.

bien adapté au modèle de temps choisi pour notre simulateur. Enfin, en utilisant ParCeL-3, nous n'avons pas à nous soucier des fonctionnalités de parallélisme de bas niveau, et notre simulateur sera portable sur différentes architectures et systèmes d'exploitation, puisque ParCeL-3 ne nécessite qu'une bibliothèque de *multithreading* et des outils de synchronisation élémentaires, comme les sémaphores, pour être implanté.

5.3.1 Le modèle de calcul de ParCeL-3

Le modèle de calcul de ParCeL-3 est basé sur sept concepts. Un programme ParCeL-3 se compose principalement de *cellules* qui s'exécutent en parallèle, qui comportent des *règles d'exécution*, et des *horloges* pour la gestion de leur exécution. Ces cellules communiquent à travers une *mémoire partagée* globale, des *canaux de communication* dédiés et l'envoi de messages dans des *boîtes aux lettres*. Enfin, un programme ParCeL-3 s'exécute en répétant un *cycle de base*, qui consiste en le calcul des cellules, l'évolution du réseau de cellules, la mise à jour des canaux de communication et des boîtes aux lettres.

Les cellules

Une cellule de ParCeL-3 est une sorte de processus ultra-léger, et toutes les cellules s'exécutent concurremment. Chaque cellule a un type bien déterminé, qui doit être spécifié par le concepteur du programme. Tous les types cellulaires sont définis au moment de la compilation, et chaque cellule est créée dynamiquement à partir d'un type cellulaire durant l'exécution du programme. Il n'existe qu'une seule cellule au début de l'exécution, qui crée de nouvelles cellules, qui peuvent à leur tour en créer d'autres (figure 5.1). Une cellule a ses propres variables locales, quelques règles d'exécution communes à toutes les cellules du même type cellulaire, et une horloge désignée quand la cellule est créée, et qui peut être partagée avec d'autres cellules (figure 5.2).

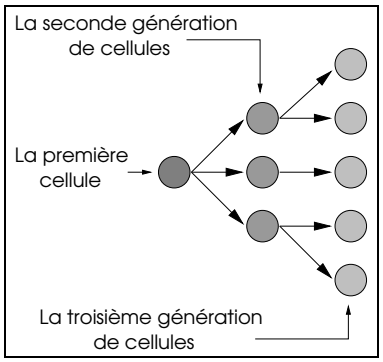

FIG. 5.1 – *Création d'un réseau de cellules*

Les règles d'exécution

Chaque règle d'exécution comporte deux types d'instructions : celles pour effectuer du calcul classique, et celles pour déterminer la prochaine règle à exécuter et le prochain

top d'horloge qui va la déclencher. De plus, il est possible de choisir dynamiquement la prochaine règle à déclencher, en fonction des variables locales par exemple. Les instructions de calcul classique d'une règle incluent la gestion des variables locales, l'accès à la mémoire globale partagée, la lecture et l'écriture dans les canaux de communication, l'exécution des requêtes d'évolution du réseau de cellules (création de cellules, connexion de canaux, destruction de cellules, ...), l'envoi de messages à d'autres cellules et la lecture de la boîte aux lettres locale.

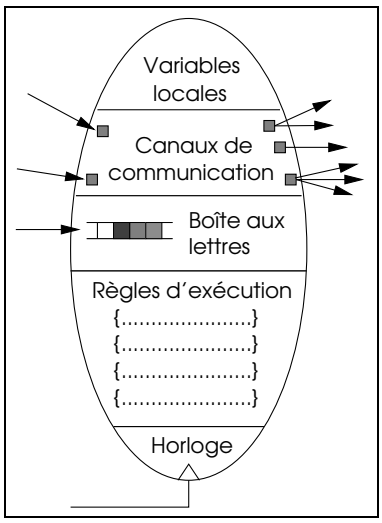

FIG. 5.2 – *Les composantes d'une cellule classique*

Les horloges

Un programme ParCeL-3 s'exécute selon un mode cyclique. Chaque cycle n'a qu'une seule étape de calcul cellulaire, et chaque cellule ne peut être activée qu'une seule fois au cours de cette étape. Il existe donc une *horloge de base* dans le modèle de calcul de ParCeL-3, qui est l'horloge définie par le mécanisme de fonctionnement cyclique, suffisante pour exécuter les cellules des applications simples et régulières. Mais, lorsque les applications deviennent plus complexes, en ayant des parties nécessitant des ordonnancements différents, il n'est pas pratique d'utiliser une seule horloge. Aussi, ParCeL-3 permet de définir autant d'horloges que nécessaires, chacune ayant ses propres caractéristiques.

Toutes les horloges d'un programme ParCeL-3 sont déclarées statiquement, avant la compilation. La déclaration d'une horloge i consiste à définir son identificateur clk_i, ainsi que sa période $period_i$. Une fois démarrée, l'horloge clk_i génère un top à chaque cycle ParCeL-3, et les tops sont cycliquement numérotés de 1 à $period_i$. Cette numérotation est employée par chaque règle d'exécution de chaque cellule associée à clk_i, pour déterminer la prochaine règle à exécuter et le top d'horloge à utiliser pour la prochaine exécution.

Au début de l'exécution toutes les horloges sont arrêtées, excepté la première horloge (l'horloge *principale*) qui est attachée automatiquement à la première cellule (la cellule

principale). Toutes les autres horloges doivent être explicitement démarrées par une instruction de lancement d'horloge, exécutée depuis une règle de cellule. Par exemple, une méthode classique est de créer une nouvelle cellule, de préciser son horloge et de la lancer depuis la même règle d'une autre cellule (sa cellule mère). Une horloge peut être démarrée pour un nombre fini ou infini de périodes, et ce temps de fonctionnement est nommé le temps de *mission* de l'horloge. De plus, l'horloge d'une cellule fille peut fonctionner en parallèle de celle de sa mère : la cellule mère continue à fonctionner parallèlement à sa fille (diagramme `h1-clk` de la figure 5.3), ou elle peut suspendre l'horloge de sa mère jusqu'à finir sa mission : toutes les cellules dépendantes de l'horloge de la cellule mère seront suspendues jusqu'à ce que la cellule fille termine ses traitements (diagramme `h2-clk` de la figure 5.3). Cette dernière possibilité est nommée "mode inséré". Enfin, les horloges peuvent être explicitement suspendues avant la fin de leurs missions, et être relancées pour terminer leurs missions ou réinitialisées. Dans ce dernier cas, l'horloge réinitialisée est arrêtée et elle commence une nouvelle période et une nouvelle mission quand elle est relancée (c'est-à-dire qu'elle oublie son ancienne mission).

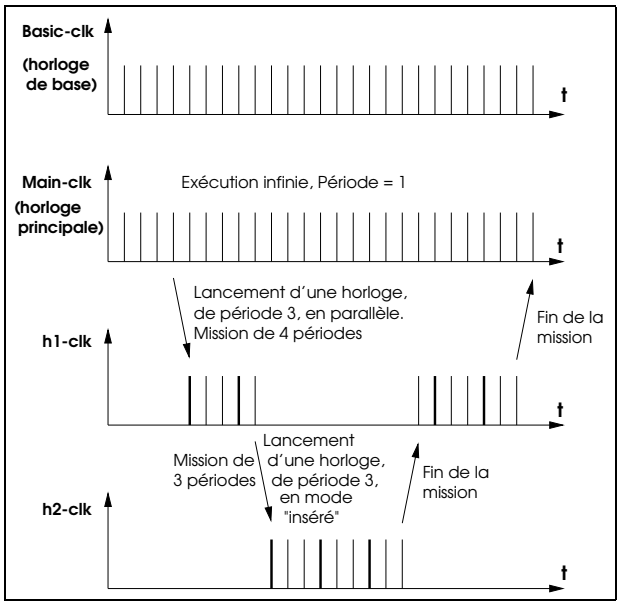

FIG. 5.3 – *Exemple d'exécution d'horloges*

La mémoire partagée

Toutes les cellules ParCeL-3 possèdent un espace d'adressage privé, où elles stockent leurs variables locales, et partagent un espace d'adressage global. Cette mémoire partagée est très utile pour construire des simulateurs multi-agents où les agents évoluent dans un environnement global partagé.

Les canaux de communication

Les canaux de communication permettent de mettre en œuvre des connexions de longues durées entre les cellules, comme les connexions entre les neurones d'un réseau de neurones artificiel. Ces canaux peuvent aussi servir pour établir certains types de communications entre des agents. Le programmeur doit employer les tops d'horloge pour écrire dans les canaux de sortie et lire depuis les canaux d'entrée à différents cycles, et être sûr d'éviter les conflits d'accès. Tout nouveau message écrit dans un canal de communication écrase le message précédent. Enfin, les canaux de communication peuvent être déclarés statiquement ou assignés dynamiquement.

Les boîtes aux lettres

Chaque cellule ParCeL-3 possède une boîte aux lettres, qui peut recevoir des messages de toute autre cellule et envoyer des messages à n'importe quelle cellule dont elle connaît l'identifiant. Une cellule peut recevoir n'importe quel nombre de messages au cours d'un ou plusieurs cycles, et sa boîte aux lettres garde ces messages jusqu'à ce qu'ils soient lus et effacés. Ces boîtes aux lettres peuvent à leur tour servir pour faire communiquer un ensemble d'agents.

5.3.2 L'implantation de ParCeL-3

La machine virtuelle de ParCeL-3 a été implantée en utilisant des bibliothèques de processus légers (*threads*) de deux systèmes différents. Le premier est *Irix*, installé sur la machine SGI-Origin 2000 du CCH, et l'implantation a été effectuée en utilisant les *threads* natifs d'*Irix*. Le second système d'exploitation est *Windows-NT*, dont on a aussi utilisé les *threads* natives pour construire une autre version de ParCeL-3, qui tourne aussi bien sur une machine quadri-processeur à mémoire partagée (SGI-VWS 540 de SUPELEC), que sur un PC classique mono-processeur. Le portage de ParCeL-3 a été facilement et rapidement effectué du premier système d'exploitation au second.

D'autres équipes mènent des recherches pour donner une vision d'un système à espace d'adressage uniforme d'une grappe de machines inter-connectées, c'est-à-dire pour donner l'illusion que la grappe est une seule machine parallèle à mémoire partagée [Anderson *et al.*, 1995]. Comme exemple de travaux, nous citons le système GLUnix, implanté au dessus d'un système Unix [Ghormley *et al.*, 1998], et le système Gobelins destiné aux applications qui s'exécutent sur une grappe de PC [Lottiaux et Morin, 2000; Morin *et al.*, 2000]. Nous pensons que ParCeL-3 peut être facilement porté sur un de ces systèmes, offrant la vision d'une machine parallèle virtuelle à mémoire partagée, d'une grappe de machines. De tels ensembles de machines inter-connectées sont disponible dans presque tous les laboratoires de recherche de nos jours, et sont donc facilement accessibles à toutes les équipes.

En conclusion, notre simulateur parallèle de robots mobiles peut être utilisé sur n'importe quelle machine parallèle à mémoire partagée réelle ou virtuelle, puisqu'il suffit de porter la bibliothèque ParCeL-3 sur la nouvelle architecture, ce qui ne semble pas poser de problèmes jusqu'ici. Ceci n'exclut pas, évidemment, l'utilisation de notre simulateur

sur une machine mono- processeur classique. L'efficacité de tels portages dépend de l'efficacité du partage de mémoire de la nouvelle architecture parallèle.

5.4 L'implantation du simulateur PIOMAS

Nous décrivons dans cette partie les différents choix que nous avons effectués pour implanter notre outil de simulation PIOMAS, concernant son architecture cellulaire, au sens de ParCeL-3, et les structures de données des deux *work-pools* [Lester, 1993], ainsi que leurs algorithmes de gestion.

5.4.1 L'architecture cellulaire de PIOMAS

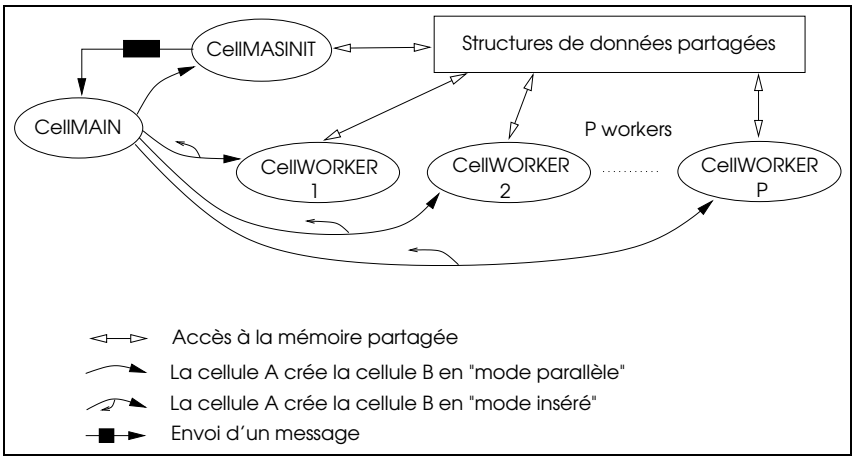

FIG. 5.4 – *L'architecture du réseau de cellules ParCeL-3, utilisée pour notre simulateur*

Dans notre application, toutes les cellules sont créées par la cellule principale `cellMAIN` (figure 5.4). Cette cellule commence par créer la cellule `cellMASINIT`, pour la création et la configuration de tout le système multi-agent, à savoir l'environnement, les agents, les différents objets, ... , spécifiés par l'utilisateur. L'horloge de la cellule `cellMASINIT` sera lancée en parallèle avec celle de la cellule `cellMAIN`. Une fois l'initialisation terminée, la cellule `cellMASINIT` envoie un message à la cellule principale et puis se suicide. De cette façon, l'initialisation peut prendre n'importe quel nombre de cycles, et la cellule `cellMAIN` peut continuer à s'exécuter en parallèle pour, par exemple, signaler à l'utilisateur que l'initialisation est en cours et que les cycles s'écoulent normalement. Notre programme sera ainsi extensible et facile à suivre durant les phases de mise au point et d'évolution des routines d'initialisation.

Quand la cellule principale reçoit le message de fin d'initialisation, elle crée les P[13] cellules `cellWORKER` en "mode inséré". Une fois les initialisations finies et les *workers* lancés, le fonctionnement du programme est beaucoup plus cyclique et planifié que pendant

13. C.f. section 3.3.3

les initialisations : on exécute les algorithmes du *work-pool* et des mécanismes de perception et d'action stochastiques. On ne risque pas d'être interrompu par un manque de mémoire, comme dans l'initialisation. La cellule `cellMAIN` n'a donc plus rien à faire et la création des cellules *workers* en "mode inséré" s'avère suffisant. Le nombre P de ces cellules est spécifié par l'utilisateur aussi, et doit être égal au nombre de processeurs utilisés. Les cellules `cellWORKER` seront réparties à raison d'une par processeur.

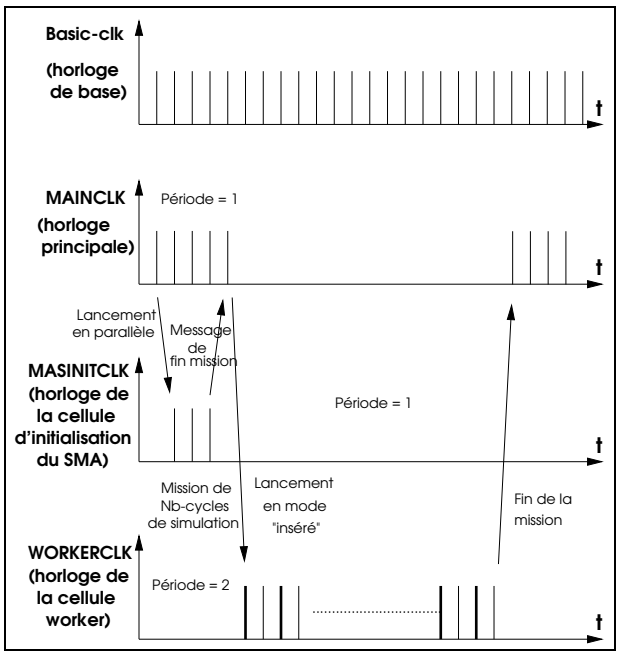

FIG. 5.5 – *Les exécutions des horloges de notre outil de simulation*

Nous avons prévu trois horloges pour la gestion des trois types de cellules dont nous disposons. La première est évidemment l'horloge principale qui a une période égale à 1 top, et qui est lancée automatiquement à l'infini (figure 5.5, l'horloge *MAINCLK*). La seconde, appelée l'horloge *MASINITCLK*, est attribuée à la cellule `cellMASINIT`, qui a aussi une période égale à 1 top, mais qui ne sera lancée finalement que pour une seule période ; juste le temps nécessaire pour initialiser le système multi-agent. A ce top d'horloge viennent s'ajouter automatiquement une période d'initialisation de la cellule fraîchement créée, et une période de destruction après son suicide (mécanisme général de gestion des cellules de ParCeL-3). L'horloge *MASINITCLK* sera lancée en parallèle avec l'horloge principale. La troisième horloge, nommé *WORKERCLK*, permettant de gérer les cellules de type `cellWORKER`, a une période égale à 2 tops. Elle sera lancée en "mode inséré" pour un temps de mission égal au nombre de cycles de simulation spécifié par l'utilisateur. Durant cette période, la cellule `cellMAIN` sera suspendue jusqu'à ce que les cellules `cellWORKER` finissent la simulation. Nous avons choisi une période de 2 tops pour l'horloge *WORKERCLK*, puisque durant un cycle chaque *worker* commence par traiter les tâches du premier *work-pool* durant le premier top d'horloge, puis passe au traitement des tâches du second *work-pool* au cours du second top. Le passage du traitement d'un *work-pool* à l'autre passe, d'un point de vue conceptuel, par une barrière de

synchronisation de tous les *workers*. Dans l'implantation en ParCeL-3 cette barrière est réalisée implicitement par le fonctionnement cyclique de ParCeL-3, et la synchronisation des règles des cellules sur différents tops d'horloge.

5.4.2 Implantation des work-pools

Etant donné que la simulation parallèle de l'ensemble des robots est basée sur le mécanisme de double *work-pool* que nous avons proposé, nous présentons dans cette partie les structures de données que nous avons choisies pour les deux *work-pools*, ainsi que leurs algorithmes de gestion.

La structure du premier work-pool

Nous avons signalé dans la section 3.3.3 que le premier *work-pool* contient initialement tous les agents et les objets actifs du système à simuler, représentant des tâches basiques. Pour construire le premier *work-pool*, nous avons utilisé la liste des robots, celle des objets mobiles, ainsi qu'une structure de données permettant d'encapsuler un robot ou un objet mobile pour le transformer en une tâche (figure 5.6). Les tâches sont donc construites dynamiquement à partir des composants de la liste des robots, et puis à partir de la liste des objets mobiles une fois celle des robots est épuisée.

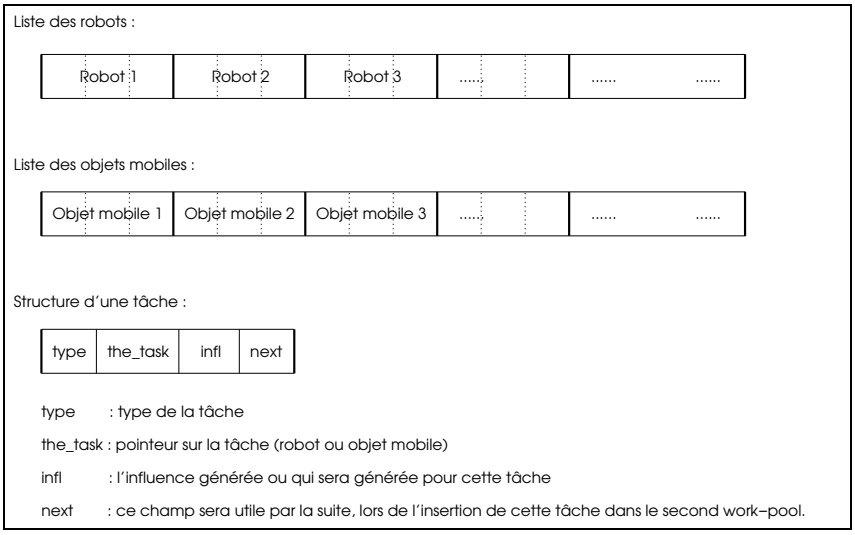

FIG. 5.6 – *Les structures constituant le 1er work-pool*

La structure du second work-pool

Une tâche du second *work-pool* est formée d'un ensemble de robots et/ou d'objets mobiles en conflit. Nous avons donc décidé qu'une tâche sera constituée d'une liste de tâches élémentaires, représentant chacune un robot ou un objet mobile. Par conséquent, le second *work-pool* sera composé de listes de tâches élémentaires (figure 5.7).

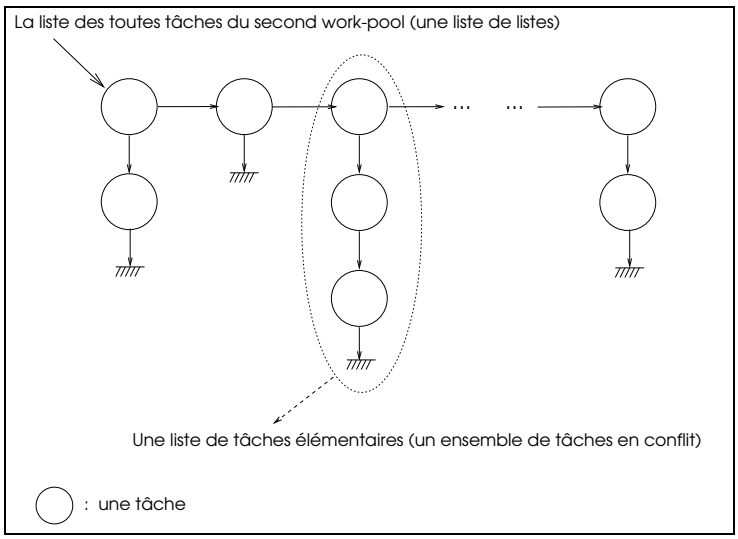

FIG. 5.7 – *La structure logique du second work-pool*

Etant donné que le nombre maximum de tâches qu'on peut avoir au cours d'un cycle est connu d'avance (égal à la somme des nombres d'agents et d'objets mobiles), nous avons décidé d'utiliser une structure de données statique pour l'implantation de cette liste de tâches élémentaires, afin de réduire le temps d'accès et celui de simulation en conséquence. Cette liste consiste alors en un tableau de tâches élémentaires avec des liens entre les différents éléments formant une seule tâche finale (figure 5.8).

Le premier top d'horloge d'une période de cellWORKER

Nous rappelons qu'un ensemble de processus *workers* traitent les tâches du *work-pool* en parallèle. Au premier top d'horloge, les *workers* récupèrent les tâches du premier *work-pool*, les traitent et les insèrent si nécessaire dans le second *work-pool*. L'algorithme général est le suivant :

```
DEBUT
  encore_une_tache  <--  VRAI
  TANTQUE (encore_une_tache) FAIRE        // Tant qu'il y a des tâches dans
                                          // le premier work-pool.
    tache  <--  Recuperer_tache_wpl_1()   // On récupère une tâche.
    CAS (tache.type)
       ROBOT :                            // Si c'est un robot,
           SI Actif(tache)                // et s'il est actif au cours du
           ALORS                          // cycle courant,
             Generer_infl_robot(tache)    // on génère son influence et
             Inserer_tache_wpl_2(tache)   // on l'insère dans le 2nd work-pool.
           FSI
       MOBILE_OBJ :                       // Si c'est un objet mobile,
           SI Actif(tache)                // et s'il est actif au cours du
           ALORS                          // cycle courant,
             Generer_infl_m_obj(tache)    // on génère son influence et
             Inserer_tache_wpl_2(tache)   // on l'insère dans le 2nd work-pool.
```

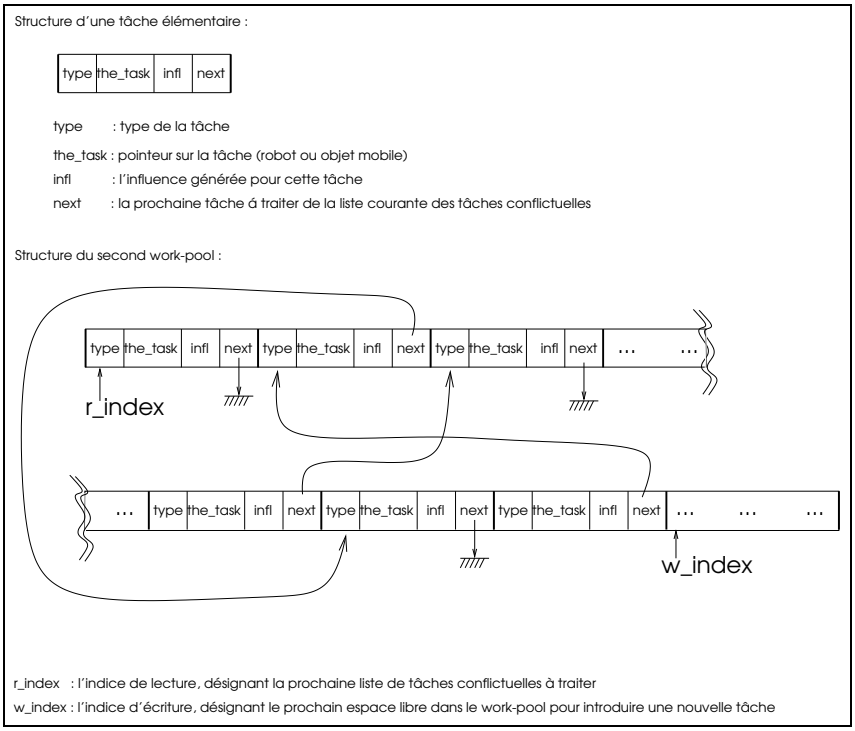

FIG. 5.8 – *La structure finale du second work-pool*

```
          FSI
       VIDE :
          encore_une_tache  <--  FAUX
    FCAS
  FTQUE
FIN
```

Nous décrivons ci-dessous les étapes relatives à la récupération d'une tâche du premier *work-pool*, et de son insertion dans le second *work-pool*:

- récupération d'une tâche du premier *work-pool*: dans cette étape, le *worker* récupère un indice de lecture parcourant la liste des robots, puis celle des objets mobiles. Cette indice est protégé par un mécanisme d'exclusion mutuelle, puisqu'il est accédé simultanément par tous les *workers*. Le robot ou l'objet mobile correspondant à cet indice est encapsulé dans une structure de type tâche, et est renvoyée au processus *worker*. Son algorithme est le suivant :

```
FONCTION Recuperer_tache_wpl_1() : wpl_tache
DEBUT
  EntreeMutex(MutexIndex)             // Récupération de l'indice de lecture
  lo_ind  <--  index                  // "index", partagé par tous les workers
  index   <--  index + 1              // et protégé par un mécanisme
                                      // d'exclusion mutuelle en conséquence.
  SI (lo_ind = 0)
  ALORS                               // Remise à 0 des indices de lecture
    r_index  <--  0                   // et d'écriture du 2nd work-pool,
```

```
    w_index  <--  0                    // par le premier worker accédant le
  FSI                                  // 1er work-pool.
  SortieMutex(MutexIndex)
  SI (lo_ind < Nb_Robot)
  ALORS
    tache.type      <--  ROBOT
    tache.the_task  <--  &(robots[lo_ind]) // On mémorise l'adresse du robot
                                         // à traiter (un pointeur).
  SINON
    SI (lo_ind < Nb_Robot + Nb_Mobile_obj)
    ALORS
      tache.type      <--  MOBILE_OBJ
      tache.the_task  <--  mobile_objs[lo_ind-Nb_Mobile_obj]
                                       // On mémorise l'adresse de l'objet
                                       // mobile à traiter (un pointeur).
    SINON
      SI (lo_ind = Nb_Robot + Nb_Mobile_obj + Nb_Wk - 1)
      ALORS
         index   <--   0               // Remise à 0 de l'index de lecture du
      FSI                              // 1er work-pool.
      tache.type      <--  VIDE        // Indiquer qu'il n'y a plus de tâches.
    FSI
  FSI
  tache.next  <--  NIL
  RETOURNER(tache)
FIN

Lexique :
  MutexIndex : MUTEX                   // Un sémaphore d'exclusion mutuelle.
```

- insertion d'une tâche dans le second *work-pool*: dans cette phase le *worker* insère la tâche récupérée, après avoir généré son influence, dans le second *work-pool*. Mais il doit vérifier s'il y a un conflit entre cette tâche et celles déjà présentes dans le second *work-pool*. Il parcourt donc les listes de toutes les tâches qui y sont et teste l'existence d'un éventuel conflit. Si c'est le cas, les tâches conflictuelles doivent former une seule liste.

  Nous avons appelé cette étape lors de la présentation de notre modèle de simulation parallèle dans la section 3.3.3, la phase de combinaison des influences des tâches. Nous rappelons que nous avons décidé dans la même section 3.3.3, que les processus *workers* vont effectuer les combinaisons des influences en séquentiel. Chaque *worker* aura donc un accès exclusif à toutes les listes de tâches élémentaires existantes dans le second *work-pool*, afin de garder leurs cohérences. Nous avons déjà signalé que ceci va avoir un impact négatif sur les performances parallèles de notre simulateur.

  Nous remarquons qu'étant donné que le test de présence de conflit doit porter sur toutes les listes de tâches élémentaires du second *work-pool*, les performances parallèles de notre simulateur peuvent aussi diminuer puisqu'on perd facilement le contenu de la mémoire cache à cause de la non-localisation des données.

  Nous nous sommes focalisés alors sur ce problème, d'un point de vue technique, afin de lui trouver une implantation parallèle. L'idée que nous avons eu est de séparer les deux étapes de détection de conflits et celle de fusion de tâches conflictuelles. En

effet, la phase de détection de conflits peut s'effectuer totalement en parallèle, sans contrainte. Alors que celle de fusion nécessite une certaine synchronisation entre les processus *workers*, pour que chaque ensemble de tâches conflictuelles forme une seule liste correspondant à une seule grande tâche.

La solution consiste à prévoir au cours de la phase de détection de conflits, une matrice dont les lignes et les colonnes correspondent aux numéros des tâches (leurs ordres d'insertion dans le second *work-pool*), pour indiquer les différents conflits entre les tâches. Il faut donc prévoir une matrice dont le nombre de lignes est égal au nombre de colonnes, qui soit égal au nombre de tâches de l'étape de simulation en cours (la matrice 5.1 en est un exemple).

Ceci revient à définir un graphe, dont les sommets représentent les tâches et les arcs représentent les conflits. La matrice que nous construisons correspond alors à la matrice de connexité du graphe. La construction des listes de tâches conflictuelles, revient à la recherche des composantes connexes du graphe, puisque deux tâches (i.e sommets) reliées par un arc forment une même tâche, et par conséquent toutes les tâches qui appartiennent à un même chemin (i.e. à une même composante connexe) doivent former une même liste de tâches.

Il existe dans la littérature des algorithmes parallèles pour la recherche des composantes connexes d'un graphe [JáJá, 1992; Karp et Ramachandran, 1990]. Certains de ces algorithmes ont une complexité en temps égale à $O(log(n))$ (n étant le nombre de sommets), comme ceux de Awerbuch [Awerbuch et Shiloach, 1987] et Shiloach [Shiloach et Vishkin, 1981], mais sur un nombre de processeurs égal à $O(n+m)$ (m étant le nombre d'arcs du graphe) ; c'est à dire sur un nombre de processeurs fonction de la taille du problème, et non pas sur un nombre de processeurs constant, disponible sur la machine utilisée.

Les tâches	T_1	T_2	...	T_{n-1}	T_n
T_1	-	1	...	1	0
T_2		-		0	1
...			...		
T_{n-1}				-	0
T_n					-

TAB. 5.1 – *Un exemple de matrice indiquant les différents conflits entre les tâches. 0 : Absence de conflit. 1 : présence d'un conflit*

Nous comparons maintenant cette solution à la première que nous avons proposée, celle où les phases de détection de conflit et de fusion se font en une seule étape, mais en donnant un accès exclusif à chaque *worker* au second *work-pool*. Nous remarquons que la deuxième solution demande un nombre de processeurs excessif (en $O(n+m)$) pour être exécutée en un temps égal à $O(log(n))$. Elle introduit aussi une étape supplémentaire de synchronisation des processus *workers*, entre la phase de détection de conflits (remplissage de la matrice) et celle de fusion des tâches. Ceci va induire une certaine perte de temps au moment de la simulation. Nous ajoutons à ceci la perte de temps provenant du parcourt en deux temps de la matrice (détection, puis fusion).

Finalement, mous pensons que la seconde solution, basée sur la matrice de conflits, peut être plus efficace en temps d'exécution que la première. Mais nous pensons aussi que la différence en performance ne sera pas très importante vu les inconvénients que nous venons de citer pour la deuxième solution, notamment le nombre de processeurs lié à la taille du problème.
Pour cette raison, et afin de ne pas compliquer davantage l'algorithme de simulation que nous avons proposé, nous avons choisi d'implanter la première solution pour la détection de conflits et la fusion des tâches dans le second *work-pool*.

```
PROCEDURE Inserer_tache_wpl_2(tache : wpl_tache)
DEBUT
  EntreeMutex(MutexWIndex)          // Récupération de l'indice d'écriture
  wind     <--  w_index             // dans le 2nd work-pool, "w_index",
  w_index  <--  w_index + 1         // partagé par tous les workers
                                    // et protégé par un mécanisme
                                    // d'exclusion mutuelle en conséquence.
  Copier_tache(wpl_2[wind], tache)  // Copie de la tâche dans le work-pool.
  EntreeMutex(MutexRIndex)          // Récupération de l'indice de lecture
                                    // dans le 2nd work-pool, "r_index",
                                    // partagé par tous les workers
                                    // et protégé par un mécanisme
                                    // d'exclusion mutuelle en conséquence.
  SortieMutex(MutexWIndex)          // Libération de l'indice d'écriture,
                                    // afin de permettre à un autre worker
                                    // d'insérer une autre tâche en parallèle.
  Detect_Confl_Fusion(wpl_2, r_index, wind) // Détecter les conflits et
                                    // fusionner les tâches conflictuelles.
  SortieMutex(MutexRIndex)
  EntreeMutex(MutexWWC)             // Incrémentation du nombre de tâches
  wwc  <--  wwc + 1                 // en attente (variable globale partagée
  SortieMutex(MutexWWC)             // par tous les workers).
FIN

Lexique :
  r_index, w_index : ENTIER         // Indices de lecture et d'ecriture
                                    // dans le second work-pool.
  wwc : ENTIER                      // Nombre de tâches en attente dans le
                                    // 2nd work-pool.
  wpl_2 : Liste de wpl_tache        // Le second work-pool.
  MutexWIndex : MUTEX               // Un sémaphore d'exclusion mutuelle
                                    // pendant l'écriture (2nd work-pool).
  MutexRIndex : MUTEX               // Un sémaphore d'exclusion mutuelle
                                    // pendant la lecture (2nd work-pool).
  MutexWWC : MUTEX                  // Un sémaphore d'exclusion mutuelle
                                    // pour la protection du compteur wwc.
```

Nous remarquons que l'appel du mécanisme d'exclusion mutuelle $EntreeMutex(MutexRIndex)$ pour la récupération de l'indice de lecture dans le second *work-pool*, avant la libération de celui d'écriture $MutexWIndex$, permet au processus *worker* courant d'être sûr qu'il va tester l'existence de conflit entre la tâche qu'il insère

et toutes celles déjà présentes dans ce *work-pool*.

Une fois l'indice de lecture récupéré, le processus *worker* libère l'indice d'écriture ($SortieMutex(MutexWIndex)$) afin de permettre à un autre processus *worker* d'insérer une nouvelle tâche dans le second *work-pool* en parallèle, et de gagner ainsi un peu de temps au niveau de la simulation.

Nous remarquons aussi qu'il n'y a aucun risque d'interblocage entre les processus *workers* utilisant ce mécanisme, puisque la récupération des tâches du second *work-pool* (besoin de l'indice $MutexRIndex$), en vue de leurs traitements, ne s'effectue qu'au prochain top d'horloge, après l'insertion de toutes les tâches dans ce *work-pool* et la synchronisation de tous les processus *workers* à la fin de ce premier top d'horloge.

Le second top d'horloge d'une période de cellWORKER

Comme nous l'avant déjà signalé, une fois toutes les tâches du premier *work-pool* traitées, les processus *workers* passent au traitement de celles du second, toujours en parallèle. L'algorithme général est le suivant :

```
DEBUT
  encore_une_tache  <--  VRAI
  TANTQUE (encore_une_tache) FAIRE        // Tant qu'il y a des tâches dans
                                          // le second work-pool.
    *tache  <--  Recuperer_tache_wpl_2()  // On récupère une tâche.
    Resoudre_Conflits(tache)              // Résoudre les conflits entre les
                                          // différentes tâches, et trier cette
                                          // liste de tâches en conséquence.
    TANTQUE (tache <> NIL) FAIRE
      CAS (tache-->type)
         ROBOT :                          // Si c'est un robot,
           robot_react(tache)             // on met en oeuvre son influence.
         MOBILE_OBJ :                     // Si c'est un objet mobile,
           m_obj_react(tache)             // on met en oeuvre son action.
         VIDE :
           encore_une_tache  <--  FAUX
      FCAS
      tache  <--  (tache-->next)
    FTQUE
  FTQUE
FIN
```

Nous détaillons maintenant l'algorithme de récupération d'une tâche du second *work-pool*.

```
FONCTION Recuperer_tache_wpl_2() : wpl_tache
DEBUT
  EntreeMutex(MutexWWC)              // Récupération et décrémentation du
  wwc  <--  wwc - 1                  // nombre de tâches en attente.
  lo_wwc  <--  wwc
  SortieMutex(MutexWWC)
  SI (lo_wwc < 0)                    // S'il n'y a plus de tâches dans ce
  ALORS                              // work-pool,
     tache.type  <--  VIDE           // Créer une tâche vide.
     tache.next  <--  NIL
     SI (lo_wwc = -Nb_Wk)
```

```
     ALORS
        wwc  <--  0                 // Remise à 0 du nombre de tâches
     FSI                            // en attente dans le 2nd work-pool.
  SINON
     EntreeMutex(MutexRIndex)       // Récupération de l'indice de lecture
     lo_ind   <--  r_index          // dans le 2nd work-pool, "r_index",
     r_index  <--  r_index + 1      // partagé par tous les workers
                                    // et protégé par un mécanisme
                                    // d'exclusion mutuelle en conséquence.
     SortieMutex(MutexRIndex)
     Copier_tache(tache, wpl_2[lo_ind]) // Copie de la tâche à partir du
                                    // work-pool.
  FSI
  RETOURNER(tache)
FIN

Lexique :
  wwc : ENTIER                      // Nombre de tâches en attente dans le
                                    // 2nd work-pool.
  Nb_Wk   : ENTIER                  // Le nombre de processus workers (i.e. P)
  wpl_2   : Liste de wpl_tache      // Le second work-pool.
  MutexRIndex : MUTEX               // Un sémaphore d'exclusion mutuelle
                                    // pendant la lecture (2nd work-pool).
  MutexWWC : MUTEX                  // Un sémaphore d'exclusion mutuelle
                                    // pour la protection du compteur wwc.
```

5.4.3 Optimisation de la gestion des work-pools

Après avoir implanté la première version de notre simulateur, nous avons testé une optimisation du mécanisme de gestion des *work-pools*, afin d'améliorer ses performances parallèles. Chaque processus *worker* traitait en priorité les tâches du second *work-pool* qu'il avait déjà traité dans le premier. Ceci avait pour but de réduire les pertes de temps provenant de la migration des données entre les processeurs d'une machine parallèle à mémoire virtuellement partagée, mais physiquement distribuée, et d'une meilleure réutilisation des caches.

Malheureusement les résultats sont restés les mêmes, et nous expliquons ceci par le fait que les processus *workers* accèdent à toutes les tâches déjà présentes dans le second *work-pool*, lors du test de la présence de conflits entre la nouvelle tâche à insérer et celles déjà présentes. Ceci engendre inévitablement une migration fréquente des données entre les différents processeurs.

5.5 L'utilisation de notre simulateur PIOMAS

5.5.1 L'entrée du simulateur

Afin de faciliter l'usage de notre simulateur PIOMAS, nous avons développé une interface graphique en Java, pour la configuration des systèmes à simuler, qui peut s'exécuter sur n'importe quelle machine supportant la machine virtuelle de Java.

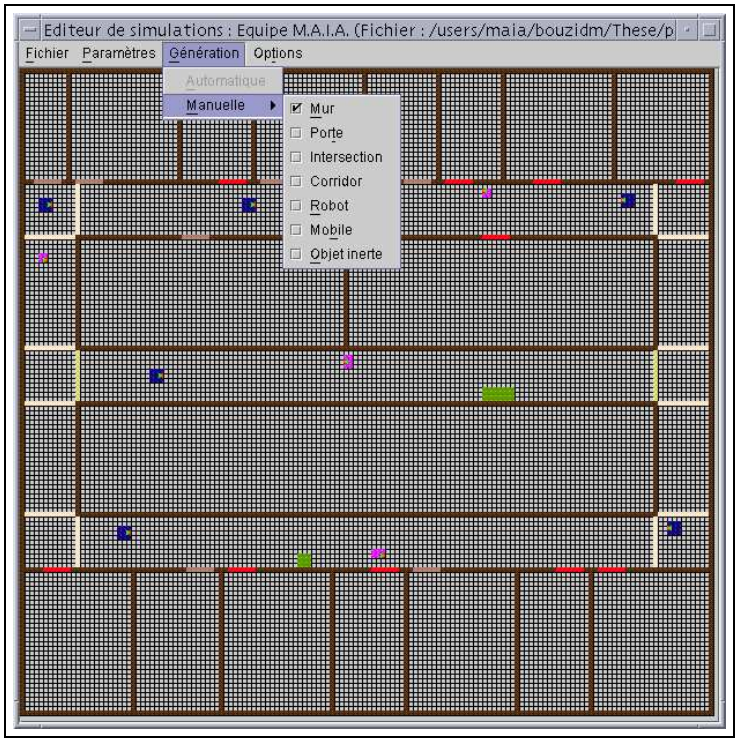

FIG. 5.9 – *L'éditeur graphique de configuration des systèmes à simuler*

Cette interface est un éditeur graphique (figure 5.9), qui permet à l'utilisateur de spécifier et de configurer différents systèmes, avec des environnements de différentes tailles, ainsi que différents nombres d'agents, d'objets inertes et actifs. Les agents et les objets actifs peuvent avoir différentes propriétés (comme la vitesse). Nous signalons que cette interface est dédiée aux applications de simulation de robots mobiles, du type que nous avons indiqué dans le chapitre 4. Néanmoins, elle produit un fichier texte de configuration, qui sera utilisé par le simulateur PIOMAS, et elle peut être étendue pour devenir plus générique et couvrir plus d'applications.

Nous pouvons aussi spécifier d'autres paramètres pour PIOMAS avant de lancer les simulations, comme le nombre de processus *workers* utilisés, le nombre de cycles de simulation, ... Ces paramètres, ainsi que le nom du fichier de configuration généré par l'éditeur graphique, doivent être spécifiés sur la ligne de commande de PIOMAS.

Nous avons prévu d'autres paramètres pour la configuration de notre simulateur PIOMAS, mais qui doivent être modifiés au niveau du code source et qui nécessitent, en conséquence, sa re-compilation. Nous pouvons, par exemple, modifier les matrices de confusion des observations, et le nombre de cases occupés par le robot. Le changement des comportements des robots fait partie de cette catégorie de paramètres, mais nous envisageons son insertion dans la catégorie des paramètres qui peuvent être spécifiés sur la ligne de commande dans le futur proche, afin de faciliter la tâche de l'utilisateur. Néanmoins, dans la version actuelle du simulateur la spécification du comportement de l'agent est relativement simple. En effet, nous avons prévu une fonction, en langage C, à définir par l'utilisateur. Ce dernier dispose des fonctions de récupération des observa-

tions disponibles pour l'agent (des différents capteurs et des différentes faces du robot), et d'un ensemble d'influences pour l'agent qui peuvent être mises en œuvre par le simulateur. L'utilisateur doit générer à partir de ces observations l'influence adéquate, selon le modèle de comportement qu'il adopte pour l'agent.

5.5.2 La sortie du simulateur

Nous avons conçu une seconde interface graphique en Java (figure 5.10)[14] pour la visualisation et le suivi *post-mortem* de la simulation, en vue d'une analyse qualitative des résultats. Cette interface permet également une analyse quantitative des résultats des simulations, en générant quelques statistiques, aussi bien, sur une seule exécution que sur un ensemble d'exécutions. Nous citons comme exemple de statistiques, le nombre moyen de chocs que subit un robot ou le temps moyen pour faire un aller/retour dans un couloir durant une simulation, ainsi que sur une série de simulations. Cette interface va aider l'utilisateur à mettre au point le simulateur et ajuster ainsi les comportements des agents.

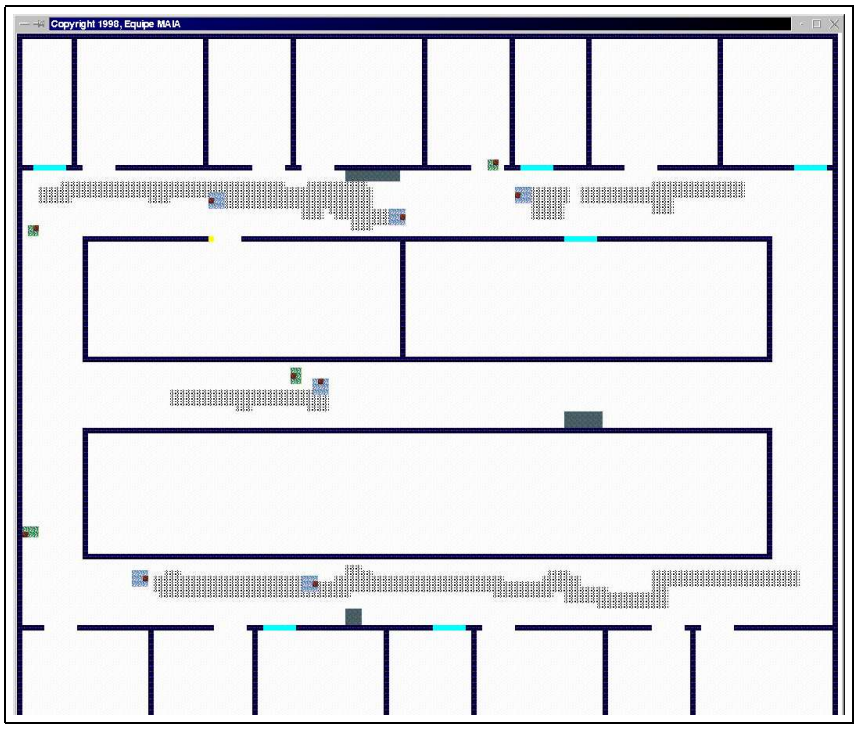

FIG. 5.10 – *L'interface de suivi post-mortem des simulations*

14. qui peut aussi s'exécuter sur n'importe quelle machine supportant la machine virtuelle de Java

5.6 Conclusion

Nous avons présenté dans ce chapitre l'implantation de notre simulateur de robots mobiles PIOMAS, tout en passant par une description des machines parallèles utilisées, ainsi que de la bibliothèque parallèle ParCeL-3 que nous avons utilisée. Nous avons donc présenté l'architecture cellulaire de notre simulateur, au sens de ParCeL-3, ainsi que certains détails d'implantation qui nous ont paru intéressants à décrire.

Nous avons aussi présenté comment utiliser notre simulateur, en passant par une brève description des interfaces graphiques que nous avons développées pour notre simulateur, aussi bien pour la création et la configuration des systèmes à simuler, que pour le suivi *post-mortem* des simulations.

Nous décrivons dans le chapitre suivant les tests que nous avons menés sur PIOMAS, ainsi que les résultats obtenus suite à leurs évaluations.

Chapitre 6

Tests et évaluation de PIOMAS

6.1 Introduction

Nous présentons dans ce chapitre les tests et les expériences que nous avons menés sur notre simulateur PIOMAS, en vue de son évaluation et de sa validation. Nous commençons par détailler les buts de ces expériences, puis nous expliquons leurs principes. Par la suite nous présentons et discutons les résultats obtenus tant du point de vue "simulation" que parallélisme. Enfin, nous discutons théoriquement l'apport de nos modèles stochastique d'interaction agent/environnement et de simulation parallèle, ainsi que celui du simulateur de robots construit sur la base de ces modèles [Bouzid *et al.*, 2001].

6.2 Description des expérimentations

6.2.1 Objectifs

Le simulateur PIOMAS que nous avons développé constitue une première version d'une plate-forme de simulation de robots mobiles. Les buts des expériences que nous avons menées étaient donc de vérifier la facilité d'utilisation de ce simulateur, sa généricité et son extensibilité pour la famille d'applications que nous avons choisie (i.e. celles des robots mobiles, en interaction dans un environnement physique), afin de servir de véritable plate-forme de simulation. Nous avons aussi voulu nous assurer de la validité de notre modèle d'interaction stochastique entre l'agent et l'environnement, et montrer l'importance de la reproduction des incertitudes et des erreurs des capteurs et des effecteurs au niveau d'un simulateur multi-agent. Enfin, nous avons voulu vérifier que notre modèle de simulation parallèle, basé sur les conflits, est à la fois adéquat et efficace pour la simulation d'agents situés.

6.2.2 Configuration des expériences

Notre simulateur possède plusieurs paramètres que nous avons changés, au travers de différentes expériences, afin de vérifier toutes les propriétés citées dans le paragraphe

précédent. Nous classons les paramètres du simulateur en deux classes :

- ceux qui nous permettent de spécifier la configuration initiale du monde à simuler. Ces paramètres sont précisés au niveau de la ligne de commande du simulateur, comme le nombre de processeurs et de processus *workers*, le nombre de cycles de simulations et le nom du fichier de configuration du système à simuler. Ce fichier contient à son tour d'autres paramètres, comme la taille de l'environnement, le nombre, les positions et les périodes d'actions des robots et des objets mobiles, ainsi que les coordonnées des différents objets inertes, etc ... ,
- des paramètres qui dictent le fonctionnement et l'évolution des systèmes multi-agents à simuler, comme le changement du comportement d'un agent, l'insertion de nouveaux types de capteurs, ainsi que de leurs matrices de confusions, le changement du nombre de cases occupées par un robot, etc ... Ces paramètres nécessitent la re-compilation du code source.

6.2.3 Choix des simulations

Nous avons effectués plusieurs tests sur notre simulateur, en faisant varier le nombre de processeurs, de un à seize pour l'évaluation des performances parallèles. Les expériences ont portées sur des environnements de différentes configurations et tailles, variant de 140x150 à 250x350 cases, ayant un nombre variable d'agents et d'objets mobiles. Un de ces environnements correspond au premier étage de notre laboratoire LORIA, les autres sont virtuels.

Nous avons aussi fait varier le nombre de cases occupées par un robot de 3x3 à 5x5, pour tester la généricité de notre simulateur.

Afin de vérifier l'impact de notre modèle d'interaction stochastique sur les comportements des agents, et ainsi sur les résultats de la simulation, nous avons effectués des tests sur un ensemble d'environnements ayant des configurations particulières, comme ceux constitués d'un seul couloir ou de deux couloirs perpendiculaires. Ces environnements ont fait intervenir un nombre variable de robots, ayant des comportements simples mais particuliers, comme la navigation de bout en bout d'un couloir (i.e. des mouvements d'aller/retour), et celui où un ensemble de robots se suivent dans un mouvement d'aller/retour (figures 6.1 et 6.2). Ce dernier test implique une coordination implicite entre les agents via la perception, pour qu'ils réussissent à se suivre.

Pour chaque configuration donnée, nous avons réalisé quatre types de tests :

- le premier suppose que les capteurs et les effecteurs des agents sont totalement fiables,
- le second type de tests portait sur des agents ayant des capteurs fiables, mais des effecteurs non-fiables,
- le troisième est le cas inverse du précédent ; c'est-à-dire que les capteurs sont supposés être non-fiables et les effecteurs sont fiables,
- le quatrième cas est le plus général, faisant intervenir des agents ayant des capteurs et des effecteurs non-fiables.

Ces quatre types de tests ont été réalisés en changeant juste les matrices de confusion des capteurs et les distributions de probabilité sur les transitions des actions produites par les effecteurs.

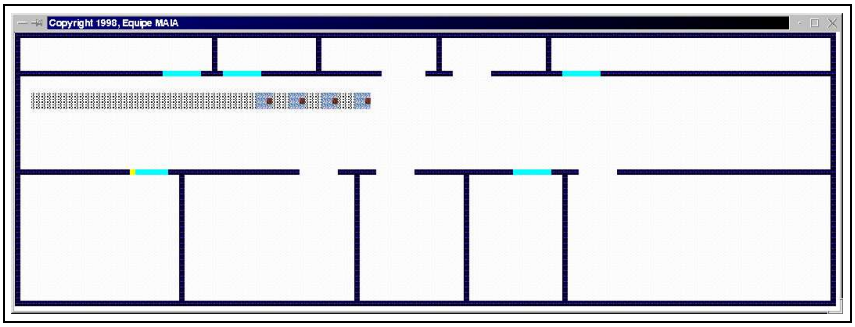

FIG. 6.1 – *Exemple de robots qui se suivent dans un mouvement d'aller/retour dans un couloir*

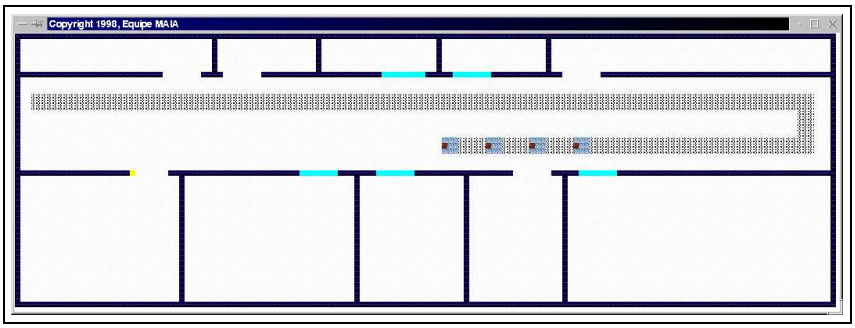

FIG. 6.2 – *Exemple de robots qui se suivent dans un mouvement d'aller/retour dans un couloir, après un demi tour*

6.3 Résultats des expérimentations

6.3.1 Outil de simulation

Nous avons effectués tous les tests mentionnés, en changeant les différents paramètres de configuration indiqués dans le paragraphe 6.2.2. Nous avons vérifié la facilité d'utilisation de notre simulateur, plus particulièrement après le développement de l'éditeur graphique de spécification et de configuration d'environnements.

La généricité du simulateur a été vérifiée suite à la réussite de ses exécutions après la modification du nombre de cases occupées par un robot, que nous avons fait varier de 3x3 jusqu'à 5x5 cases. En effet, dans ces cas toutes les distributions de probabilité sur l'ensemble des transitions des agents et les observations ont été mises à jour correctement en fonction des règles implantées au niveau du simulateur, sans aucune modification du code source (sauf, évidemment, le nombre de cases occupées par un agent).

Nous avons en plus vérifié l'extensibilité de notre simulateur, lorsque nous avons modélisé le nouveau système de perception correspondant à la caméra du robot. Nous n'avons effectué que quelques modifications du code source, écrit dans le langage C, concernant les observations, leurs matrices de confusion et le comportement des agents. Ces modifications du code source étaient faciles à effectuer, et leur réalisation n'a pas demandé beaucoup de temps. Cela confirme en quelque sorte la généricité de notre

simulateur. Des expériences ont été menées sur cette nouvelle version du simulateur et des résultats satisfaisants ont été obtenus.

Notre outil de simulation peut aussi être exécuté sur plusieurs machines, avec différents systèmes d'exploitation et différentes architectures. En effet, nous l'avons exécuté avec succès sous le système *Windows-NT* aussi bien sur un PC mono-processeur que sur une machine quadri-processeur à mémoire partagée. Nous l'avons aussi exécuté avec succès sous le système *Irix*, sur une machine parallèle à mémoire physiquement distribuée, mais virtuellement partagée (DSM).

6.3.2 Résultats d'un point de vue agent

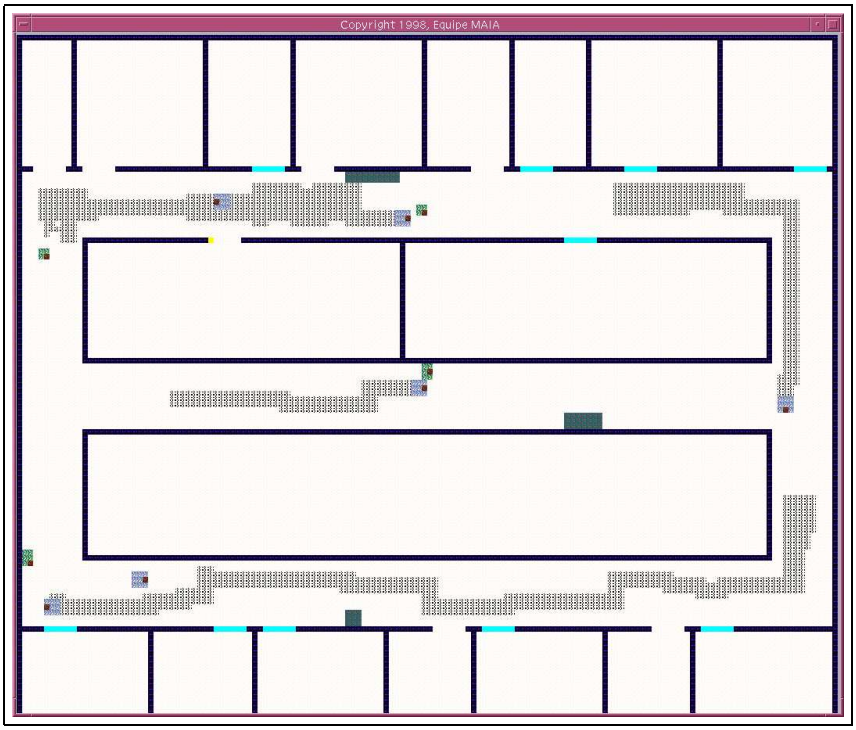

FIG. 6.3 – *Les trajectoires d'un ensemble de robots, visualisées par l'interface de suivi post-mortem des simulations*

Nous rappelons que notre but est de construire une plate-forme de simulation multi-robot, qui reproduit la non-fiabilité des capteurs et des effecteurs des robots. Cette plate-forme doit permettre aux utilisateurs de tester, évaluer et mettre au point différents comportements d'agent dans le futur, avant de passer à leurs tests sur le système réel.

Le but des expériences que nous avons menées sur la première version de notre outil de simulation, est de vérifier l'impact de notre modèle d'interaction stochastique sur les comportements des agents. Cela va nous permettre de confirmer notre idée de départ, concernant l'importance de modéliser la non-fiabilité des capteurs et des effecteurs des agents lors de la conception et le développement d'un simulateur d'agents.

Comparaison à une plate-forme réelle

Nous avons utilisé l'interface graphique que nous avons développée pour le suivi *post-mortem* de la simulation, afin d'évaluer les résultats des simulations effectuées. Nous avons remarqué, dans le cas où les capteurs et les effecteurs des agents ne sont pas fiables, que les trajectoires des robots ne sont pas linéaires (figure 6.3). Ces trajectoires ne ressemblent que légèrement à celles du robot réel, puisque le comportement implanté sur le robot réel est plus complexe (une architecture multi-couche) que celui que nous avons implanté dans le simulateur. Malheureusement, nous n'avons pas eu le temps pour reproduire le comportement réel sur le simulateur, ni la chance pour simplifier le comportement du robot réel, puisqu'il est tombé en panne jusqu'à la fin de cette thèse.

Vérification de l'impact de la non-fiabilité des capteurs et des effecteurs sur les comportements des agents

Nous avons bien vérifié que la non-fiabilité des capteurs et des effecteurs des agents a naturellement un impact négatif sur l'efficacité des comportements des robots au cours de la simulation, mais plus particulièrement celle de ses capteurs, pour le type d'expériences que nous avons réalisées. En effet, nous avons remarqué que les agents peuvent, par exemple, subir plus de chocs lorsque des incertitudes et des erreurs se produisent au niveau de leurs capteurs que lorsqu'elles se produisent au niveau de leurs effecteurs.

Le tableau 6.1 résume le nombre moyen de chocs subits par un ensemble de 4 robots qui se suivent dans un couloir, pour les quatre types de tests cités dans le paragraphe 6.2.3 que nous avons réalisés. Les 4 robots avancent à la même vitesse d'une case/cycle et tous les tests de simulation ont duré 20000 cycles. Nous avons remarqué dans le cas où les capteurs et les effecteurs des agents sont parfaitement fiables, que les robots se suivent dans des mouvements d'aller/retour, tout au long du couloir sans difficulté.

Dans le second type de tests, où uniquement les capteurs des agents sont fiables, nous avons trouvé que les robots arrivent à se suivre au début de la simulation, mais par la suite ils échouent dans cette mission à cause de l'accumulation des glissements, qui déclenchent souvent le comportement d'évitement d'obstacles gênant la réussite de celui de suivi. En effet, les glissements font rapprocher les robots des murs, ce qui provoque le déclenchement du comportement d'évitement d'obstacles, qui est plus prioritaire. Après plusieurs déclenchements, les robots se séparent et ne se perçoivent plus pour pourvoir se suivre. Dans ce test un faible nombre de chocs est apparu à cause des glissements.

Les tests du troisième type que nous avons menés (i.e. ceux où uniquement les effecteurs sont fiables), ont montré que les robots n'arrivent généralement pas à se suivre à cause de la confusion de l'observation identifiant un robot avec d'autres observations du monde. Un nombre de chocs important est apparu, aussi à cause de la mauvaise identification des différents objets du monde.

Les résultats que nous avons obtenus suite aux quatrième type de tests, qui s'approche le plus de la réalité (i.e. celui où les capteurs et effecteurs des agents sont non fiables), sont identiques à ceux du troisième type de tests avec un nombre plus important de chocs subits par les robots à cause des glissements.

Les résultats de ces trois derniers types de tests confirment bien la nécessité de prise en compte de la non-fiabilité des capteurs et des effecteurs des agents, lors de la conception de leurs comportements, pour qu'ils réussissent dans ce genre de missions au niveau du simulateur même.

Nb. robots	Capteurs fiables	Effecteurs fiables	Nb. chocs
4	Oui	Oui	0
4	Oui	Non	3
4	Non	Oui	837
4	Non	Non	1333

TAB. 6.1 – *Le nombre moyen de chocs subits par un ensemble de 4 robots qui se suivent dans un couloir, sur des simulations de 20000 cycles*

Afin de vérifier davantage l'impact négatif des erreurs et incertitudes qui peuvent se produire au niveau des capteurs et effecteurs des agents, nous avons mesuré le temps moyen nécessaire pour un robot, ayant une vitesse de mouvement égale à 1 case/cycle, pour traverser un couloir tout en évitant les obstacles. Les tests ont porté sur des petites simulations de 1000 cycles, ainsi que des grandes de 20000 cycles. Les résultats obtenus sont les mêmes, et le tableau 6.2 résume ceux obtenus pour 1000 cycles. Nous avons trouvé que le temps nécessaire pour traverser le couloir, exprimé en nombre de cycles, est plus important (plus d'une fois et demi) lorsque les capteurs et les effecteurs du robot sont non-fiables, et même quand c'est uniquement les capteurs qui sont non-fiables. En revanche, nous pouvons remarquer que lorsque les effecteurs du robot ne sont pas fiables, le temps nécessaire pour traverser le couloir est un peu plus court que celui où les effecteurs sont parfaitement fiables, en gardant un même état pour les capteurs (soit fiables, soit non-fiables). Cela s'explique par le fait que les glissements du robot, à cause de la non-fiabilité de ses effecteurs, diminuent le temps nécessaire pour la traversée du couloir. Cela est bien évidemment en contre partie d'un manque de précision au niveau des actions des robots et d'un risque d'échec plus important dans leurs missions.

Sur un plan qualitatif, nous avons remarqué dans les deux premiers types de tests (i.e. capteurs fiables), que le robot a réussi dans sa mission de navigation tout au long du couloir, dans un mouvement d'aller/retour. D'ailleurs, le nombre moyen d'aller/retour, qui est égal à 8 et qu'a effectué l'agent pendant une simulation de 1000 cycles, confirme cette constatation. Mais, lorsque les capteurs sont devenus non-fiables, nous avons trouvé que le robot a des difficultés à naviguer correctement dans le couloir à cause des confusions entre les observations. Cela est aussi confirmé par le nombre d'aller/retour qu'il a effectué, qui chute à environ 5.

Vérification de l'impact de la non-fiabilité des capteurs et des effecteurs sur la coordination des agents

Nous avons effectué d'autres expériences, toujours dans le but de vérifier l'impact négatif de la non-fiabilité des capteurs et des effecteurs des agents sur leurs comportements au cours de la simulation. Nous avons placé deux robots dans un couloir, chacun dans une extrémité (figure 6.4), ayant pour mission de naviguer avec la même vitesse

Longueur du couloir (en nombre de cases)	Capteurs fiables	Effecteurs fiables	Temps de parcours (en nombre de cycles)	Nb. d'aller/ retour
80	Oui	Oui	116	8
80	Oui	Non	113	8
80	Non	Oui	206	4.8
80	Non	Non	186	5

TAB. 6.2 – *Le temps moyen nécessaire pour un robot, ayant une vitesse de 1 case/cycle, pour traverser un couloir, sur des simulations de 1000 cycles*

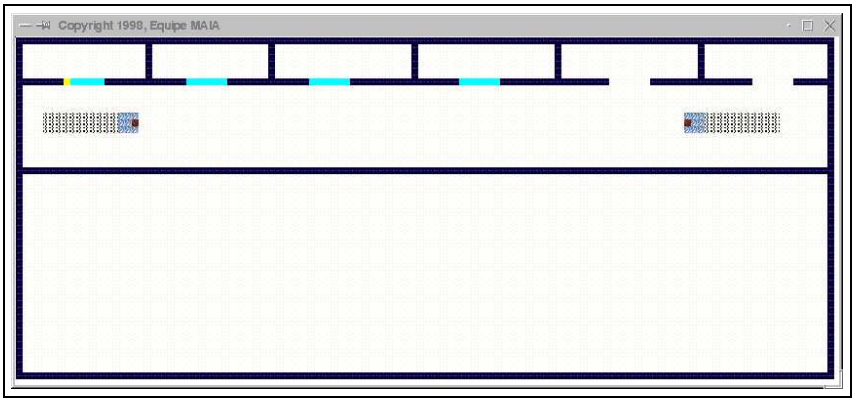

FIG. 6.4 – *La configuration initiale pour la navigation de 2 robots, placé chacun dans une extrémité d'un couloir*

dans des mouvements d'aller/retour, tout en s'évitant lorsqu'ils se retrouvent face-à-face (figure 6.5). Cette mission implique une coordination implicite entre les agents, via la perception, pour être réussie. Cette coordination consiste par exemple à serrer à droite de la part de chaque agent. Nous avons trouvé que dans le cas où les capteurs sont parfaitement fiables, les robots réussissent dans leur mission de navigation sans difficulté. Nous avons noté l'apparition de quelques chocs, lorsque les effecteurs sont non-fiables, toujours à cause des glissements. Mais, dans le cas où les capteurs sont non-fiables, le mécanisme de coordination des robots n'arrive pas à fonctionner correctement, à cause des confusions entre les observations. Cela est d'autant pire lorsque l'on rajoute des incertitudes et des erreurs à la mise en œuvre des actions des effecteurs.

Conclusion

Tous les résultats que nous venons de présenter nous ont parus logiques, et nous ont permis de confirmer nos réflexions théoriques sur la nécessité de modéliser la non-fiabilité des capteurs et effecteurs des agents pendant les simulations multi-agents. Cela nous a donc permis de vérifier que la conception des comportements des agents sur le simulateur n'est pas évidente, à cause des erreurs et incertitudes qui peuvent se produire au niveau de leurs capteurs, ainsi que de leurs effecteurs, et que nous avons intégrées dans les règles d'interaction entre les agents et l'environnement.

Suite à l'implantation de notre modèle d'interaction stochastique, nous avons testé

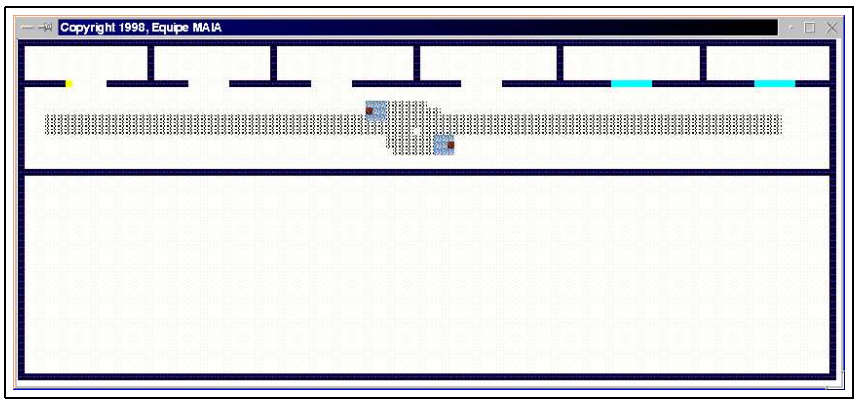

FIG. 6.5 – *Comment les 2 robots doivent s'éviter lorsqu'ils se retrouvent face-à-face; chacun serre à droite et continue sa navigation*

certains comportements de robot simples, comme celui où un ensemble de robots se suivent, ou celui consistant à effectuer des aller/retour de bout en bout d'un couloir. Nous avons bien vérifié que ces comportements s'implantent avec succès sur le simulateur en supposant que les capteurs et les effecteurs des agents sont complètement fiables, mais malheureusement ils ne fonctionnent pas correctement lorsque des erreurs et des incertitudes se produisent au niveau des effecteurs des agents et surtout au niveau de leurs capteurs. Nous devons donc tenir compte de l'occurrence de ces erreurs et incertitudes lors de la modélisation des comportements des agents, dans le cadre de la simulation, afin d'obtenir des comportements plus sophistiqués et robustes aux bruits, à l'inverse des simulateurs multi-agents classiques où les capteurs et effecteurs des agents sont supposés parfaitement fiables et la conception de leurs comportements est ainsi simple. Cela aura comme conséquence que les résultats qui seront obtenus sur le simulateur, seront assez valide lors du passage à l'expérimentation des comportements des agents sur le système réel.

6.3.3 Les performances parallèles

Nous avons effectué plusieurs tests sur des environnements de 200x300 cases, contenant 16 robots, 8 objets mobiles, 19 portes, des murs et des objets inertes. Le comportement des robots consiste à naviguer dans l'environnement, constitué de couloirs et de bureaux, tout en évitant les obstacles. Nous avons utilisé un nombre de processus *workers* égal au nombre de processeurs. Nous avons lancé les simulations sur différents nombres de cycles (1000, 10000, 20000, 50000, etc ...), et nous avons trouvé des résultats presque équivalents. Nous avons choisi de reporter dans ce qui suit les résultats obtenus avec des simulations s'étalant sur 20000 cycles [Vialle *et al.*, 2000], afin d'avoir des temps d'exécution de quelques centaines de secondes et pouvoir négliger les erreurs de mesures.

Le temps d'exécution que nous avons mesuré inclut le temps de calcul de tous les cycles de simulation et le temps nécessaire pour les opérations d'entrée/sortie, comprenant principalement le temps de sauvegarde des résultats de simulation sur le(s)

disque(s) dur(s). Mais ce temps d'exécution n'inclut pas celui qui est relatif à la phase d'initialisation, que nous n'avons pas parallélisée. En effet, il n'est pas évident de trouver une implantation parallèle efficace pour cette phase, qui soit indépendante de la configuration initiale du système à simuler. Cependant, cette phase d'initialisation ne représente que 1% du temps d'exécution global d'une simulation de 20000 cycles. La figure 6.6 résume les résultats obtenus sur les deux machines parallèles utilisées, qui sont la SGI-Origin 2000 et la SGI-VWS 540.

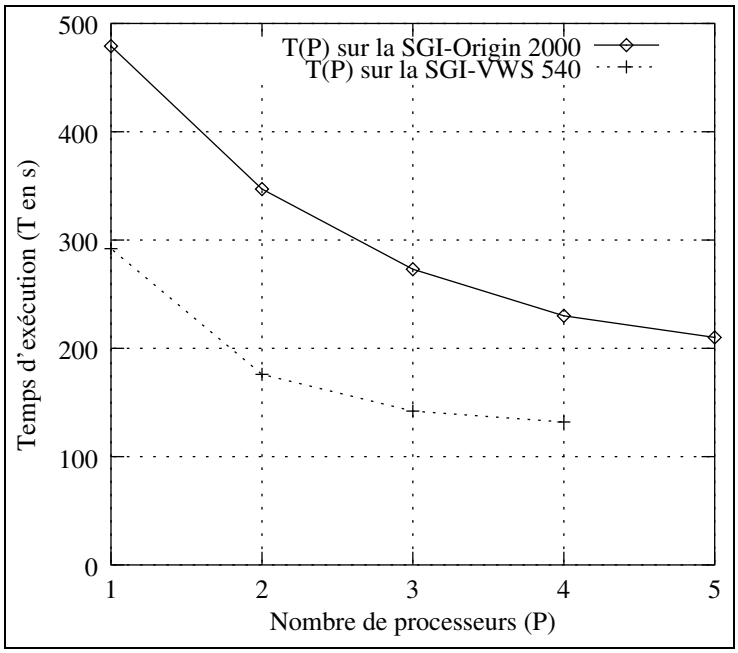

FIG. 6.6 – *Le temps d'exécution diminue en fonction du nombre de processeurs*

Les tableaux 6.3 et 6.4 représentent les accélérations (*speed-up*) et les efficacités de la parallélisation, obtenues sur les deux machines parallèles indiquées précédemment. Le *speed-up* est [Almasi et Gottlieb, 1989] :

$$S(P) = T(1)/T(P),$$

qui correspond au rapport du temps d'exécution sur 1 processeur, par celui sur P processeurs. L'efficacité est [Almasi et Gottlieb, 1989] :

$$e(P) = S(P)/S_{idéal}(P) = S(P)/P,$$

puisque le *speed-up* idéal que nous pouvons obtenir avec P processeurs est égal à P. L'efficacité est donc la fraction du *speed-up* idéal que nous pouvons obtenir avec une certaine parallélisation.

Nous remarquons d'après les résultats que nous avons obtenus, que si le temps d'exécution diminue, le *speed-up* augmente lentement et l'efficacité chute rapidement. Cependant, nous avons obtenu avec cette première implantation parallèle des *speed-up* supérieurs à 2 en utilisant 4 processeurs, sur les deux machines parallèles que nous avons utilisées.

Nb. de processeurs P	Temps d'exécution $T(P)$ (s)	*Speed-up* $S(P)$	Efficacité $e(P)$ (%)
1	479	1	100
2	347	1.38	69
3	273	1.76	59
4	230	2.09	52
5	210	2.28	46

TAB. 6.3 – *Les performances parallèles sur la machine SGI-Origin 2000*

Nous rappelons que ces deux machines sont à mémoire partagée, mais qu'elles ont des architectures physiquement différentes : la SGI-VWS 540 est à mémoire physiquement partagée, et la SGI-Origin 2000 est à mémoire physiquement distribuée, mais virtuellement partagée (DSM).

Nb. de processeurs P	Temps d'exécution $T(P)$ (s)	*Speed-up* $S(P)$	Efficacité $e(P)$ (%)
1	292	1	100
2	176	1.70	83
3	142	2.06	69
4	132	2.2	55

TAB. 6.4 – *Les performances parallèles sur la machine SGI-VWS 540*

Nous avons noté que le temps d'exécution sur la machine SGI-VWS 540 est plus court que celui sur la SGI-Origin 2000 (figure 6.6). Nous expliquons cela par le fait que la première machine est plus récente que la seconde, par conséquent elle possède des processeurs plus puissants, puisque ce résultat est vérifié même sur des exécutions avec un seul processeur. La machine SGI-VWS 540 est constituée de 4 processeurs de type PIII-Xeon à 450 MHz, et la SGI-Origin 2000 possèdent 64 processeurs de type R10000 à 195 MHz.

Nous avons aussi remarqué que les accélérations et les efficacités sont légèrement meilleures sur la SGI-VWS 540 que sur la SGI-Origin 2000, jusqu'à 4 processeurs. Mais, nous avons constaté que ces mesures de performances chutent plus rapidement sur la première machine que sur la seconde (figures 6.7 et 6.8). En effet, ces deux machines ont des mémoires partagées différentes, et nous pensons que le matériel utilisé pour le partage de la mémoire sur la SGI-Origin 2000 est plus efficace que celui utilisé sur la SGI-VWS 540. La mauvaise gestion du partage de la mémoire a pour effet de faire chuter les performances du programme parallèle lorsque plusieurs processeurs accèdent concurremment à la mémoire partagée.

Conclusion

Ces résultats nous permettent d'être optimiste pour les futures versions, puisque beaucoup d'optimisations restent possibles, et nous pensons que nous obtiendrons de meilleures performances lorsque nous implanterons des comportements plus complexes

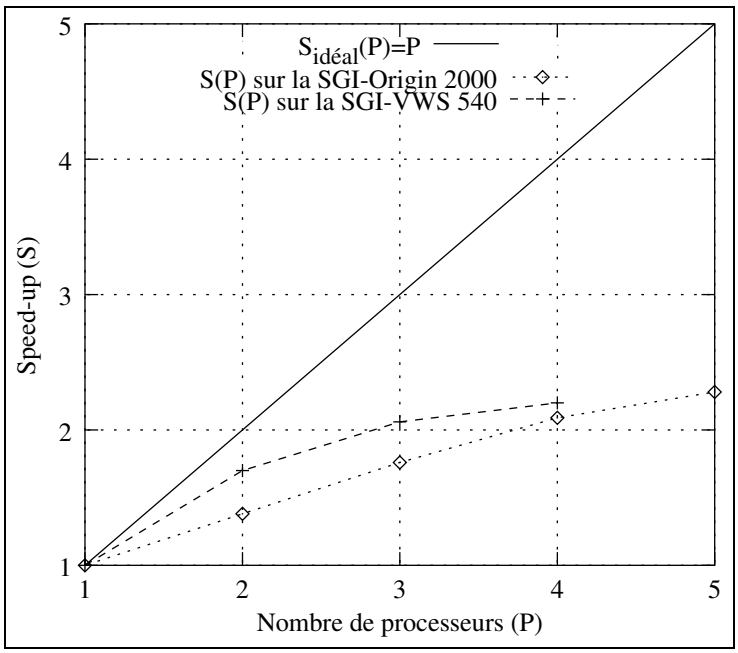

FIG. 6.7 – *Comparaison des courbes d'accélération (speed-up), en fonction du nombre de processeurs, sur les 2 machines ; la SGI-VWS 540 et la SGI-Origin 2000*

au niveau des agents, qui s'exécutent en parallèle. En effet, cela demandera plus de temps de calcul et nous obtiendrons de meilleures accélérations si les conflits ne font pas toujours intervenir tous les agents aux mêmes instants de simulation.

6.4 L'apport du simulateur

6.4.1 Apport général

Le simulateur que nous proposons est basé sur une modélisation stochastique et locale de l'interaction entre l'agent et l'environnement. En effet, pour construire le simulateur, nous devons modéliser la non-fiabilité de chaque capteur et effecteur de chaque type d'agent. Cela passe par la détermination de l'ensemble des observations qui peuvent être perçues au moyen de chaque type de capteur, ainsi que les classes de distance et leurs matrices de confusions respectives. Concernant les effecteurs, nous devons déterminer pour chaque action que l'agent peut entreprendre, l'ensemble des transitions qui peuvent en résulter, ainsi que la distribution de probabilité qui le régit.

Toutes les distributions de probabilité peuvent être apprises à partir du système réel, en entraînant les différents types d'agents dans des endroits génériques de l'environnement. Nous pouvons par la suite construire l'environnement global, et déduire toutes les distributions de probabilité de l'interaction entre chaque type d'agent et n'importe quel région de l'environnement, en appliquant les lois du monde spécifiques aux probabilités apprises (comme nous l'avons appliqué dans la section 4.3).

Nous pouvons alors construire n'importe quel environnement réel ou virtuel, afin de

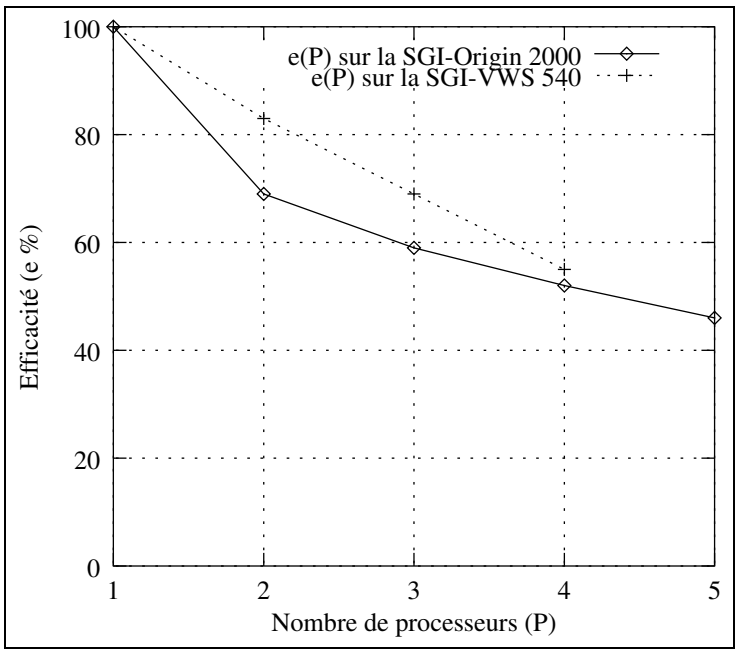

FIG. 6.8 – *Comparaison des courbes d'efficacité, en fonction du nombre de processeurs, sur les 2 machines ; la SGI-VWS 540 et la SGI-Origin 2000*

mener différents types d'expériences, qui peuvent être dangereuses, coûteuses, ou qui demandent beaucoup de temps de calcul, etc ... Ces expériences peuvent concerner différentes applications d'agents ou de multi-agents, notamment la planification et l'apprentissage.

Les expériences peuvent être dangereuses à cause, par exemple, des risques présents dans le monde réel ou d'erreurs logicielles qui peuvent endommager le système réel. Elles peuvent être coûteuses, si le système est constitué ou fait intervenir des matériaux chers. Enfin, les expériences sur le système réel peuvent demander beaucoup de temps et de contrôle de la part de l'utilisateur, sans que cela ait d'intérêt pour les résultats : dans ce cas, le simulateur peut libérer l'utilisateur de la tâche de contrôle et peut même dans certains cas mener les expériences plus rapidement que sur le système réel.

Le but des expériences sera donc d'évaluer, d'ajuster et de valider les comportements des agents, qui peuvent être des comportements de coordination, de coopération, etc ... , comme suit :

- nous pouvons tester différents comportements pour les agents, jusqu'à trouver ceux qui fonctionnent correctement, même dans la pire configuration de l'environnement, afin de maximiser les chances de l'adéquation des comportements aux agents du système réel,
- étant donné un environnement dont la configuration est bien définie, nous pouvons chercher les conditions et les paramètres que les agents doivent remplir pour que leurs comportements aboutissent à la réussite,
- cas inverse ; étant donné des agents ayant des caractéristiques et des paramètres bien connus, nous pouvons déterminer les conditions et les contraintes auxquelles doit ré-

pondre un nouvel environnement que nous voulons construire, afin d'aboutir au succès,
- nous pouvons créer aléatoirement des environnements, ou introduire des composantes aléatoires au niveau du simulateur et étudier leurs effets sur les résultats des comportements des agents. Ceci va nous permettre d'évaluer les réactions des agents face à des imprévus,
- un autre type d'évaluation de comportements d'agent, peut être envisagé. Il concerne la tolérance des agents aux pannes, malgré l'imprécision et les erreurs de leurs capteurs et effecteurs. Nous pouvons mettre en œuvre des scénarios permettant de faire tomber en panne certains capteurs et/ou effecteurs d'un ou de plusieurs agents (en changeant les distributions de probabilité correspondantes) et analyser les résultats, afin d'affiner les comportements des agents et de les rendre plus tolérants aux pannes,
- un autre type d'expériences est envisageable, mais un peu plus difficilement. Nous pouvons supposer avoir un environnement et des agents avec des comportements bien définis, et chercher les bonnes distributions de probabilité pour les capteurs et les effecteurs des agents, pour qu'ils fournissent les bons résultats (ajustement des distributions de probabilité initialement apprises). Par la suite, nous pouvons soit améliorer la précision des capteurs et des effecteurs des agents, soit chercher les bons capteurs et effecteurs qui existent sur le marché, afin de respecter les distributions de probabilité trouvées. L'ajustement de la précision des capteurs et effecteurs peut être effectué au niveau matériel et/ou logiciel, après l'étude des disparités entre les différentes distributions de probabilité (apprises et calculées). Un nouvel apprentissage de l'interaction entre l'agent et l'environnement sur le système réel est nécessaire, afin de valider le respect des distributions de probabilité trouvées analytiquement, après avoir apporté les modifications nécessaires.

Nous pensons que notre simulateur, construit sur la base de notre modèle stochastique d'interaction entre l'agent et l'environnement, peut avoir d'autres apports pour la planification et l'apprentissage agent et multi-agent. Nous discutons brièvement ces apports dans ce qui suit.

6.4.2 Apport du simulateur pour la planification

Notre simulateur est bien adapté pour le test et la mise au point de plans générés pour un agent ou pour un ensemble d'agents (i.e. dans le cadre de la planification multi-agent, incluant la coordination et/ou la coopération entre les agents). Cette validation de plans est primordiale avant le passage à leurs expérimentations sur le système réel, pour les raisons citées précédemment (danger, coût, etc ...).

Les plans générés pour un agent ou un ensemble d'agents doivent bien sûr tenir compte de la non-fiabilité des fonctions sensori-motrices des agents. Cela peut être établi en appliquant des algorithmes de planification de type stochastique, comme les MDP et les POMDP (voir section 1.4) que nous développons dans notre équipe. Nous pouvons aussi utiliser d'autres types d'algorithmes non-stochastiques de planification agent ou multi-agent, comme le paradigme proposé par Alami et al. pour la planification multi-robot [Alami *et al.*, 1998]. Ce paradigme est basé sur la fusion incrémentale de plans distribués, pour la coopération multi-robot. Il permet de faire face aux échecs des actions

des plans des robots, en replanifiant leurs actions, au niveau local de chaque robot en premier lieu, et à un niveau centralisé en cas d'échec. Néanmoins, il faut apporter quelques modifications à notre simulateur, pour pouvoir valider ce genre d'algorithmes, puisqu'il faut centraliser la planification en cas d'échec des plans locaux des agents.

6.4.3 Apport du simulateur pour l'apprentissage

Dans le cas de l'apprentissage, le simulateur va d'abord jouer le rôle de l'environnement qui réagit aux actions des agents et qui représente le support des observations de ces derniers. En plus de ce dernier rôle, le simulateur peut jouer le rôle de l'instituteur, qui va récompenser ou punir les comportements des agents, et nous pouvons ainsi effectuer de l'apprentissage par renforcement. Les récompenses et les punitions peuvent être facilement intégrées au niveau de l'environnement (de la même façon que les transitions), pour être communiquées aux agents en fonction de leurs actions. Nous pensons que les plans appris seront plus valides, que ceux qui peuvent être appris sur un simulateur classique (i.e. qui ne modélise pas la non-fiabilité des fonctions sensori-motrices des agents), lors du passage à leurs tests sur le système réel.

Un autre avantage de notre simulateur est de pouvoir changer facilement la configuration de l'environnement, et d'apprendre les nouveaux comportements et plans des agents qui y sont adéquats, sans avoir à réapprendre le modèle de l'interaction entre l'agent et l'environnement. En effet, notre modèle est basé sur une modélisation stochastique locale de l'interaction entre l'agent et l'environnement, qui reste toujours valide et qui sert de "brique générique" pour la construction du système global, en appliquant les lois du monde.

L'apprentissage sera donc plus rapide et plus facile à effectuer sur le simulateur que sur le système réel. Néanmoins, quelques expérimentations initiales sur le système réel, pour l'apprentissage du modèle d'interaction, restent toujours nécessaires.

Comme pour la planification, notre simulateur aura en plus pour rôle de valider et d'ajuster les comportements appris avant de passer à leurs expérimentations sur le système réel.

6.5 Conclusion

Nous avons présenté dans ce chapitre les expériences que nous avons menées sur notre simulateur PIOMAS, afin de valider les modèles d'interaction stochastique et de simulation parallèle que nous avons proposés. Le simulateur que nous avons construit constitue une première version d'une plate-forme de simulation pour le test et la validation de comportements de robots mobiles.

Nous avons décrit et discuté les résultats que nous avons obtenus à partir des expériences menées sur notre simulateur, aussi bien sur le plan agent que parallélisme. Tout cela est dans le but de confirmer l'intérêt de notre modèle stochastique d'interaction pour la simulation multi-agent, et de montrer l'efficacité de notre modèle de simulation parallèle à équilibrage dynamique de charge.

Enfin, nous avons discuté l'apport de notre modèle stochastique pour la simulation, ainsi que les retombées positives auxquelles nous nous attendons.

Conclusion et perspectives

1 Conclusion

1.1 Modèles proposés

Dans cette thèse, nous avons proposé un modèle formel de SMA pour la simulation d'agents situés, comprenant un modèle d'environnement, un modèle d'agent, et principalement un modèle stochastique d'interaction agent/environnement. Ce modèle stochastique reproduit les incertitudes et les erreurs des capteurs et des effecteurs des agents.

Nous rappelons que les incertitudes et les erreurs, ou encore la non-fiabilité, des capteurs et des effecteurs comprend leur imprécision, leur non-résistance aux bruits et perturbation de l'environnement, et les pannes qui peuvent éventuellement se produire à leur niveau. Concernant les capteurs, l'incertitude inclut aussi l'incomplétude des informations récupérées, qui ne permet pas aux agents de distinguer parfaitement les différents objets du système.

Notre modèle d'interaction s'inspire des modèles de décision de Markov partiellement observables (POMDP). Concernant la phase de perception de l'agent, nous avons associé à chaque objet présent dans le système multi-agent une observation symbolique permettant à l'agent de l'identifier, en fonction des capteurs utilisés. Deux objets différents ou plus peuvent être désignés par la même observation, afin d'exprimer les incertitudes de ces capteurs. Une matrice de confusion entre les observations est déterminée pour indiquer les erreurs qui peuvent se produire à leur niveau. De la même façon, nous définissons un ensemble de classes de distances permettant d'approximer la distance réelle qui sépare l'observation de l'agent (expression des incertitudes des capteurs), ainsi que la matrice de confusion entre ces classes de distance (expression des erreurs des capteurs).

En ce qui concerne la phase d'action de l'agent, nous déterminons pour chaque type d'agent et chaque action qu'il peut entreprendre l'ensemble des transitions qui peuvent résulter de l'influence générée. Cet ensemble tient compte des erreurs et incertitudes qui peuvent se produire au moment de la mise en œuvre de l'influence de l'agent. Nous définissons aussi une distribution de probabilité sur cet ensemble exprimant la fréquence d'occurrence de chaque transition.

Toutes les matrices de confusion, ainsi que les distributions de probabilité sur les ensembles des transitions des différentes influences des agents, peuvent être déterminées par apprentissage sur le système réel. Le but de notre modèle d'interaction stochastique est de pouvoir construire des simulateurs multi-agents dont les résultats des expériences

seront proches de ceux que nous pouvons obtenir sur le système réel.

Nous avons aussi proposé un modèle de simulation parallèle, pour des agents situés, basé sur l'identification et la répartition des conflits entre les agents, et sur un équilibrage dynamique de charge entre les processeurs. Notre modèle de simulation utilise un mécanisme original de double *work-pools* en cascade, gérés par un ensemble de processus *workers*. Ces processus commencent par traiter les tâches du premier *work-pool*, et une fois terminées, ils se synchronisent et passent au traitement des tâches du second *work-pool*. Les processus *workers* sont donc capables de traiter n'importe quelle tâche dans les *work-pools*, et permettent ainsi d'obtenir un équilibrage dynamique de charge entre les processeurs.

1.2 Simulateur développé

Nous avons utilisé nos deux modèles d'interaction et de simulation parallèle pour construire un simulateur de robots mobiles, en vue de les valider. Nous avons justifié ce choix de type d'application, et nous avons précisé quelques choix et détails de conception, pour la réalisation du simulateur final PIOMAS.

Nous avons expliqué les différents choix techniques que nous avons effectués pour l'implantation parallèle du simulateur. Nous avons implanté PIOMAS sur des machines parallèles à mémoire partagée (physiquement ou virtuellement), sans exclure son exécution sur des PC mono-processeur. Nous avons utilisé une bibliothèque de programmation parallèle ParCeL-3, développée à SUPELEC et basée sur le *multithreading*. Nous avons présenté quelques algorithmes, que nous avons jugés intéressants, concernant la gestion des deux *work-pools* implantés.

1.3 Evaluation

Comme résultats, nous avons reporté que le simulateur est générique, pour le type d'applications choisi, extensible, facile d'utilisation, et qu'il peut s'exécuter sur plusieurs architectures et plusieurs systèmes d'exploitation.

Nous avons aussi vérifié l'impact négatif de l'introduction d'erreurs et d'incertitudes au niveau de l'interaction entre l'agent et l'environnement, sur l'efficacité des comportements des agents. Nous avons noté qu'il faut bien tenir compte de la non-fiabilité des capteurs et des effecteurs des agents lors de la conception de leurs comportements, au niveau du simulateur même, afin de développer des comportements plus sophistiqués.

Sur le plan parallélisme, nous avons prouvé l'efficacité de notre modèle de simulation parallèle pour les agents situés. En effet, nous avons obtenu des performances parallèles satisfaisantes, en termes d'accélération (i.e. *speed-up*), pour un problème irrégulier de simulation, sur deux machines parallèles à architectures différentes. La première est une machine du CCH qui comprend 64 processeurs[15], qui ont une mémoire physiquement distribuée, mais virtuellement partagée (DSM). La seconde machine est un PC quadri-processeur de SUPELEC, avec cette fois-ci une mémoire physiquement partagée.

15. Nous n'avons utilisé que 5 processeurs

Nous avons discuté l'apport de nos modèles, et plus particulièrement celui de notre simulateur pour le domaine multi-agent de façon générale. Nous avons mentionné que notre simulateur est bien adapté pour l'implantation et le test d'algorithmes de planification multi-agent, stochastiques ou non, tenant compte de la non-fiabilité des capteurs et des effecteurs des agents. Nous avons signalé que notre simulateur est aussi bien adapté pour l'apprentissage et la validation de divers comportements pour les agents (coopération, coordination, etc ...). Les algorithmes de planification et d'apprentissage multi-agents font partie des activités de recherche de MAIA, et notre simulateur peut constituer une bonne plate-forme pour leurs implantations, tests (résistance aux erreurs et incertitudes) et validations.

2 Perspectives

2.1 Confrontation de la simulation à l'expérimentation

Il serait intéressant de confronter les résultats obtenus avec PIOMAS, ou ceux qui peuvent être obtenus avec des simulateurs développés sur la base de notre modèle d'interaction, avec les résultats des expérimentations du système réel, afin de valider davantage nos idées. Concernant PIOMAS, nous pouvons chercher une plate-forme de robots de type NOMAD 200 (celui ayant servi de modèle à PIOMAS) pour effectuer les différents tests. Nous pouvons aussi déterminer les matrices de confusion et les ensembles des transitions des influences des robots de type Koala[16], afin de modifier PIOMAS et de pouvoir ainsi utiliser la plate-forme[17] de SUPELEC constituée de deux robots de type Koala. Nous pourrons étudier les disparités entre les différents résultats (i.e. ceux de simulation et ceux du système réel), afin d'apporter éventuellement les changements adéquats au niveau de notre modèle d'interaction, et de réduire ces disparités.

2.2 Applications multi-agents

Nous pouvons développer différentes applications de robotique mobile, où les robots auraient différentes tâches complémentaires ou antagonistes à réaliser. Les comportements des agents doivent tenir compte des erreurs et des incertitudes qui peuvent se produire au niveau de leurs capteurs et de leurs effecteurs. Cela va nous aider à étudier l'impact de notre modèle stochastique sur les mécanismes de coordination et de coopération entre les agents situés.

L'apport de nos modèles et de notre simulateur, décrit dans la section 6.4, présente une partie des perspectives de nos travaux, que nous précisons dans ce qui suit. Nous comptons donc étudier et mettre en œuvre des algorithmes d'apprentissage multi-agents, comme une adaptation de ceux que nous développons dans MAIA [Dutech *et al.*, 2001], qui nous semble être une extension très intéressante à court et moyen terme de nos travaux.

16. fabriqué par la société K-Team
17. récemment construite

Comme nous l'avons déjà signalé, notre simulateur est bien adapté pour valider des algorithmes de planification multi-agents, prenant en compte la non-fiabilité des capteurs. A court terme, nous pouvons adapter et mettre en œuvre certains algorithmes développés dans notre équipe [Chadès *et al.*, 2001] ou déjà existants dans la littérature, et étudier et développer de nouveaux algorithmes à moyen terme, pour mieux profiter de l'aspect stochastique de notre simulateur.

Concernant l'application de notre modèle stochastique d'interaction dans des domaines autres que la robotique mobile, notre équipe est engagée dans des projets dans le domaine médical, comme la surveillance des personnes âgées à domicile ou la réalisation d'un robot d'anesthésie. Dans ce genre d'application, le passage par un simulateur qui reproduit fidèlement la réalité, nous paraît primordial. Le simulateur doit donc reproduire les erreurs et les incertitudes qui peuvent se produire au niveau des capteurs et des effecteurs du système réel, afin de tester la robustesse des applications développées avant de les utiliser sur des patients réels. Notre modèle d'interaction est donc parfaitement adapté pour développer ce genre de simulateur, et nous pouvons encore imaginer différents scénarios entraînant des dysfonctionnements au niveau des capteurs et des effecteurs (en changeant facilement les distributions de probabilité), afin de contrôler le comportement de l'application développée dans de telles situations.

2.3 Amélioration de l'outil de simulation

D'un point de vue outil de simulation, nous comptons faciliter davantage la tâche de l'utilisateur, en lui donnant la possibilité de changer ou d'intégrer de nouveaux capteurs et effecteurs pour l'agent, y compris les matrices de confusion et les distributions de probabilité, sans recompiler le code source. Nous pouvons étendre l'éditeur graphique pour la saisie des nouveaux capteurs et effecteurs, et la génération d'un fichier texte qui sera spécifié comme argument de PIOMAS. Ce dernier analysera ce fichier avant de créer le système correspondant. Nous pouvons également envisager de modifier notre simulateur, pour permettre à l'utilisateur de spécifier et de changer le comportement des agents d'une manière simple, sans avoir à recompiler le code source, en implantant un interpréteur de commandes. Mais cela risque de faire chuter les performances du simulateur du point de vue temps d'exécution.

2.4 Enrichissement du modèle d'interaction

Nous comptons étudier l'amélioration de notre modèle d'interaction, par l'intégration des communications entre agents et la modélisation de l'interaction agent/agent, afin de fournir un modèle complet pour la simulation multi-agent. L'intégration des communications entre les agents, sera facile à réaliser au niveau de notre simulateur, en exploitant les mécanismes de communication disponibles au niveau de ParCeL-3 (décrits dans la section 5.3.1).

Notre modèle d'interaction agent/environnement peut aussi être enrichi, en ajoutant des traces à déposer par les agents dans l'environnement (une sorte de gradient), qui pourront faciliter la coordination et/ou la coopération entre les agents dans certaines applications.

2.5 Optimisation des performances parallèles

D'un point de vue parallélisme, nous pouvons continuer l'étude du problème de l'implantation parallèle de la combinaison des influences des agents, qui présente un goulot d'étranglement séquentiel pour notre simulateur. Cette étude peut porter à la fois sur la partie théorique de notre modèle de simulation parallèle (section 3.3.3), et sur le côté technique de l'implantation, comme celui que nous avons discuté dans la section 5.4.2. Cela nous permettrait d'améliorer les performances parallèles de notre simulateur.

Le temps de calcul de cette phase de combinaison des influences des agents, peut être aussi réduit en introduisant la modélisation de l'interaction agent/agent au niveau de notre modèle (cf. paragraphe précédent). En effet, dans le cas où nous avons une connaissance à priori sur les agents qui seront impliqués dans une même interaction, nous pouvons les fusionner directement en une seule tâche, sans passer par la phase séquentielle de parcours et de test de présence de conflit entre la tâche nouvellement insérée dans le second *work-pool* et celles qui y sont déjà présentes.

Une autre amélioration de l'implantation de notre simulateur PIOMAS est envisageable. Nous pouvons chercher un algorithme parallèle pour la phase de création et d'initialisation du système multi-agent à simuler. Cette phase n'est pas vraiment pénalisante pour les performances parallèles actuelles de notre simulateur, mais elle peut le devenir lorsque les performances du reste du code seront améliorées.

Bibliographie

[Agha *et al.*, 1993] G. Agha, P. Wegner et A. Yonezawa. *Research Directions in Concurrent Object Oriented Programming*. MIT Press, 1993.

[Agha, 1986] G. Agha. *ACTORS, A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.

[Alami *et al.*, 1998] Rachid Alami, Francois Felix Ingrand et S. Qutub. A Scheme for Coordinating Multi-robot Planning Activities and Plans Execution. Dans *European Conference on Artificial Intelligence*, pages 617–621, 1998.

[Almasi et Gottlieb, 1989] G. S. Almasi et A. Gottlieb. *Highly parallel computing*. The Benjamin/Cummings publishing company, Inc, 1989.

[Anderson *et al.*, 1995] T. E. Anderson, D. E. Culler et D. A. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, Février 1995.

[Awerbuch et Shiloach, 1987] B. Awerbuch et Y. Shiloach. New Connectivity and MSF Algorithms for Shuffle-Exchange Network and PRAM. *IEEE Transactions on Computers*, 36:1258–1263, 1987.

[Aycard, 1998] Olivier Aycard. Contribution à la Navigation d'un Robot Mobile. Thèse de Doctorat de l'Université Henri Poincaré - Nancy 1, CRIN/CNRS & INRIA-Lorraine, 1998.

[Balch *et al.*, 1994] T. Balch, H. Forbes et K. Schwan. Dynamic scheduling for mobile robots. Dans *Proceedings of the 6th EuroMicro Workshop*, Juin 1994.

[Banks et Carson, 1984] Jerry Banks et John S. Carson. *Discrete-Event System Simulation*. Prentice Hall, 1984.

[Bates, 1994] Joseph Bates. The Role of Emotion in Believable Agents. *Communications of the ACM*, 37(7):122–125, Juillet 1994.

[Bellman, 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N.J., 1957.

[Blelloch *et al.*, 1994] Guy E. Blelloch, Jonathan C. Hardwick, Jay Sipelstein, Marco Zagha et Siddhartha Chatterjee. Implementation of a Portable Nested Data-Parallel Language. *Journal of Parallel and Distributed Computing*, 21(1):4–14, Avril 1994.

[Booch, 1994] G. Booch. *Analyse et conception orientées objets (2ème édition)*. Addison Wesley, 1994.

[Bourjot *et al.*, 1999] C. Bourjot, V. Chevrier, A. Bernard et B. Krafft. Coordination par le biais de l'environnement : une approche biologique. Dans *Ingénierie des SMA. JFIADSMA'99 : Actes des 7èmes Journées Francophones d'Intelligence Artificielle et Systèmes Multi-Agents*, pages 237–250, Saint-Gilles, Ile de la Réunion, 1999. Hermes.

[Bousquet *et al.*, 1998] F. Bousquet, I. Bakam, H. Proton et C. Le Page. Cormas: Common-Pool Resources and Multi-Agents Systems. Dans *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-98, Volume II*, volume 1416 de *LNAI*, pages 826–837, Castellón, Spain, Juin 1998. Springer.

[Boutilier, 1996] Craig Boutilier. Planning, Learning and Coordination in Multiagent Decision Processes. Dans *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'96)*. Morgan Kaufmann Publishers, 1996.

[Bouton, 1998] Lionel Bouton. ParMASS : un simulateur de systèmes multi-agents parallèle. Mémoire de DEA, Université Henri-Poincaré Nancy 1 et Supélec, 1998.

[Bouzid *et al.*, 1999] Makram Bouzid, Vincent Chevrier, Stéphane Vialle et François Charpillet. Un environnement de simulation orienté Agents : Apport des modèles stochastiques et du parallélisme. Dans *Ingénierie des SMA. JFIADSMA'99 : Actes des 7èmes Journées Francophones d'Intelligence Artificielle et Systèmes Multi-Agents*, pages 329–330, Saint-Gilles, Ile de la Réunion, 1999. Hermes.

[Bouzid *et al.*, 2000a] Makram Bouzid, Vincent Chevrier, François Charpillet et Stéphane Vialle. Parallélisation d'un simulateur d'agents situés. Dans *Journées Scientifiques Centre Charles Hermite*, 2000.

[Bouzid *et al.*, 2000b] Makram Bouzid, Vincent Chevrier, Stéphane Vialle et François Charpillet. A Stochastic Model of Interaction for Situated Agents and its Parallel Implementation. Dans *Proceedings of the ACIDCA'2000: IEEE International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications*, Monastir, Tunisie, 2000.

[Bouzid *et al.*, 2001] Makram Bouzid, Vincent Chevrier, Stéphane Vialle et François Charpillet. Parallel Simulation of a Stochastic Agent/Environment Interaction Model. *ICAE: Integrated Computer-Aided Engineering*, 8(3):189–203, 2001.

[Bratman *et al.*, 1988] Michael E. Bratman, David Israel et Martha Pollack. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, 4:349–355, 1988.

[Bratman, 1987] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, Massachusetts, U.S.A., 1987.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, Avril 1986.

[Brooks, 1991a] Rodney A. Brooks. Intelligence Without Reason. Dans *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991. Morgan Kaufman.

[Brooks, 1991b] Rodney A. Brooks. Intelligence Without Representation. *Artificial Intelligence, January 1991(1-3)*, 47:139–159, 1991.

[Bryson *et al.*, 2000] J. Bryson, W. Lowe et L. A. Stein. Hypothesis testing for complex agents. Dans *Workshop on Performance Metrics for Intelligent Systems*, NIST, 2000.

[Burmeister *et al.*, 1997] B. Burmeister, A. Haddadi et G. Matylis. Applications of multi-agent systems in traffic and transportation. *IEE Transactions on Software Engineering*, 144(1):51–60, Février 1997.

[Carriero et Gelernter, 1990] Nicholas Carriero et David Gelernter. *How to Write Parallel Programs*. The MIT Press, Cambridge, MA, 1990.

[Cassandra *et al.*, 1994] Anthony R. Cassandra, Leslie Pack Kaelbling et Michael L. Littman. Acting optimally in partially observable stochastic domains. Dans *AAAI-94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, 1994.

[Chadès *et al.*, 2001] I. Chadès, B. Scherrer et F. Charpillet. A Heuristic Approach for Solving Decentralized-POMDP: Assessment on the Pursuit Problem. Rapport de recherche, LORIA - UMR 7503, Mai 2001.

[Chaib-draa, 1999] Brahim Chaib-draa. Agents et systèmes multiagents. Notes de cours, Département d'informatique, Faculté Des Sciences et de Génie, Université Laval, Québec, Novembre 1999. http://www.damas.ift.ulaval.ca/~chaib/cours.pdf.

[Chavez et Maes, 1996] A. Chavez et P. Maes. Kasbah: An agent marketplace for buying and selling goods. Dans *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, London, Avril 1996.

[Chen et Sycara, 1998] Liren Chen et Katia Sycara. WebMate: A Personal Agent for Browsing and Searching. Dans *Proceedings of the 2nd International Conference on Autonomous Agents (AGENTS-98)*, rédacteurs Katia P. Sycara et Michael Wooldridge, New York, Mai 1998. ACM Press.

[Conway, 1963] M. E. Conway. A multiprocessor system design. Dans *Proceedings AFIPS Fall Joint Computer Conference*, volume 24, pages 139–146. AFIPS Press, 1963.

[Cosnard et Trystram, 1993] Michel Cosnard et Denis Trystram. *Algorithmes et architectures parallèles*. InterEditions, Paris, 1993.

[Currie et Tate, 1985] K. Currie et A. Tate. O-Plan — Control in the Open Planning Architecture. Dans *Expert Systems 85*, Cambridge, England, 1985. Cambridge University Press.

[Dagaeff et Chantemargue, 1997] T. Dagaeff et F. Chantemargue. Distributing Agents. Dans *Swiss Workshop on Collaborative and Distributed Systems*, Lausanne, Switzerland, Mai 1997.

[Dagum et Menon, 1998] Leonardo Dagum et Ramesh Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science & Engineering*, 5(1):46–55, Janvier/Mars 1998.

[Dean et Wellman, 1991] Thomas L. Dean et Michael P. Wellman. *Planning and Control*. Morgan Kaufmann, San Mateo, California, 1991.

[Dedu, 2000] Eugen Dedu. Bibliothèque parallèle pour l'implantation de systèmes multiagent à composantes connexionnistes. Dans *RENPAR : Rencontres Francophones du Parallélisme, des Architectures et des Systèmes*, pages 211–216, Besançon, France, Juin 2000.

[d'Inverno *et al.*, 1996] M. d'Inverno, M. Fisher, A. Lomuscio, M. Luck, M. de Rijke, M. Ryan et M. Wooldridge. Formalisms for multi-agent systems. Dans *First UK Workshop on Foundations of Multi-Agent Systems*, 1996.

[Drogoul et Fresneau, 1998] Alexis Drogoul et Dominique Fresneau. Métaphore du fourragement et modèle d'exploitation collective de l'espace sans communication ni interaction pour des colonies de robots autonomes mobiles. Dans *JFIADSMA'98 : Actes*

des 6e Journées Francophones d'Intelligence Artificielle et Systèmes Multi-Agents, France, 1998.

[Durand *et al.*, 2000] P.-Y. Durand, J. Chanliau, A. Mariot, M. Kessier, J.-P. Thomesse, L. Romary, F. Charpillet et R. Hervy. Telemedecine and Dialysis. Dans *XVII Congreso argentino de control automatico, Buenos Aires*, 2000.

[Durfee et Montgomery, 1989] E. Durfee et T. Montgomery. MICE: A Flexible Testbed for Intelligent Coordination Experiments. Dans *Proceedings of the Ninth Workshop on Distributed AI*, pages 25–40, Rosario, Washington, Septembre 1989.

[Dutech *et al.*, 2001] A. Dutech, O. Buffet et F. Charpillet. Multi-Agent Systems by Incremental Gradient Reinforcement Learning. Dans *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, 2001. A paraître.

[Dutech, 1999] A. Dutech. Apprentissage d'environnement : approches cognitives et comportementales. Thèse de Doctorat, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, Février 1999.

[Epstein et Axtell, 1996] J. M. Epstein et R. Axtell. *Growing Artificial Societies*. The MIT Press, Cambridge, MA, 1996.

[Feautrier, 1995] Paul Feautrier. Compiling for massively parallel architectures: a perspective. *Microprocessing and Microprogramming*, pages 425–439, 1995.

[Feautrier, 1996] Paul Feautrier. Distribution automatique des données et des calculs. *TSI : Technique et Science Informatiques*, 15(5):529–557, 1996.

[Ferber et Müller, 1996] Jacques Ferber et Jean-Pierre Müller. Influences and Reaction: a Model of Situated Multiagent Systems. Dans *ICMAS-96: Proceedings of the Second International Conference on Multi-Agent Systems*, Japan, 1996.

[Ferber, 1995] Jacques Ferber. *Les Systèmes Multi-Agents : Vers une intelligence collective*. InterEditions, 1995.

[Ferber, 1997] Jacques Ferber. Les systèmes multi-agents : un aperçu général. *TSI : Technique et Science Informatiques*, 16(8), 1997.

[Fianyo *et al.*, 1997] Edem Fianyo, Jean-Pierre Treuil et Yves Demazeau. Simulation de l'exploitation des périmètres irrigués : étude sur la coordination réactive par apprentissage dans l'utilisation d'une ressource. Dans *JFIADSMA'97 : Actes des 5e Journées Francophones d'Intelligence Artificielle et Systèmes Multi-Agents*, France, 1997.

[Fianyo *et al.*, 1998] Edem Fianyo, Jean-Pierre Treuil, Edith Perrier et Yves Demazeau. Multi-agent Architecture Integrating Heterogeneous Models of Dynamical Processes: the Representation of Time. Dans *MABS'98: Proceedings of the First International Workshop of Multi-Agent Systems and Agent-Based Simulation*, volume 1534 de *LNAI: Lecture Notes in Artificial Intelligence*, Paris-France, 1998.

[Fikes et Nilsson, 1971] R. E. Fikes et N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Flynn, 1972] M. Flynn. Some Computer Organisations and theis Effectiveness. *IEEE Transactions on Computers*, pages 948–960, 1972.

[Franklin et Graesser, 1996] S. Franklin et A. Graesser. Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. Dans *Proceedings of the Third International Workshop Agent Theories, Architectures, and Languages*, pages 21–35. Springer-Verlag, 1996.

[Gagné *et al.*, 1993] D. Gagné, G. Nault, A. Garant et J. Desbiens. Aurora: A Multi-Agent Prototype Modelling Crew Interpersonal Communication Network. Dans *Proceedings of the Department of National Defence Workshop on Knowledge Base Systems/Robotics*, Ottawa, Ontario, 1993.

[Gamboa Dos Santos, 1995] Carlos Gamboa Dos Santos. Analyse comparée de bibliothèques de type "message passing" pour le calcul distribué. Mémoire de DEA, Université Henri Poincaré, Nancy 1, Septembre 1995.

[Georgeff *et al.*, 1999] M. Georgeff, B. Pell, M. Pollack, M. Tambe et M. Wooldridge. The Belief-Desire-Intention Model of Agency. Dans *Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98)*, rédacteurs Jörg Müller, Munindar P. Singh et Anand S. Rao, volume 1555, pages 1–10. Springer-Verlag: Heidelberg, Germany, 1999.

[Germain-Renaud et Sansonnet, 1991] Cécile Germain-Renaud et Jean-Paul Sansonnet. *Les ordinateurs massivement parallèles*. Armand Colin, 1991.

[Ghormley *et al.*, 1998] Douglas P. Ghormley, David Petrou, Steven H. Rodrigues, Amin M. Vahdat et Thomas E. Anderson. GLUnix: A Global Layer Unix for a Network of Workstations. *Software: Practice and Experience*, 28(9):929–961, Juillet 1998.

[Gilbert et Conte, 1995] rédacteurs N. Gilbert et R. Conte. *Artificial Societies: the Computer Simulation of Social Life*. UCL Press, London, 1995.

[Gilbert et Doran, 1994] rédacteurs N. Gilbert et J. Doran. *Simulating societies: The computer simulation of social phenomena*. UCL Press, London, 1994.

[Ginot et Le Page, 1998] V. Ginot et C. Le Page. Mobidyc, a Generic Multi-Agents Simulator for Modeling Populations Dynamics. Dans *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-98, Volume II*, volume 1416 de *LNAI*, pages 805–814, Castellón, Spain, Juin 1998. Springer.

[Goldspink, 2000] Chris Goldspink. Modelling social systems as complex: Towards a social simulation meta-model. *Journal of Artificial Societies and Social Simulation*, 3(2), 2000. http://www.soc.surrey.ac.uk/JASSS/3/2/1.html.

[Graham et Wavish, 1991] Michael Graham et Peter Wavish. Simulating and Implementing Agents and Multiple Agent Systems. Dans *Proceedings of the 1991 European Simulation Multiconference (ESM'91)*, rédacteur Erik Mosekilde, pages 226–231, Copenhagen, Denmark, Juin 1991.

[Grand et Cliff, 1998] Stephen Grand et Dave Cliff. Creatures: Entertainment Software Agents with Artifical Life. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):39–57, 1998.

[Hayes-Roth *et al.*, 1989] Barbara Hayes-Roth, Micheal Hewett, Richard Washington, Rattikorn Hewett et Adam Seiver. Distributing Intelligence within an Individual.

Dans *Distributed Artificial Intelligence*, rédacteurs Les Gasser et Michael N. Huhns, volume 2 de *Research Notes in Artificial Intelligence*, pages 385–412. Pitman, 1989.

[Hewitt *et al.*, 1973] C. Hewitt, P. Bishop et R. Steiger. A universal modular actor formalism for artificial intelligence. Dans *IJCAI-73*, 1973.

[Hilaire *et al.*, 2000] Vincent Hilaire, Abder Koukam, Pablo Gruer et Jean-Pierre Muller. Formal Specification and Prototyping of Multi-agent Systems. Dans *ESAW*, pages 114–127, 2000.

[Horling *et al.*, 2000] B. Horling, V. Lesser et R. Vincent. Multi-Agent System Simulation Framework. Dans *Proceedings of the 16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation*, Lausanne, Switzerland, Août 2000.

[Houck et Agha, 1992] C. Houck et G. Agha. Hal: A high-level actor language and its distributed implementation. Dans *21st International conference on parallel processing (ICCPP'92)*, volume 2, pages 158–165, St. Charles, IL, Août 1992.

[Howard, 1960] R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[Huang *et al.*, 1995] J. Huang, N. R. Jennings et J. Fox. An Agent-based Approach to Health Care Management. *Applied Artificial Intelligence: An International Journal*, 9(4):401–420, 1995.

[Ishida, 1995] Toru Ishida. Parallel, Distributed and Multi-Agent Production Systems. Dans *ICMAS-95: Proceedings of the First International Conference on Multi-Agent Systems*, California, 1995.

[Jennings *et al.*, 1996] N. R. Jennings, J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek et L. Z. Varga. Using ARCHON to develop real-word DAI applications for electricity transportation management and particle accelerator control. *IEEE Expert*, 11(6), 1996.

[Jennings *et al.*, 1998] N. Jennings, K. Sycara et M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1:275–306, 1998.

[JáJá, 1992] Joseph JáJá. *Introduction to Parallel Algorithms*. Addison-Wesley, New York, 1992.

[Karp et Ramachandran, 1990] R. M. Karp et V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. Dans *Handbook of Theoretical Computer Science*, rédacteur J. van Leeuwen, volume A: Algorithms and Complexity, chapitre 17, pages 870–941. Elsevier Science Publishers, 1990.

[Karp, 1987] A. H. Karp. Programming for Parallelism. *IEEE Computer*, 20(9):43–57, 1987.

[Kim et Agha, 1993] W. Kim et G. Agha. *Languages and compilers for parallel computing,* Compilation of a highly parallel actor-based language, pages 1–15. Numéro 757 dans Lecture notes in computer science. Springer-Verlag, 1993.

[Kipp, 1997] Lionel Kipp. Application du parallélisme aux systèmes multi-agents. Mémoire de DEA, Université Henri-Poincaré Nancy 1 et Supélec, 1997.

[Klein et Backstrom, 1991] I. Klein et C. Backstrom. On the planning problem in sequential control. Dans *Proceedings of the 30th IEEE Conference on Decision and Control*, pages 1819–1823, Brighton, England, Décembre 1991.

[Koenig et Simmons, 1996] S. Koenig et R. Simmons. Unsupervised Learning of Probabilistic Models for Robot Navigation. Dans *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

[Konstantas, 1993] D. Konstantas. HYBRID CELL: an implementation of an object based strongly distributed system. Dans *ISADS-93*, Kawasaki, Japan, Mars 1993.

[Labbani-Igbida, 1998] Ouiddad Labbani-Igbida. Contribution à une méthodologie de conception de comportements collectifs émergents dans une colonie de robots miniatures et autonomes. Thèse de Doctorat de l'Université de Franche-Comte, Laboratoire d'Automatique de Besançon, 1998.

[Laroche, 1997] Pierre Laroche. Modélisation stochastique pour la planification en robotique : tendances actuelles. Rapport de recherche, CRIN/CNRS - INRIA Lorraine, 1997.

[Laroche, 2000] Pierre Laroche. Processus Décisionnels de Markov appliqués à la planification sous incertitude. Thèse de Doctorat de l'Université Henri Poincaré - Nancy 1, LORIA - UMR 7503, Janvier 2000.

[Lester, 1993] B. P. Lester. *The art of parallel programming.* Prentice Hall, 1993.

[Littman *et al.*, 1995a] Michael L. Littman, Anthony R. Cassandra et Leslie Pack Kaelbling. Learning Policies for Partially Observable Environments: Scaling up. Dans *Proceedings the 12th International Conference on Machine Learning*, pages 362–370. Morgan Kaufmann, 1995.

[Littman *et al.*, 1995b] Michael L. Littman, Thomas L. Dean et Leslie Pack Kaelbling. On the Complexity of Solving Markov Decision Problems. Dans *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, pages 394–402, Montreal, Québec, Canada, 1995.

[Littman, 1994] Michael L. Littman. The Witness Algorithm: Solving Partially Observable Markov Decision Processes. Rapport Technique CS-94-40, Department of Computer Science, Brown University, Décembre 1994.

[Lottiaux et Morin, 2000] R. Lottiaux et C. Morin. A Cluster Operating System Based on Software COMA Memory Management. Dans *Proc. of second workshop on software distributed shared memory*, Mai 2000.

[Maes, 1994] Pattie Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, Juillet 1994.

[Magnin, 1996] Laurent Magnin. Modélisation et simulation de l'environnement dans les systèmes multi-agents : Application aux robots footballeurs. Thèse de Doctorat de l'Université Paris VI, Institut Blaise Pascal, 1996.

[McCallum, 1995] Andrew Kachites McCallum. Reinforcement Learning with Selective Perception and Hidden State. PhD thesis, University of Rochester, Computer Science Department, New York, 1995.

[Merz *et al.*, 1997] M. Merz, B. Liberman et W. Lamersdorf. Using Mobile Agents to Support Interorganizational Workflow Management. *Applied Artificial Intelligence*, 11(6):551–572, 1997.

[Miglino *et al.*, 1996] O. Miglino, H. Lund et S. Nolfi. Evolving Mobile Robots in Simulated and Real Environments. *Artificial Life*, 2:417–434, 1996.

[Minar *et al.*, 1996] Nelson Minar, Roger Burkhart, Chris Langton et Manor Askenazi. The Swarm Simulation System: A Toolkit For Building Multi-Agent Simulations. Rapport Technique 96-06-042, Santa Fe Institute, 1996. Working Paper http://www.santafe.edu/projects/swarm/overview.ps.

[Montesello *et al.*, 1998] F. Montesello, A. D'angelo, C. Ferrari et E. Paggello. Implicit Coordination in a Multi-Agent System using a Behavior-based Approach. Dans *RoboCup Papers: ICRA-98 and DARS-98*, 1998.

[Morin *et al.*, 2000] C. Morin, R. Lottiaux et A.-M. Kermarrec. High Availability of the Memory Hierarchy in a Cluster. Dans *Proc. of the 19th IEEE Symposium on Reliable Distributed Systems*, 2000.

[Moulin et Chaib-draa, 1996] Bernard Moulin et Brahim Chaib-draa. An Overview of Distributed Artificial Intelligence. Dans *Foundations of Distributed Artificial Intelligence*, rédacteurs Greg O'Hare et Nick Jennings, chapitre 1. John Wiley and Sons, 1996.

[Müller, 1996a] Jörg P. Müller. A Markovian Model for Interaction among Behavior-Based Agents. Dans *Proceedings of the 2nd International Workshop on Intelligent Agents II: Agent Theories, Architectures, and Languages (ATAL-95)*, volume 1037 de *LNAI*, pages 376–391, Berlin, 1996. Springer.

[Müller, 1996b] Jörg P. Müller. *The Design of Intelligent Agents: A Layered Approach*, volume 1177 de *LNAI: Lecture Notes in Artificial Intelligence*. Springer, 1996.

[Nichols *et al.*, 1996] B. Nichols, D. Buttlar et J. Proulx-Farrel. *Pthreads programming*. O'Reilly & Associates, Inc, 1996.

[Nierstrasz, 1987] O. Nierstrasz. Active objects in HYBRID. Dans *OOPSLA'87*, 1987.

[Pacheco, 1997] P.S. Pacheco. *Parallel programming with MPI*. Morgan Kaufmann, 1997.

[Papadimitriou et Tsitsiklis, 1987] Christos H. Papadimitriou et John N. Tsitsiklis. The Complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–450, Août 1987.

[Parunak *et al.*, 1998] H. Van Dyke Parunak, R. Savit et R. L. Riolo. Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide. Dans *Proceedings of the 1st International Workshop on Multi-Agent Systems and Agent-Based Simulation (MABS-98)*, rédacteurs Jaime S. Sichman, Rosaria Conte et Nigel Gilbert, volume 1534 de *LNAI*, pages 10–25, Berlin, Juillet 4–6 1998. Springer.

[Parunak, 1987] H. Van Dyke Parunak. Manufacturing Experience with the Contract Net. Dans *Distributed Artificial Intelligence*, rédacteur Michael N. Huhns, Research Notes in Artificial Intelligence, chapitre 10, pages 285–310. Pitman, 1987.

[Philips et Bresina, 1991] A. Philips et J. Bresina. NASA TileWorld Manual. Rapport Technique TR-FIA-91-11, NASA Ames Research Center, Code FIA, Moffett Field, CA:, 1991. Technical Report.

[Pollack et Ringuette, 1990] Martha E. Pollack et Marc Ringuette. Introducing the Tileworld: Experimentally evaluating agent archtectures. Dans *Proceedings of the*

Ninth National Conference on Artificial Intelligence, pages 183–189, Boston, MA, 1990.

[Protić *et al.*, 1996] J. Protić, M. Tomasević et V. Milutinović. Distributed Shared Memory: concepts and systems. *IEEE Parallel & Distributed Technology*, 1996.

[Rabiner, 1989] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recoginition. *IEEE Trans. on ASSP*, 77(2):257–286, 1989.

[Resnick, 1994] Mitchel Resnick. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. MIT Press, 1994.

[Reynolds, 1999] Craig Reynolds. Individual-Based Models. 1999. Sony Computer Entertainment America. Research and Development group. http://www.red3d.com/cwr/ibm.html.

[Richter *et al.*, 2000] G. Richter, A. Schmitz et H. Veit. Towards more design flexibility for architectures of autonomous robot control systems. Dans *Proceedings of the 6th Int. Conference on Intelligent Autonomous Systems (IAS-6)*, Venice, Italy, 2000.

[Rouchier, 1998] J. Rouchier. Potlatch and multi-agent system: an analysis of structuring exchanges. Dans *MABS'98: Proceedings of the First International Workshop of Multi-Agent Systems and Agent-Based Simulation*, volume 1534 de *LNAI: Lecture Notes in Artificial Intelligence*, Paris-France, 1998.

[Russell et Norvig, 1995] S. J. Russell et P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[Sacerdoti, 1977] Earl D. Sacerdoti. *A Structure for Plans and Behavior*. Elsevier/North-Holland, Amsterdam, London, New York, 1977.

[Schneider, 1990] A. Schneider. Automatic Parallelization of FORTRAN Code for a Distributed Memory MIMD System. Dans *Proceedings of the International Workshop on Compilers for Parallel Computers*, Paris, France, Décembre 1990.

[Schoonderwoerd *et al.*, 1997] Ruud Schoonderwoerd, Owen Holland et Janet Bruten. Ant-like agents for load balancing in telecommunications networks. Dans *Proceedings of the 1st International Conference on Autonomous Agents*, rédacteurs W. Lewis Johnson et Barbara Hayes-Roth, pages 209–216, New York, Février 1997. ACM Press.

[Shiloach et Vishkin, 1981] Y. Shiloach et U. Vishkin. Finding the Maximum, Merging, and Sorting in a Parallel Computation Model. *Journal of Algorithms*, 2(1):88–102, Mars 1981.

[Smallwood et Sondik, 1973] Richard D. Smallwood et Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[Smith, 1980] R. G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.

[Steiglitz et Shapiro, 1998] K. Steiglitz et D. Shapiro. Simulating the Madness of Crowds: Price Bubbles in an Auction-Mediated Robot Market. *Computational Economics*, 12:35–59, 1998.

[Stone et Veloso, 1997] Peter Stone et Manuela Veloso. The CMUnited-97 Simulator Team. Dans *RoboCup-97: Robot Soccer World Cup I*, LNAI: Lecture Notes in Artificial Intelligence, 1997.

[Theodoropoulos et Logan, 1999] Georgios Theodoropoulos et Brian Logan. A Framework For The Distributed Simulation Of Agent-Based Systems. Dans *Proceedings of the 13th European Simulation Multiconference (ESM'99)*, Warsaw, Poland, Juin 1999.

[Toomey *et al.*, 1988] L. J. Toomey, E. C. Plachy, R. G. Scarborough, R. J. Sahulka, J. F. Shaw et A. W. Shannon. IBM Parallel FORTRAN. *IBM Systems Journal*, 27(4):416–435, Novembre 1988.

[Veit et Richter, 1998] Holger Veit et Gernot Richter. A generic architecture for distributed systems of interacting medium-grained agents. Dans *Proceedings of DAPSYS98 Distributed and Parallel Systems*, rédacteurs P. Kacsuk et G. Kotsis, Wien, 1998.

[Vialle *et al.*, 1998] S. Vialle, Y. Lallement et T. Cornu. Design and implementation of a parallel cellular language for MIMD architectures. *Computer languages*, 24(3):125 – 153, 1998.

[Vialle *et al.*, 2000] Stéphane Vialle, Makram Bouzid, Vincent Chevrier et François Charpillet. ParCeL-3: A Parallel Programming Language Based on Concurrent Cells and Multiple Clocks. Dans *Proceedings of the SNPD'00: International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing*, Reims, France, 2000.

[Vialle et Cornu, 1997] S. Vialle et T. Cornu. Parallélisme. Support de cours EPAP du DEA Informatique, Université Henri Poincaré - Nancy 1, France, Janvier 1997.

[Vialle et Dedu, 2000] S. Vialle et E. Dedu. Long parallel algorithm design vs quick parallel implementation. Dans *EWOMP: European Workshop on OpenMP*, pages 145–150, Edinburgh, Scotland, UK, Septembre 2000.

[Vincent *et al.*, 1998] R. Vincent, B. Horling, T. Wagner et V. Lesser. Survivability Simulator for Multi-Agent Adaptive Coordination. Dans *Proceedings of the First International Conference on Web-Based Modeling and Simulation*, 1998.

[Vincent *et al.*, 2000] R. Vincent, B. Horling et V. Lesser. Experiences in Simulating Multi-Agent Systems Using TAEMS. Dans *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS 2000)*, Boston, MA, Juillet 2000.

[Washington, 1996] R. Washington. Incremental Markov-model planning. Dans *Proceedings of TAI-96, Eighth IEEE International Conference on Tools With Artificial Intelligence*, pages 41–47, 1996.

[Weiss, 1999] rédacteur Gerhard Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, MA, 1999.

[Werner et Dyer, 1994] Gregory M. Werner et Michael G. Dyer. Bioland: A Massively Parallel Simulation Environment for Evolving Distributed Forms of Intelligent Behavior. Dans *Massively Parallel Artificial Intelligence*. The MIT Press, 1994.

[Wilkins, 1984] David E. Wilkins. Domain-Independent Planning: Representation and Plan Generation. *Artificial Intelligence*, 22(3):269–301, 1984.

[Wolper, 2000] P. Wolper. Systèmes parallèles. Transparents du cours Systèmes parallèles, Université de Liege. Institut Montefiore, Belgique, 2000. http://www.montefiore.ulg.ac.be/ pw/cours/syspar.html.

[Wooldridge et Jennings, 1995] M. Wooldridge et N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[Yasugi *et al.*, 1992] M. Yasugi, S. Matsuoka et A. Yonezawa. ABCL/onEM-4: A new software/hardware architecture for object-oriented concurrent computing on an extended dataflow supercomputer. Dans *ACM international conference on supercomputing*, Washington D.C., Juillet 1992.

[Yonezawa *et al.*, 1986] A. Yonezawa, J. P. Briot et E. Shibayama. Object-oriented concurrent programming in ABCL/1. Dans *OOPSLA'86*, 1986.

[Zeghal, 1993] Karim Zeghal. Un modèle de coordination d'actions réactive appliqué au trafic aérien. Dans *JFIADSMA'93 : Actes des 1ères Journées Francophones d'Intelligence Artificielle et Systèmes Multi-Agents*, France, 1993.

[Zhang et Liu, 1996] Nevin L. Zhang et Wenju Liu. Planning in Stochastic Domains: Problem Characteristics and Approximation. Rapport Technique HKUST-CS96-31, Department of Computer Science, Hong Kong University of Science and Technology, 1996.